3GPP TSG CN Plenary Meeting #12 Tdoc NP-010398
Stockholm, Sweden, 13th - 15th June 2001

3GPP TS 29.198-4 V2.0.0 (2001-06)

Technical Specification

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

Keywords

UMTS, API, OSA

3GPP

Postal address

3GPP support office address

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

Internet

http://www.3gpp.org

Contents

Foreword ... 6
Introduction ... 6
1 Scope ... 7
2 References ... 7
3 Definitions and abbreviations ... 8
3.1 Definitions ... 8
3.2 Abbreviations ... 8
4 Call Control SCF ... 8
5 The Service Interface Specifications ... 8
5.1 Interface Specification Format ... 8
5.1.1 Interface Class ... 8
5.1.2 Method descriptions ... 9
5.1.3 Parameter descriptions ... 9
5.1.4 State Model ... 9
5.2 Base Interface ... 9
5.2.1 Interface Class IpInterface ... 9
5.3 Service Interfaces ... 9
5.3.1 Overview ... 9
5.4 Generic Service Interface ... 9
5.4.1 Interface Class IpService ... 9
6 Generic Call Control Service ... 11
6.1 Sequence Diagrams ... 11
6.1.1 Additional Callbacks ... 11
6.1.2 Alarm Call ... 12
6.1.3 Application Initiated Call ... 14
6.1.4 Call Barring 1 ... 16
6.1.5 Number Translation 1 ... 18
6.1.6 Number Translation 1 (with callbacks) ... 20
6.1.7 Number Translation 2 ... 22
6.1.8 Number Translation 3 ... 24
6.1.9 Number Translation 4 ... 26
6.1.10 Number Translation5 ... 28
6.1.11 Prepaid ... 29
6.1.12 Pre-Paid with Advice of Charge (AoC) ... 31
6.2 Class Diagrams ... 34
6.3 Generic Call Control Service Interface Classes ... 36
6.3.1 Interface Class IpCallControlManager ... 37
6.3.2 Interface Class IpAppCallControlManager ... 41
6.3.3 Interface Class IpCall ... 43
6.3.4 Interface Class IpAppCall ... 48
6.4 Generic Call Control Service State Transition Diagrams ... 53
6.4.1 State Transition Diagrams for IpCallControlManager ... 53
6.4.1.1 Active State ... 53
6.4.1.2 Notification terminated State ... 54
6.4.2 State Transition Diagrams for IpCall ... 54
6.4.2.1 Network Released State ... 54
6.4.2.2 Finished State ... 55
6.4.2.3 Application Released State ... 55
6.4.2.4 No Parties State ... 55
6.4.2.5 Active State ... 55
6.4.2.6 1 Party in Call State ... 55
6.4.2.7 2 Parties in Call State ... 55

6.4.2.8 Routing to Destination(s) State....56
6.5 Generic Call Control Service Properties....57
6.5.1 List of Service Properties....57
6.5.2 Service Property values for the CAMEL Service Environment.....57
6.6 Generic Call Control Data Definitions....59
6.6.1 Generic Call Control Event Notification Data Definitions....59
6.6.2 Generic Call Control Data Definitions....61

7 MultiParty Call Control Service....75
7.1 Sequence Diagrams....75
7.1.1 Application initiated call setup....75
7.1.2 Call Barring 2....76
7.1.3 Complex Card Service....78
7.2 Class Diagrams....81
7.3 MultiParty Call Control Service Interface Classes....83
7.3.1 Interface Class IpMultiPartyCallControlManager....83
7.3.2 Interface Class IpAppMultiPartyCallControlManager....87
7.3.3 Interface Class IpMultiPartyCall....89
7.3.4 Interface Class IpAppMultiPartyCall....94
7.3.5 Interface Class IpCallLeg....97
7.3.6 Interface Class IpAppCallLeg....104
7.4 MultiParty Call Control Service State Transition Diagrams....108
7.4.1 State Transition Diagrams for IpMultiPartyCallControlManager....108
7.4.1.1 Active State....108
7.4.1.2 Interrupted State....108
7.4.1.3 Overview of allowed methods....108
Active 109
7.4.2 State Transition Diagrams for IpMultiPartyCall....109
7.4.2.1 IDLE State....109
7.4.2.2 ACTIVE State....109
7.4.2.3 FAULTY State....110
7.4.2.4 RELEASED State....110
7.4.2.5 Overview of allowed methods....110
7.4.3 State Transition Diagrams for IpCallLeg....110
7.4.3.1 Idle State....111
7.4.3.2 Routing State....111
7.4.3.3 Connected State....111
7.4.3.4 Failed or Disconnected State....111
7.4.3.5 Incoming State....112
7.4.3.6 Progress State....112
7.4.3.7 Alerting State....112
7.4.3.8 Redirected State....112
7.4.3.9 Attached State....112
7.4.3.10 Detached State....112
7.4.3.11 Overview of allowed methods....112
7.5 Multi-Party Call Control Service Properties....114
7.5.1 List of Service Properties....114
7.5.2 Service Property values for the CAMEL Service Environment.....114
7.6 Multi-Party Call Control Data Definitions....116
7.6.1 Event Notification Data Definitions....116
7.6.2 Multi-Party Call Control Data Definitions....116

8 Common Call Control Data Types....126

Annex A (normative): OMG IDL Description of Call Control SCF....137

Annex B (informative): Differences between this draft and 3GPP TS 29.198 R99....138
B.1 Interface IpCallControlManager....138
B.2 Interface IpAppCallControlManager....138
B.3 Interface IpCall....138
B.4 Interface IpAppCall....138

Annex C (informative): Change history 140

Foreword

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

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

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

The present document is part 4 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

Part 1:	Overview	
Part 2:	Common Data Definitions	
Part 3:	Framework	
Part 4:	**Call Control SCF**	
Part 5:	User Interaction SCF	
Part 6:	Mobility SCF	
Part 7:	Terminal Capabilities SCF	
Part 8:	Data Session Control SCF	
Part 9:	Generic Messaging SCF	(not part of 3GPP Release 4)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 4)
Part 11:	Account Management SCF	
Part 12:	Charging SCF	

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

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

1 Scope

The present document is Part 4 of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA).

The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.127 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present document specifies the Call Control Service Capability Feature (SCF) aspects of the interface. All aspects of the Call Control SCF are defined here, these being:

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

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

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

2 References

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

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document.*

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

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

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

[4] 3GPP TS 22.002: "Circuit Bearer Services Supported by a PLMN".

[5] ISO-4217:1995: "".

[6] 3GPP TS 24.002: "GSM-UMTS Public Land Mobile Network (PLMN) Access Reference Configuration".

[7] 3GPP TS 22.003: "Circuit Teleservices supported by a Public Land Mobile Network (PLMN)".

3 Definitions and abbreviations

3.1 Definitions

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

3.2 Abbreviations

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

4 Call Control SCF

Two flavours of Call Control (CC) APIs have been included in Rel.4. These are the Generic Call Control (GCC) and the Multi-Party Call Control (MPCC). The GCC is the same API as was already present in the Release 99 specification (TS 29.198 v3.3.0) and is in principle able to satisfy the requirements on CC APIs for Release 4.

However, the joint work between 3GPP CN5, ETSI SPAN12 and the Parlay CC Working group with collaboration from JAIN has been focussed on the MPCC API. A number of improvements on CC functionality have been made and are reflected in this API. For this it was necessary to break the inheritance that previously existed between GCC and MPCC.

The joint CC group has furthermore decided that the MPCC is to be considered as the future base CC family and the technical work will not be continued on GCC. Errors or technical flaws will of course be corrected.

The following clauses describe each aspect of the CC Service Capability Feature (SCF).

The order is as follows:

- The Sequence diagrams give the reader a practical idea of how each of the SCF is implemented.
- The Class relationships clause shows how each of the interfaces applicable to the SCF, relate to one another.
- The Interface specification clause describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show transition between states in the SCF. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.~~the progression of internal processes either in the application, or Gateway.~~
- The Data definitions clause show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification (29.198-2).

5 The Service Interface Specifications

5.1 Interface Specification Format

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

5.1.1 Interface Class

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

5.1.2 Method descriptions

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

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

5.1.3 Parameter descriptions

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

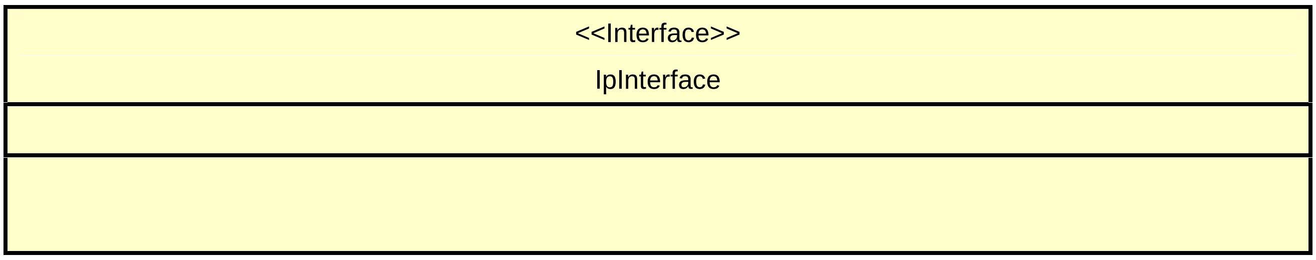
5.1.4 State Model

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

5.2 Base Interface

5.2.1 Interface Class IpInterface

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



5.3 Service Interfaces

5.3.1 Overview

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

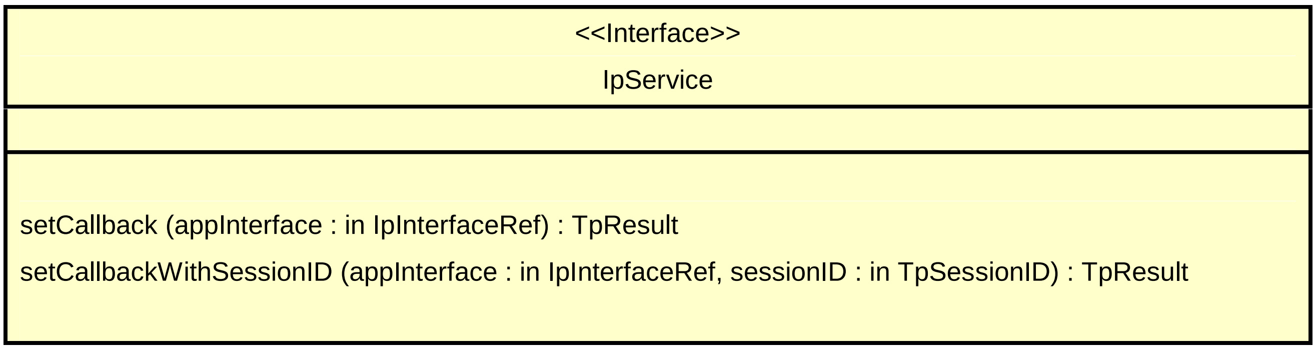
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

5.4 Generic Service Interface

5.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.


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

Method
setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application. It is not allowed to invoke this method on an interface that uses SessionID's.

Parameters

appInterface : in IpInterfaceRef

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

Raises

TpCommonExceptions

Method
setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg. It is not allowed to invoke this method on an interface that does not uses SessionID's.

Parameters

appInterface : in IpInterfaceRef

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

sessionID : in TpSessionID

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

Raises

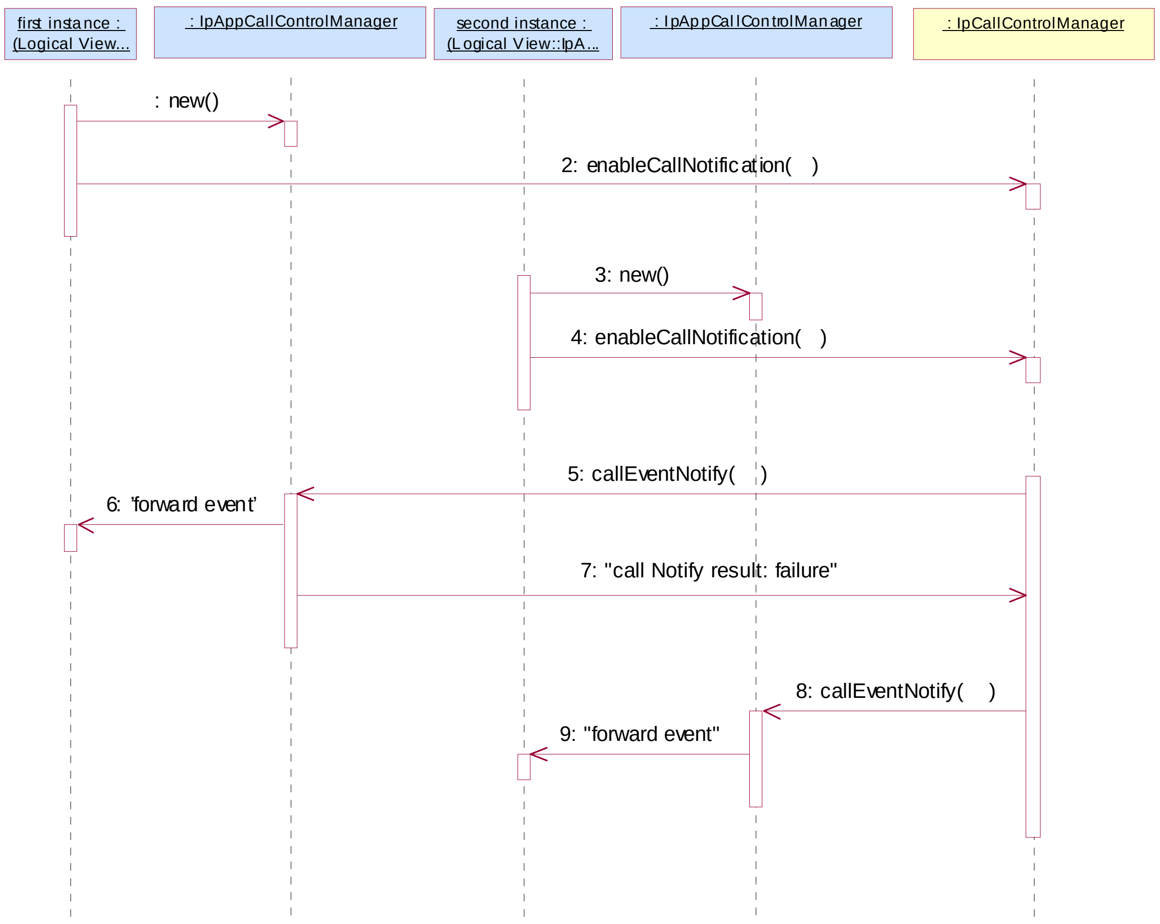
TpCommonExceptions, P_INVALID_SESSION_ID

6 Generic Call Control Service

6.1 Sequence Diagrams

6.1.1 Additional Callbacks

The following sequence diagram shows how an application can register two call back interfaces for the same set of events. If one of the call backs can not be used, e.g., because the application crashed, the other call back interface is used instead.


first instance : (Logical View...
: IpAppCallControlManager
second instance : (Logical View::IpA...
: IpAppCallControlManager
: IpCallControlManager
: new()
2: enableCallNotification( )
3: new()
4: enableCallNotification( )
5: callEventNotify( )
6: 'forward event'
7: "call Notify result: failure"
8: callEventNotify( )
9: "forward event"

1: The first instance of the application is started on node 1. The application creates a new IpAppCallControlManager to handle callbacks for this first instance of the logic.

2: The enableCallNotfication is associated with an applicationID. The call control manager uses the applicationID to decide whether this is the same application.

3: The second instance of the application is started on node 2. The application creates a new IpAppCallControlManager to handle callbacks for this second instance of the logic.

4: The same enableCallNotfication request is sent as for the first instance of the logic. Because both requests are associated with the same application, the second request is not rejected, but the specified callback object is stored as an additional callback.

5: When the trigger occurs one of the first instance of the application is notified. The gateway may have different policies on how to handle additional callbacks, e.g., always first try the first registered or use some kind of round robin scheme.

6: The event is forwarded to the first instance of the logic.

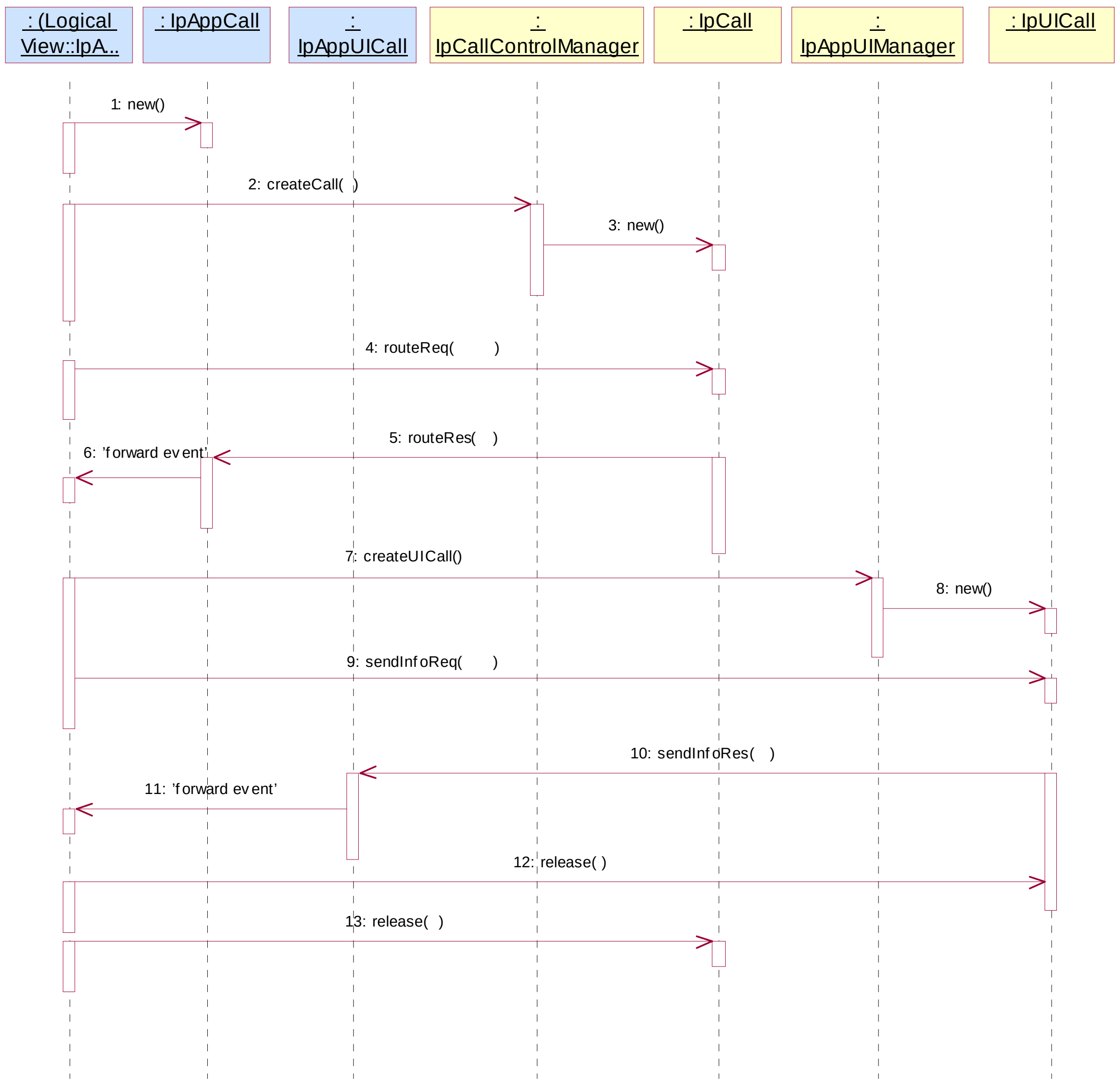
7: When the first instance of the application is overloaded or unavailable this is communicated with an exception to the call control manager.

8: Based on this exception the call control manager will notify another instance of the application (if available).

9: The event is forwarded to the second instance of the logic.

6.1.2 Alarm Call

The following sequence diagram shows a 'reminder message', in the form of an alarm, being delivered to a customer as a result of a trigger from an application. Typically, the application would be set to trigger at a certain time, however, the application could also trigger on events.



1: This message is used to create an object implementing the IpAppCall interface.

2: This message requests the object implementing the IpCallControlManager interface to create an object implementing the IpCall interface.

3: Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met it is created.

4: This message instructs the object implementing the IpCall interface to route the call to the customer destined to receive the 'reminder message'

5: This message passes the result of the call being answered to its callback object.

6: This message is used to forward the previous message to the IpAppLogic.

7: The application requests a new UICall object that is associated with the call object.

8: Assuming all criteria are met, a new UICall object is created by the service.

9: This message instructs the object implementing the IpUICall interface to send the alarm to the customer's call.

10: When the announcement ends this is reported to the call back interface.

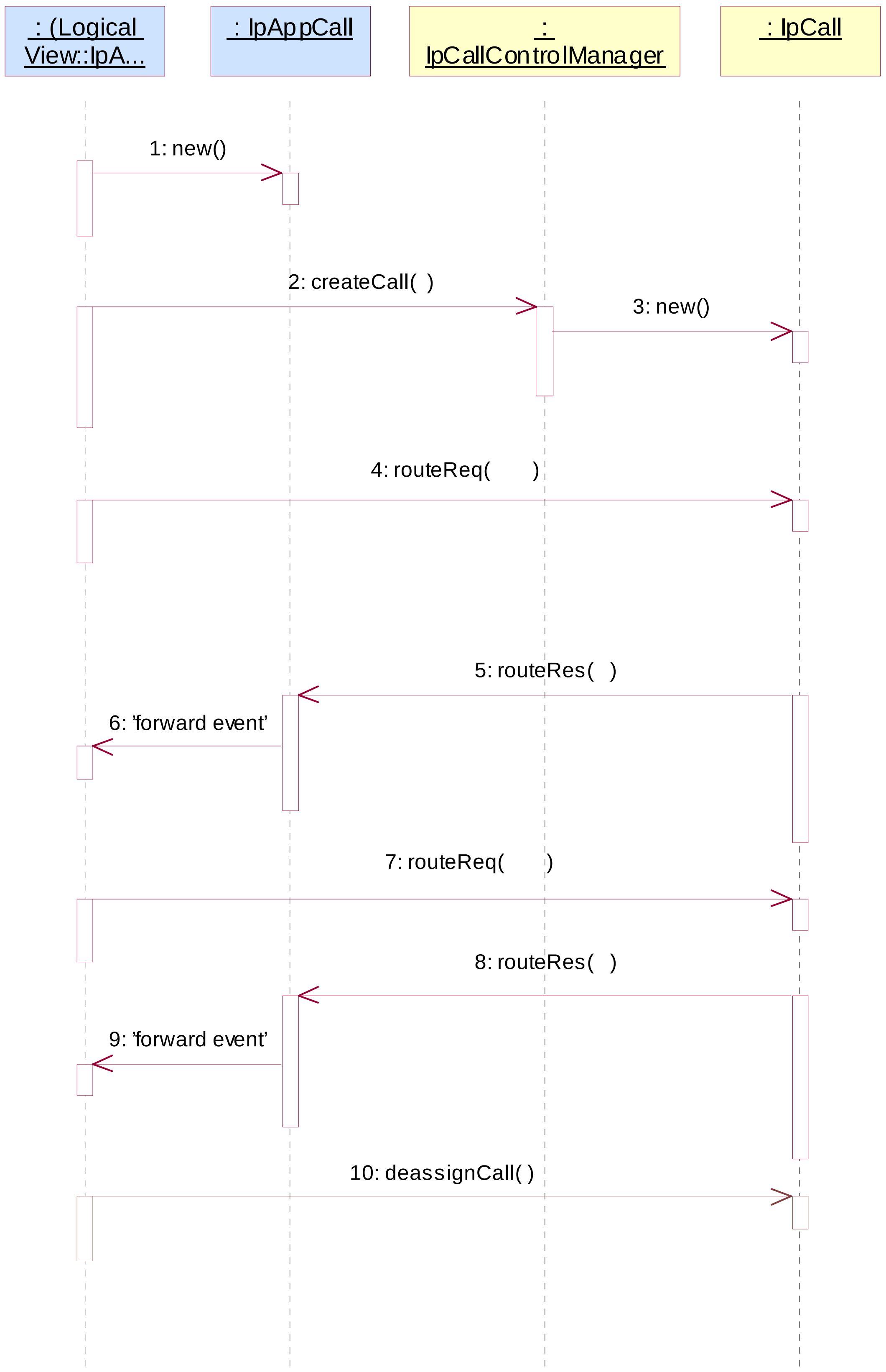
11: The event is forwarded to the application logic.

12: The application releases the UICall object, since no further announcements are required. Alternatively, the application could have indicated P_FINAL_REQUEST in the sendInfoReq in which case the UICall object would have been implicitly released after the announcement was played.

13: The application releases the call and all associated parties.

6.1.3 Application Initiated Call

The following sequence diagram shows an application creating a call between party A and party B. This sequence could be done after a customer has accessed a Web page and selected a name on the page of a person or organisation to talk to.


: (Logical View::IpA...
: IpAppCall
: IpCallControlManager
: IpCall
1: new()
2: createCall( )
3: new()
4: routeReq( )
5: routeRes( )
6: 'forward event'
7: routeReq( )
8: routeRes( )
9: 'forward event'
10: deassignCall( )

1: This message is used to create an object implementing the IpAppCall interface.

2: This message requests the object implementing the IpCallControlManager interface to create an object implementing the IpCall interface.

3: Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, it is created.

4: This message is used to route the call to the A subscriber (origination). In the message the application request response when the A party answers.

5: This message indicates that the A party answered the call.

6: This message forwards the previous message to the application logic.

7: This message is used to route the call to the B-party. Also in this case a response is requested for call answer or failure.

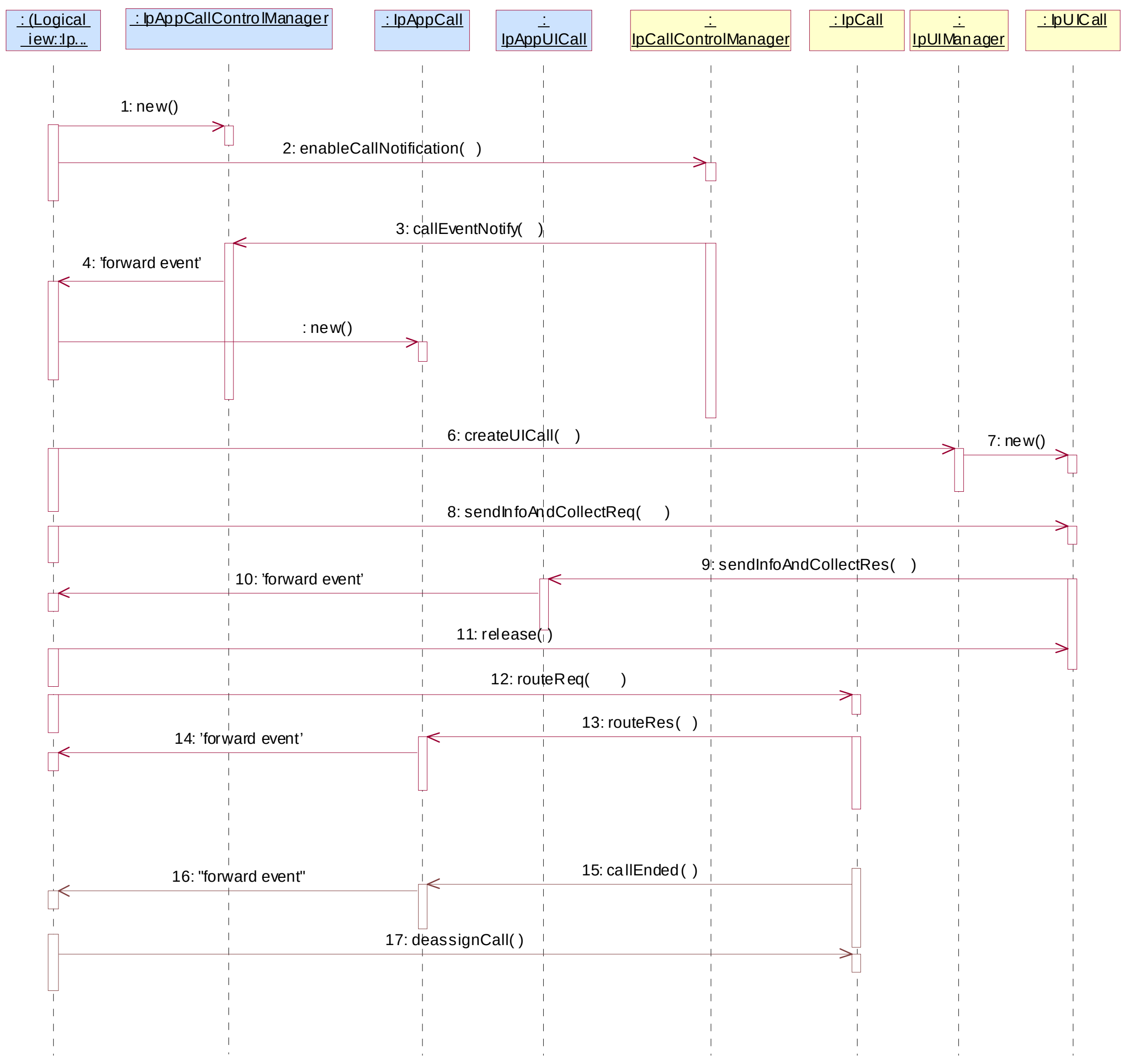
8: This message indicates that the B-party answered the call. The call now has two parties and a speech connection is automatically established between them.

9: This message is used to forward the previous message to the IpAppLogic.

10: Since the application is no longer interested in controlling the call, the application deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.4 Call Barring 1

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the framework. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is accepted and the call is routed to the original called party.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria set, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message is used to create a new UICall object. The reference to the call object is given when creating the UICall.

7: Provided all the criteria are fulfilled, a new UICall object is created.

8: The call barring service dialogue is invoked.

9: The result of the dialogue, which in this case is the PIN code, is returned to its callback object.

10: This message is used to forward the previous message to the IpAppLogic.

11: This message releases the UICall object.

12: Assuming the correct PIN is entered, the call is forward routed to the destination party.

13: This message passes the result of the call being answered to its callback object.

14: This message is used to forward the previous message to the IpAppLogic

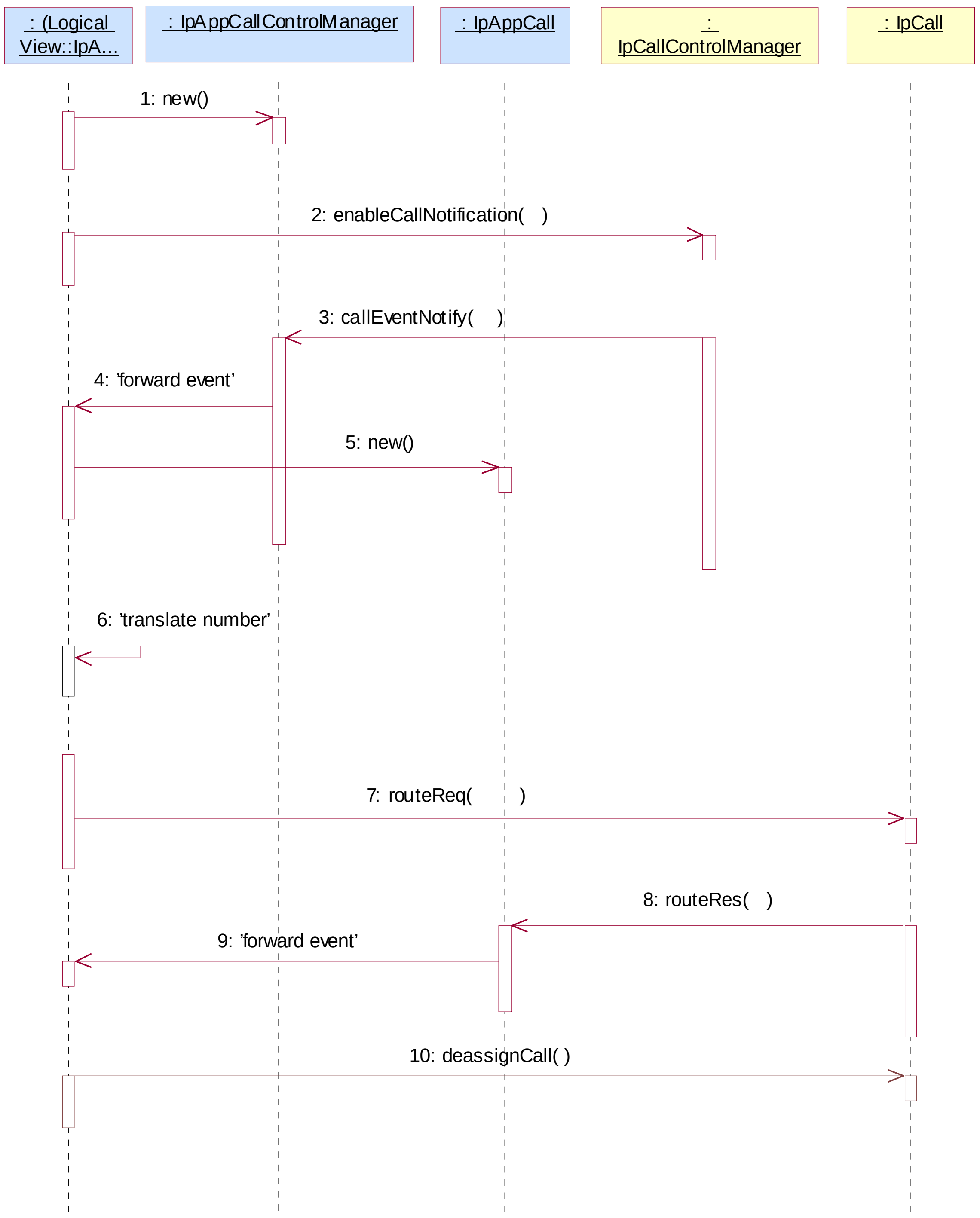
15: When the call is terminated in the network, the application will receive a notification. This notification will always be received when the call is terminated by the network in a normal way, the application does not have to request this event explicitly.

16: The event is forwarded to the application.

17: The application must free the call related resources in the gateway by calling deassignCall.

6.1.5 Number Translation 1

The following sequence diagram shows a simple number translation service, initiated as a result of a prearranged event being received by the framework.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward message 3 to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of message 3.

6: This message invokes the number translation function.

7: The returned translated number is used in message 7 to route the call towards the destination.

8: This message passes the result of the call being answered to its callback object

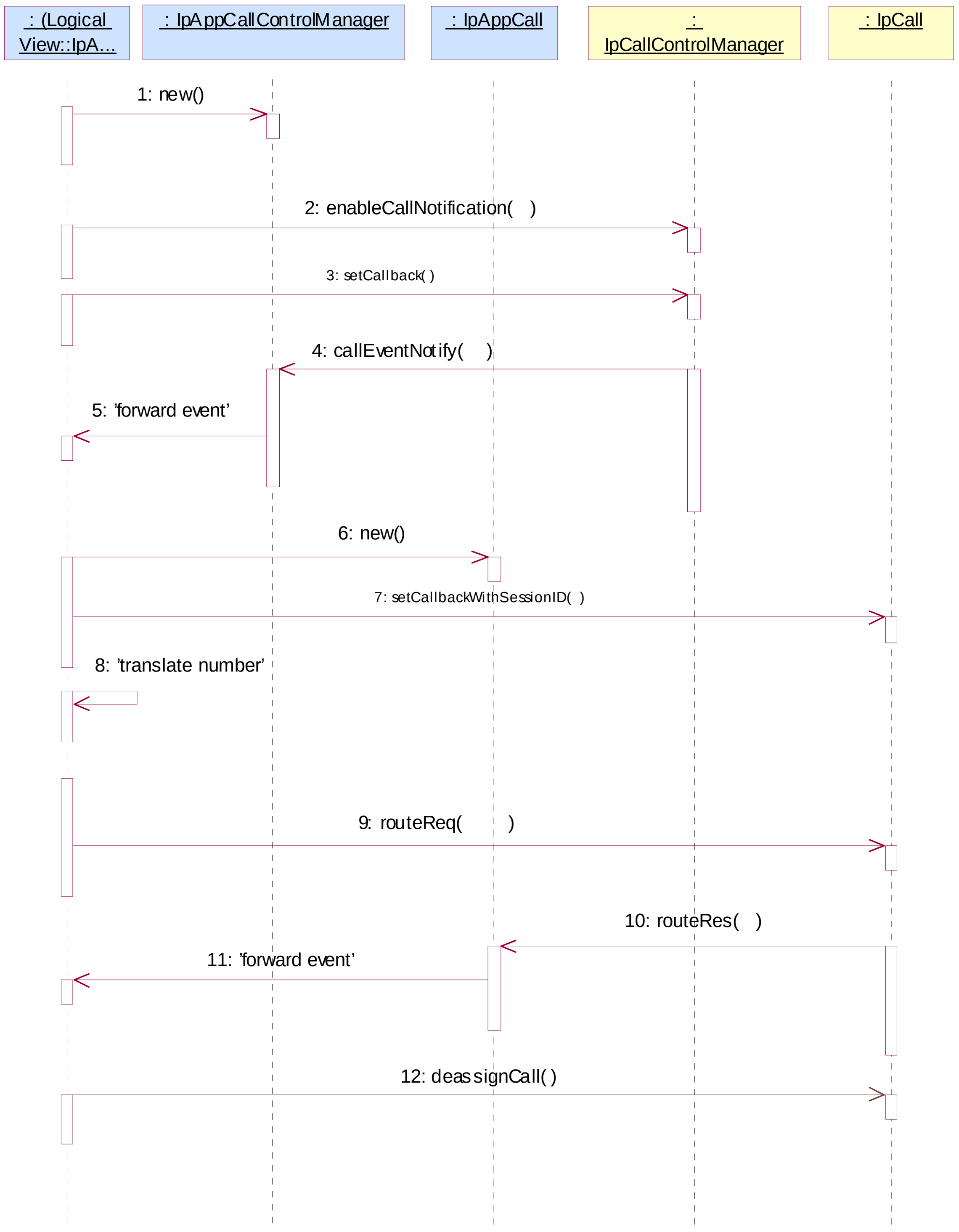
9: This message is used to forward the previous message to the IpAppLogic.

10: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.6 Number Translation 1 (with callbacks)

The following sequence diagram shows a simple number translation service, initiated as a result of a prearranged event being received by the framework.

For illustation, in this sequence the callback references are set explictly. This is optional. All the callbacks references can also be passed in other methods. From an efficiency point of view that is also the preferred method. The rest of the sequences use that mechanism.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message sets the reference of the IpAppCallControlManager object in the CallControlManager. The CallControlManager reports the callEventNotify to referenced object only for enableCallNotification's that do not have a explicit IpAppCallControlManager reference specified in the enableCallNotification.

4: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

5: This message is used to forward message 4 to the IpAppLogic.

6: This message is used by the application to create an object implementing the IpAppCall interface.

7: This message is used to set the reference to the IpAppCall for this call.

8: This message invokes the number translation function.

9: The returned translated number is used in message 7 to route the call towards the destination.

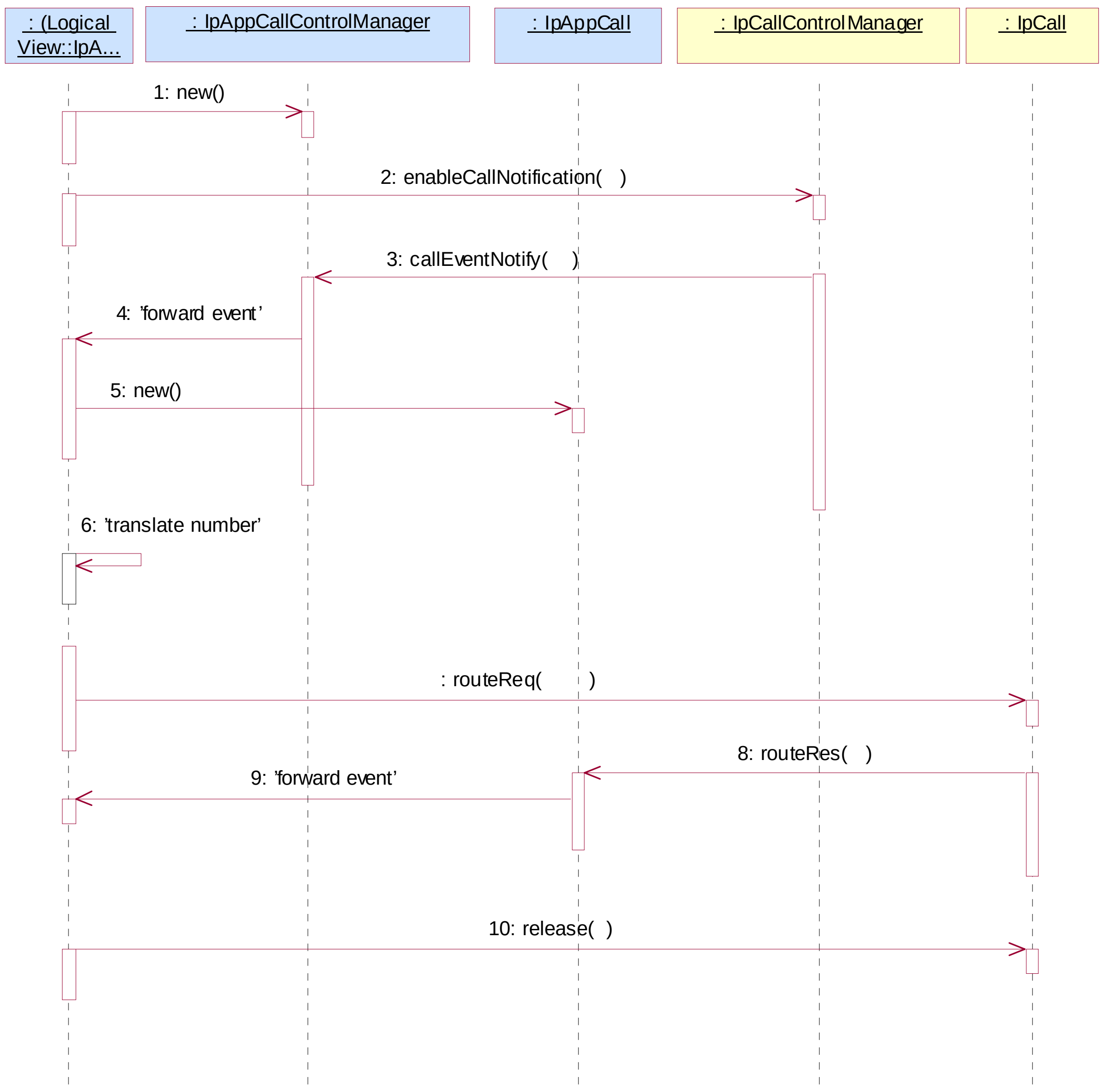
10: This message passes the result of the call being answered to its callback object

11: This message is used to forward the previous message to the IpAppLogic.

12: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.7 Number Translation 2

The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the framework. If the translated number being routed to does not answer or is busy then the call is automatically released.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message invokes the number translation function.

7: The returned translated number is used to route the call towards the destination.

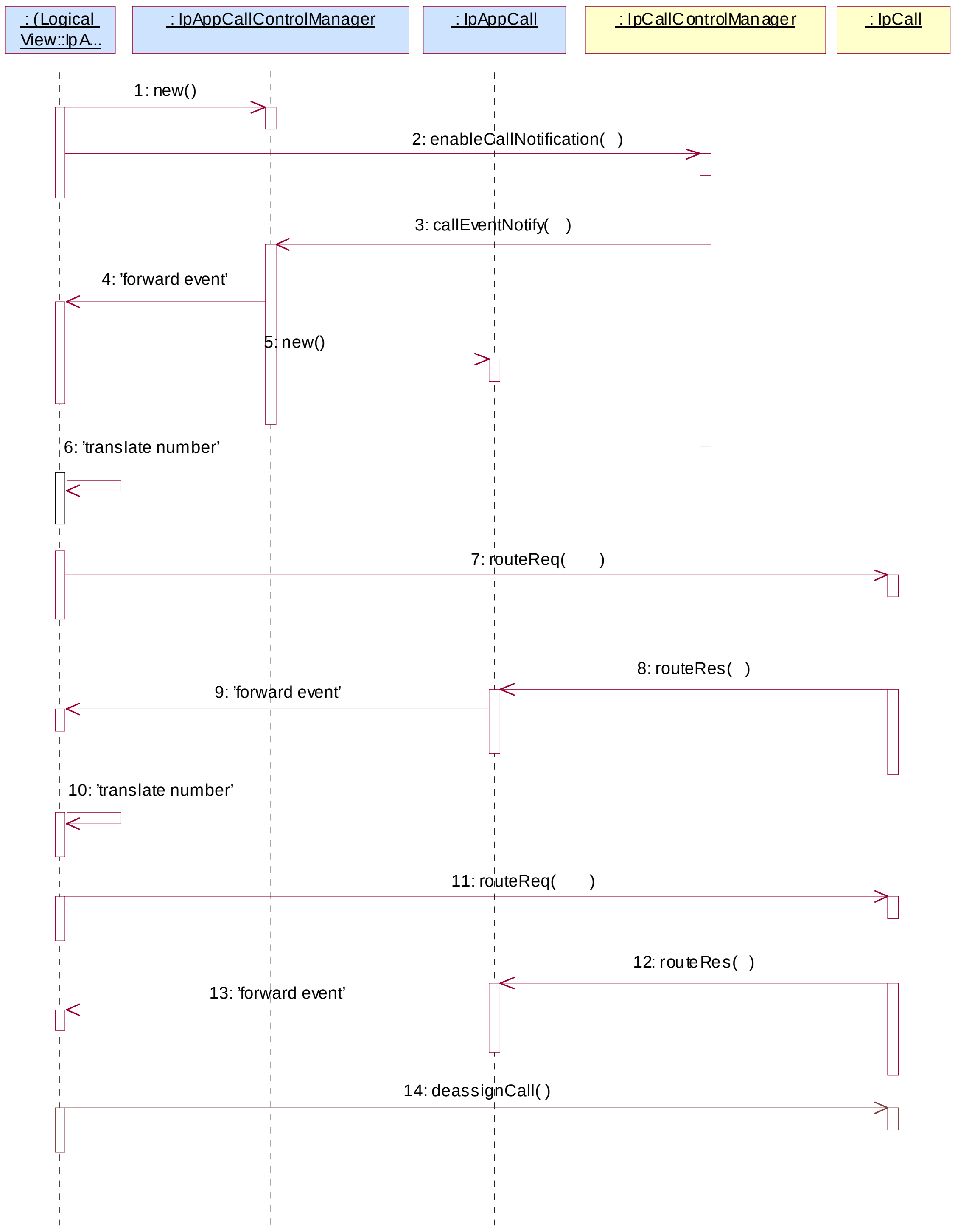
8: Assuming the called party is busy or does not answer, the object implementing the IpCall interface sends a callback in this message, indicating the unavailability of the called party.

9: This message is used to forward the previous message to the IpAppLogic.

10: The application takes the decision to release the call.

6.1.8 Number Translation 3

The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the framework. If the translated number being routed to does not answer or is busy then the call is automatically routed to a voice mailbox.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message invokes the number translation function.

7: The returned translated number is used to route the call towards the destination.

8: Assuming the called party is busy or does not answer, the object implementing the IpCall interface sends a callback, indicating the unavailability of the called party.

9: This message is used to forward the previous message to the IpAppLogic.

10: The application takes the decision to translate the number, but this time the number is translated to a number belonging to a voice mailbox system.

11: This message routes the call towards the voice mailbox.

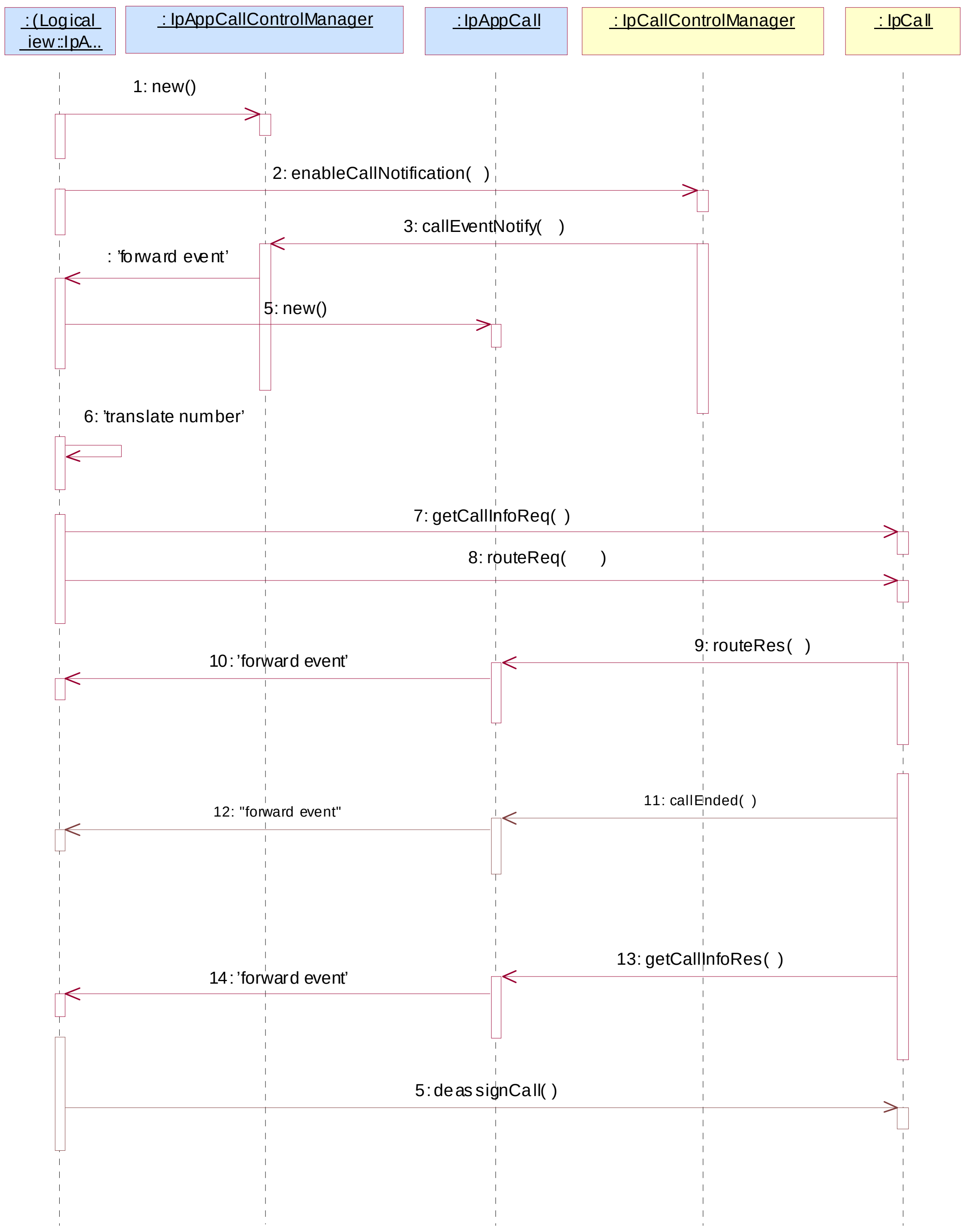
12: This message passes the result of the call being answered to its callback object.

13: This message is used to forward the previous message to the IpAppLogic.

14: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

6.1.9 Number Translation 4

The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the framework. Before the call is routed to the translated number, the application requests for all call related information to be delivered back to the application on completion of the call.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward the previous message to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.

6: This message invokes the number translation function.

7: The application instructs the object implementing the IpCall interface to return all call related information once the call has been released.

8: The returned translated number is used to route the call towards the destination.

9: This message passes the result of the call being answered to its callback object.

10: This message is used to forward the previous message to the IpAppLogic.

11: Towards the end of the call, when one of the parties disconnects, a message (not shown) is directed to the object implementing the IpCall. This causes an event, to be passed to the object implementing the IpAppCall object.

12: This message is used to forward the previous message to the IpAppLogic.

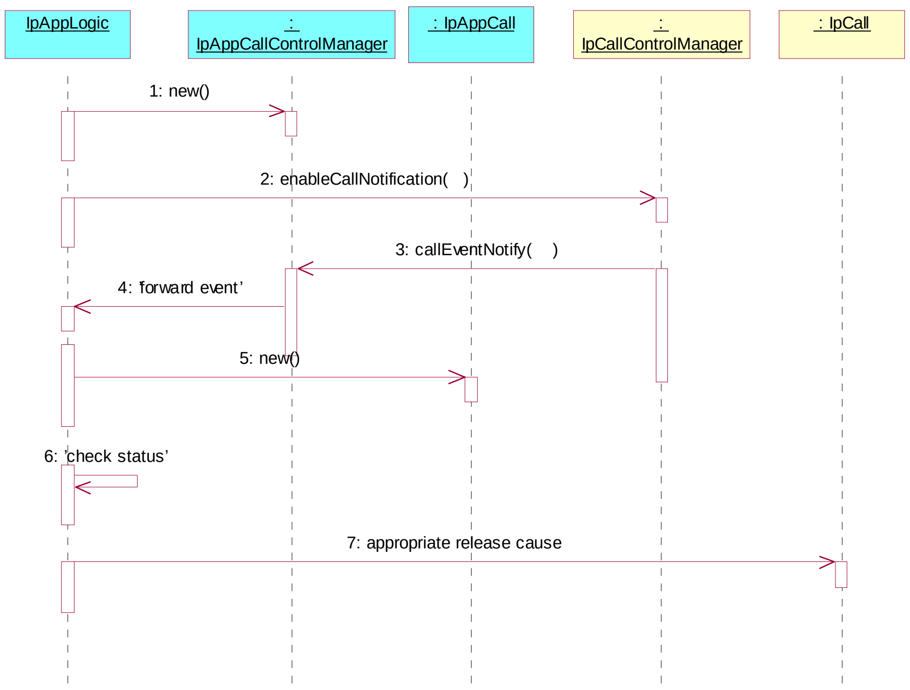
13: The application now waits for the call information to be sent. Now that the call has completed, the object implementing the IpCall interface passes the call information to its callback object.

14: This message is used to forward the previous message to the IpAppLogic

15: After the last information is received, the application deassigns the call. This will free the resources related to this call in the gateway.

6.1.10 Number Translation5

The following sequence diagram shows a simple number translation service which contains a status check function, initiated as a result of a prearranged event being received. In the following sequence, when the application receives an incoming call, it checks the status of the user, and returns a busy code to the calling party.



1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled.

When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.

4: This message is used to forward message 3 to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of message 3.

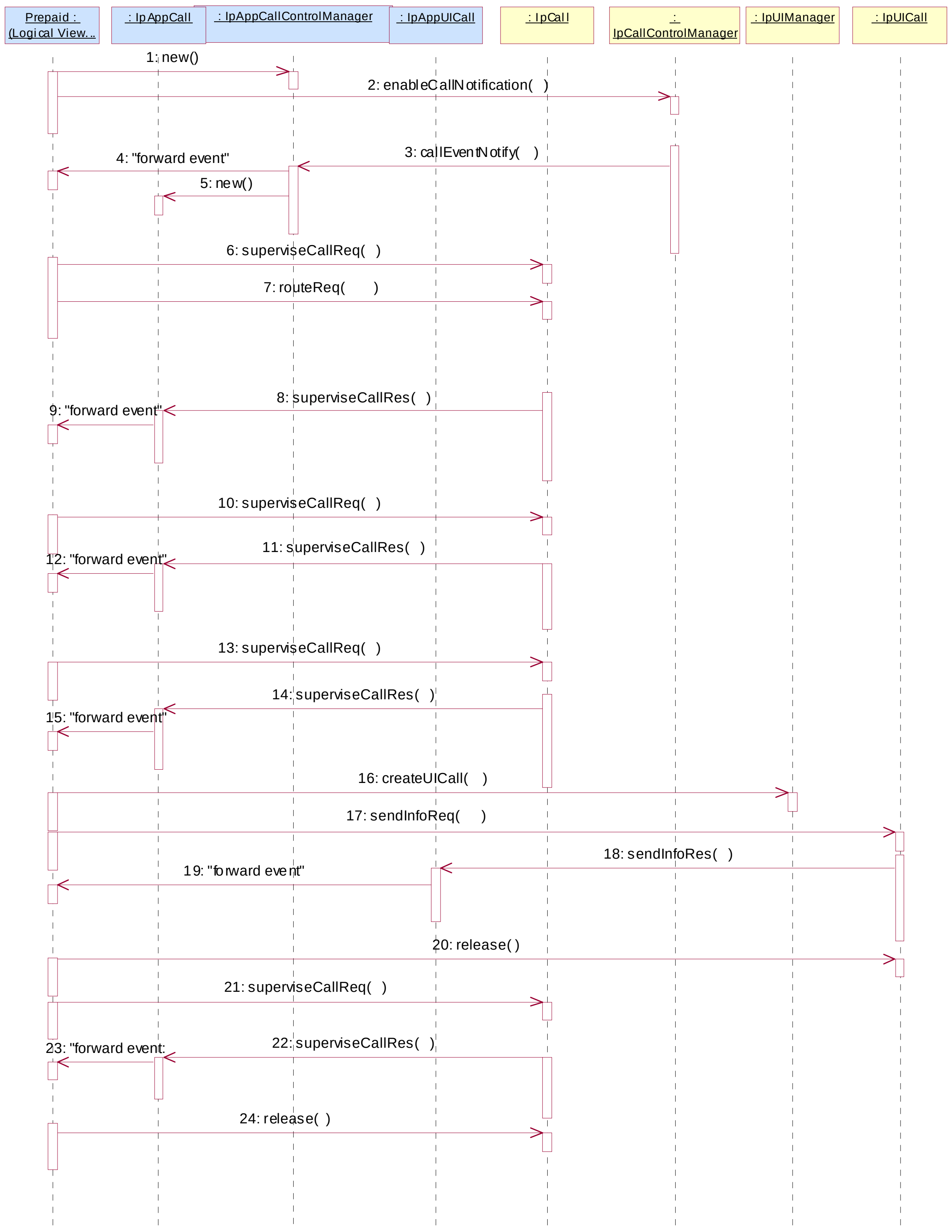
6: This message invokes the status checking function.

7: The application decides to release the call, and sends a release cause to the calling party indicating that the user is busy.

6.1.11 Prepaid

This sequence shows a Pre-paid application.

The subscriber is using a pre-paid card or credit card to pay for the call. The application each time allows a certain timeslice for the call. After the timeslice, a new timeslice can be started or the application can terminate the call. In the following sequence the end-user will received an announcement before his final timeslice.



1: This message is used by the application to create an object implementing the IpAppGenericCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: The incoming call triggers the Pre-Paid Application (PPA).

4: The message is forwarded to the application.

5: A new object on the application side for the Generic Call object is created

6: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.

7: Before continuation of the call, PPA sends all charging information, a possible tariff switch time and the call duration supervision period, towards the GW which forwards it to the network.

8: At the end of each supervision period the application is informed and a new period is started.

9: The message is forwarded to the application.

10: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

11: At the end of each supervision period the application is informed and a new period is started.

12: The message is forwarded to the application.

13: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

14: When the user is almost out of credit an announcement is played to inform about this. The announcement is played only to the leg of the A-party, the B-party will not hear the announcement.

15: The message is forwarded to the application.

16: A new UICall object is created and associated with the controlling leg.

17: An announcement is played to the controlling leg informing the user about the near-expiration of his credit limit. The B-subscriber will not hear the announcement.

18: When the announcement is completed the application is informed.

19: The message is forwarded to the application.

20: The application releases the UICall object.

21: The user does not terminate so the application terminates the call after the next supervision period.

22: The supervision period ends

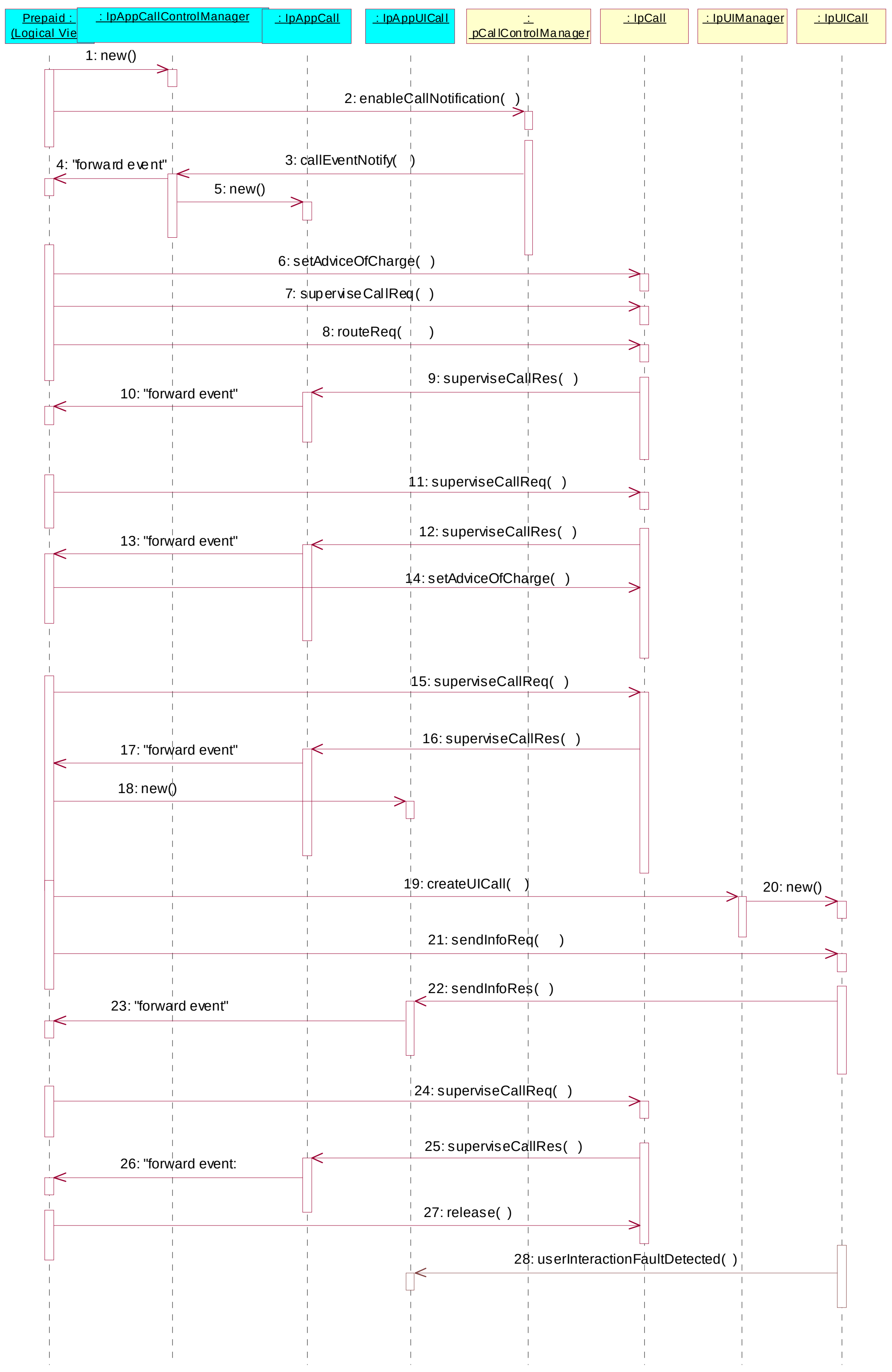
23: The event is forwarded to the logic.

24: The application terminates the call. Since the user interaction is already explicitly terminated no userInteractionFaultDetected is sent to the application.

6.1.12 Pre-Paid with Advice of Charge (AoC)

This sequence shows a Pre-paid application that uses the Advice of Charge feature.

The application will send the charging information before the actual call setup and when during the call the charging changes new information is sent in order to update the end-user. Note: the Advice of Charge feature requires an application in the end-user terminal to display the charges for the call, depending on the information received from the application.


Prepaid : (Logical Vie
: IpAppCallControlManager
: IpAppCall
: IpAppUICall
: pCallControlManager
: IpCall
: IpUIManager
: IpUICall
1: new()
2: enableCallNotification( )
3: callEventNotify( )
4: "forward event"
5: new()
6: setAdviceOfCharge( )
7: superviseCallReq( )
8: routeReq( )
9: superviseCallRes( )
10: "forward event"
11: superviseCallReq( )
12: superviseCallRes( )
13: "forward event"
14: setAdviceOfCharge( )
15: superviseCallReq( )
16: superviseCallRes( )
17: "forward event"
18: new()
19: createUICall( )
20: new()
21: sendInfoReq( )
22: sendInfoRes( )
23: "forward event"
24: superviseCallReq( )
25: superviseCallRes( )
26: "forward event:
27: release( )
28: userInteractionFaultDetected( )

1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: The incoming call triggers the Pre-Paid Application (PPA).

4: The message is forwarded to the application.

5: A new object on the application side for the Call object is created

6: The Pre-Paid Application (PPA) sends the AoC information (e.g the tariff switch time). (it shall be noted the PPA contains ALL the tariff information and knows how to charge the user).

During this call sequence 2 tariff changes take place. The call starts with tariff 1, and at the tariff switch time (e.g., 18:00 hours) switches to tariff 2. The application is not informed about this (but the end-user is!)

7: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.

8: The application requests to route the call to the destination address.

9: At the end of each supervision period the application is informed and a new period is started.

10: The message is forwarded to the application.

11: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

12: At the end of each supervision period the application is informed and a new period is started.

13: The message is forwarded to the application.

14: Before the next tariff switch (e.g., 19:00 hours) the application sends a new AOC with the tarif switch time. Again, at the tariff switch time,the network will send AoC information to the end-user.

15: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.

16: When the user is almost out of credit an announcement is played to inform about this (19-21). The announcement is played only to the leg of the A-party, the B-party will not hear the announcement.

17: The message is forwarded to the application.

18: The application creates a new call back interface for the User interaction messages.

19: A new UI Call object that will handle playing of the announcement needs to be created

20: The Gateway creates a new UI call object that will handle playing of the announcement.

21: With this message the announcement is played to the calling party.

22: The user indicates that the call should continue.

23: The message is forwarded to the application.

24: The user does not terminate so the application terminates the call after the next supervision period.

25: The user is out of credit and the application is informed.

26: The message is forwarded to the application.

27: With this message the application requests to release the call.

28: Terminating the call which has still a UICall object associated will result in a userInteractionFaultDetected. The UICall object is terminated in the gateway and no further communication is possible between the UICall and the application.

6.2 Class Diagrams

The generic call control service consists of two packages, one for the interfaces on the application side and one for interfaces on the service side.

The class diagrams in the following figures show the interfaces that make up the generic call control application package and the generic call control service package. Communication between these packages is indicated with the <<uses>> associations; e.g., the IpCallControlManager interface uses the IpAppGenericCallControlManager , by means of calling callback methods.

This class diagram shows the interfaces of the generic call control application package and their relations to the interfaces of the generic call control service package.

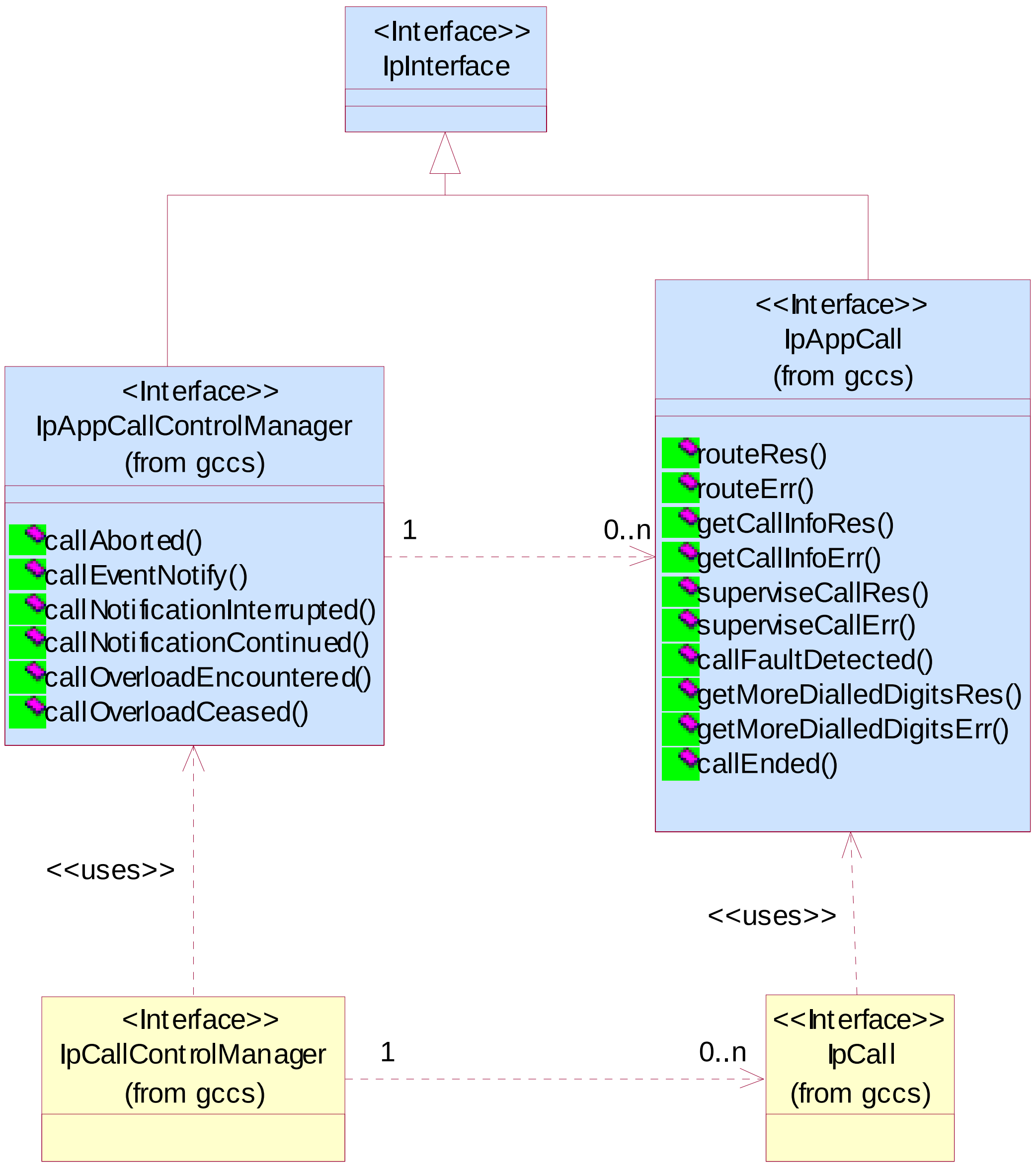


Figure: Application Interfaces

This class diagram shows the interfaces of the generic call control service package.

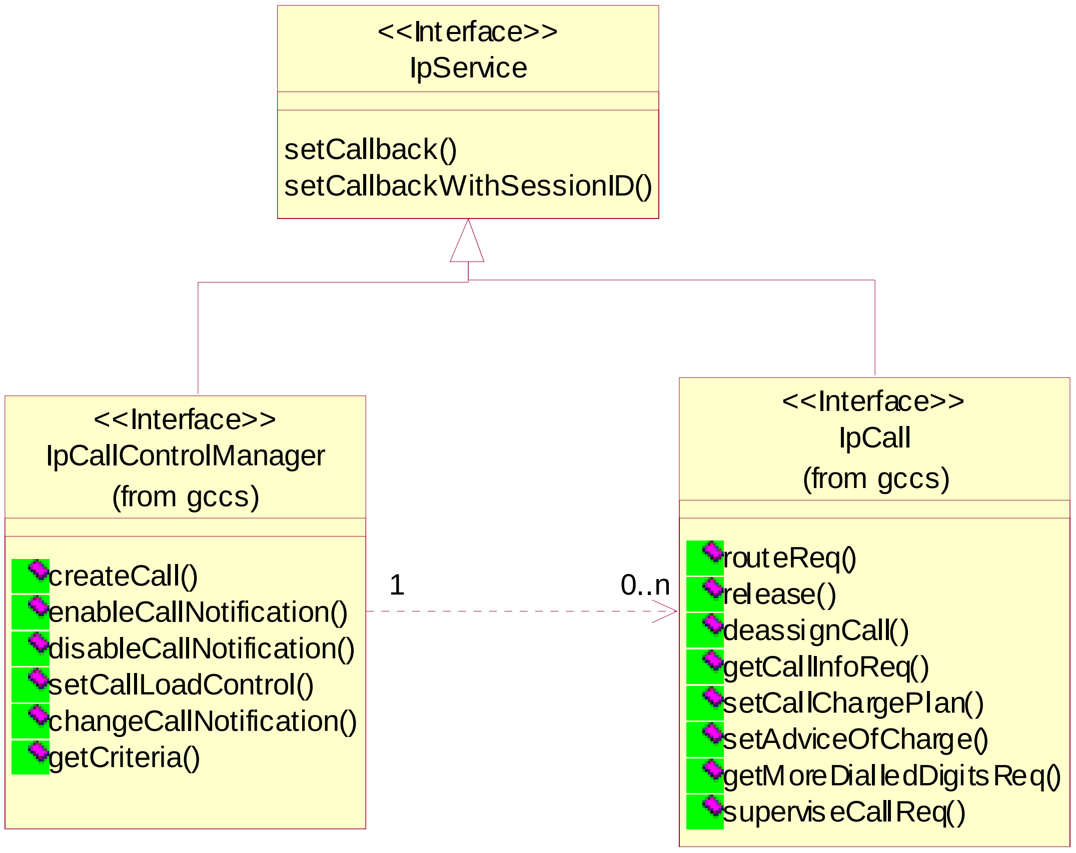


Figure: Service Interfaces

6.3 Generic Call Control Service Interface Classes

The Generic Call Control Service (GCCS) provides the basic call control service for the API. It is based around a third party model, which allows calls to be instantiated from the network and routed through the network.

The GCCS supports enough functionality to allow call routing and call management for today's Intelligent Network (IN) services in the case of a switched telephony network, or equivalent for packet based networks.

It is the intention of the GCCS that it could be readily specialised into call control specifications, for example, ITU-T recommendations H.323, ISUP, Q.931 and Q.2931, ATM Forum specification UNI3.1 and the IETF Session Initiation Protocol, or any other call control technology.

The adopted call model has the following objects. Note that not all of these concepts are used in the generic call.

* a call object. A call is a relation between a number of parties. The call object relates to the entire call view from the application. E.g., the entire call will be released when a release is called on the call. Note that different applications can have different views on the same physical call, e.g., one application for the originating side and another application for the terminating side. The applications will not be aware of each other, all 'communication' between the applications will be by means of network signalling. The API currently does not specify any feature interaction mechanisms.

* a call leg object. The leg object represents a logical association between a call and an address. The relationship includes at least the signalling relation with the party. The relation with the address is only made when the leg is routed. Before that the leg object is IDLE and not yet associated with the address.

* an address. The address logically represents a party in the call.

* a terminal. A terminal is the end-point of the signalling and/or media for a party. This object type is currently not addressed.

The call object is used to establish a relation between a number of parties by creating a leg for each party within the call.

Associated with the signalling relationship represented by the call leg, there may also be a bearer connection (e.g., in the traditional voice only networks) or a number (zero or more) of media channels (in multi-media networks).

A leg can be attached to the call or detached from the call. When the leg is attached, this means that media or bearer channels related to the legs are connected to the media or bearer channels of the other legs that are attached to the same call. I.e., only legs that are attached can ’speak’ to each other. A leg can have a number of states, depending on the signalling received from or sent to the party associated with the leg. Usually there is a limit to the number of legs that are in being routed (i.e., the connection is being established) or connected to the call (i.e., the connection is established). Also, there usually is a limit to the number of legs that can be simultaneously attached to the same call.

Some networks distinguish between controlling and passive legs. By definition the call will be released when the controlling leg is released. All other legs are called passive legs. There can be at most one controlling leg per call. However, there is currently no way the application can influence whether a Leg is controlling or not.

There are two ways for an application to get the control of a call. The application can request to be notified of calls that meet certain criteria. When a call occurs in the network that meets these criteria, the application is notified and can control the call. Some legs will already be associated with the call in this case. Another way is to create a new call from the application.

For the generic call control service, only a subset of the model is used; the API for generic call control does not give explicit access to the legs and the media channels. This is provided by the Multi-Party Call Control Service. Furthermore, the generic call is restricted to two party calls, i.e., only two legs are active at any given time. Active is defined here as ’being routed’ or connected.

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

6.3.1 Interface Class IpCallControlManager

Inherits from: IpService

This interface is the ’service manager’ interface for the Generic Call Control Service. The generic call control manager interface provides the management functions to the generic call control service. The application programmer can use this interface to provide overload control functionality, create call objects and to enable or disable call-related event notifications.

<<Interface>> IpCallControlManager
createCall (appCall : in IpAppCallRef, callReference : out TpCallIdentifierRef) : TpResult enableCallNotification (appCallControlManager : in IpAppCallControlManagerRef, eventCriteria : in TpCallEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult disableCallNotification (assignmentID : in TpAssignmentID) : TpResult setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange, assignmentID : out TpAssignmentIDRef) :

```
    TpResult
changeCallNotification (assignmentID : in TpAssignmentID, eventCriteria : in TpCallEventCriteria) :
    TpResult
getCriteria (eventCriteria : out TpCallEventCriteriaResultSetRef) : TpResult
```

Method
createCall()

This method is used to create a new call object. An IpAppCallControlManager should already have been passed to the IpCallControlManager, otherwise the call control will not be able to report a callAborted()

to the application (the application should invoke setCallback() if it wishes to ensure this).

Parameters

appCall : in IpAppCallRef

Specifies the application interface for callbacks from the call created.

callReference : out TpCallIdentifierRef

Specifies the interface reference and sessionID of the call created.

Raises

TpGCCSException,TpGeneralException

Method
enableCallNotification()

This method is used to enable call notifications so that events can be sent to the application. This is the first step an application has to do to get initial notification of calls happening in the network. When such an event happens, the application will be informed by callEventNotify(). In case the application is interested in other events during the context of a particular call session it has to use the routeReq() method on the call object. The application will get access to the call object when it receives the callEventNotify(). (Note that the enableCallNotification() is not applicable if the call is setup by the application).

The enableCallNotification method is purely intended for applications to indicate their interest to be notified when certain call events take place. It is possible to subscribe to a certain event for a whole range of addresses, e.g. the application can indicate it wishes to be informed when a call is made to any number starting with 800.

If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P_GCCS_INVALID_CRITERIA.The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used and the same CallNotificationType is used.

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. Both notifications will share the same assignmentID. The gateway will always use the most recent callback. In case this most recent callback fails the second most recent is used. In case the enableCallNotification contains no callback, at the moment the application needs to be informed the gateway will use as callback the callback that has been registered by setCallBack().

Parameters

`appCallControlManager : in IpAppCallControlManagerRef`

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the setCallback() method.

`eventCriteria : in TpCallEventCriteria`

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported. Examples of events are "incoming call attempt reported by network", "answer", "no answer", "busy". Individual addresses or address ranges may be specified for destination and/or origination.

`assignmentID : out TpAssignmentIDRef`

Specifies the ID assigned by the generic call control manager interface for this newly-enabled event notification.

Raises

`TpGCCSException,TpGeneralException`

Method

`disableCallNotification()`

This method is used by the application to disable call notifications.

Parameters

`assignmentID : in TpAssignmentID`

Specifies the assignment ID given by the generic call control manager interface when the previous enableNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID. If two callbacks have been registered under this assignment ID both of them will be disabled.

Raises

`TpGCCSException,TpGeneralException`

Method

`setCallLoadControl()`

This method imposes or removes load control on calls made to a particular address range within the generic call control service. The address matching mechanism is similar as defined for TpCallEventCriteria.

Parameters

`duration : in TpDuration`

Specifies the duration for which the load control should be set.

A duration of 0 indicates that the load control should be removed.

A duration of -1 indicates an infinite duration (i.e., until disabled by the application)

A duration of -2 indicates the network default duration.

mechanism : in TpCallLoadControlMechanism

Specifies the load control mechanism to use (for example, admit one call per interval), and any necessary parameters, such as the call admission rate. The contents of this parameter are ignored if the load control duration is set to zero.

treatment : in TpCallTreatment

Specifies the treatment of calls that are not admitted. The contents of this parameter are ignored if the load control duration is set to zero.

addressRange : in TpAddressRange

Specifies the address or address range to which the overload control should be applied or removed.

assignmentID : out TpAssignmentIDRef

Specifies the assignmentID assigned by the gateway to this request. This assignementID can be used to correlate the callOverlloadEncountered and callOverloadCeased methods with the request.

Raises

TpGeneralException,TpGCCSException

Method
changeCallNotification()

This method is used by the application to change the event criteria introduced with enableCallNotification. Any stored criteria associated with the specified assignementID will be replaced with the specified criteria.

Parameters

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic call control manager interface for the event notification. If two call backs have been registered under this assignment ID both of them will be changed.

eventCriteria : in TpCallEventCriteria

Specifies the new set of event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

Raises

TpGeneralException,TpGCCSException

Method
getCriteria()

This method is used by the application to query the event criteria set with enableCallNotification or changeCallNotification.

Parameters

eventCriteria : out TpCallEventCriteriaResultSetRef

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

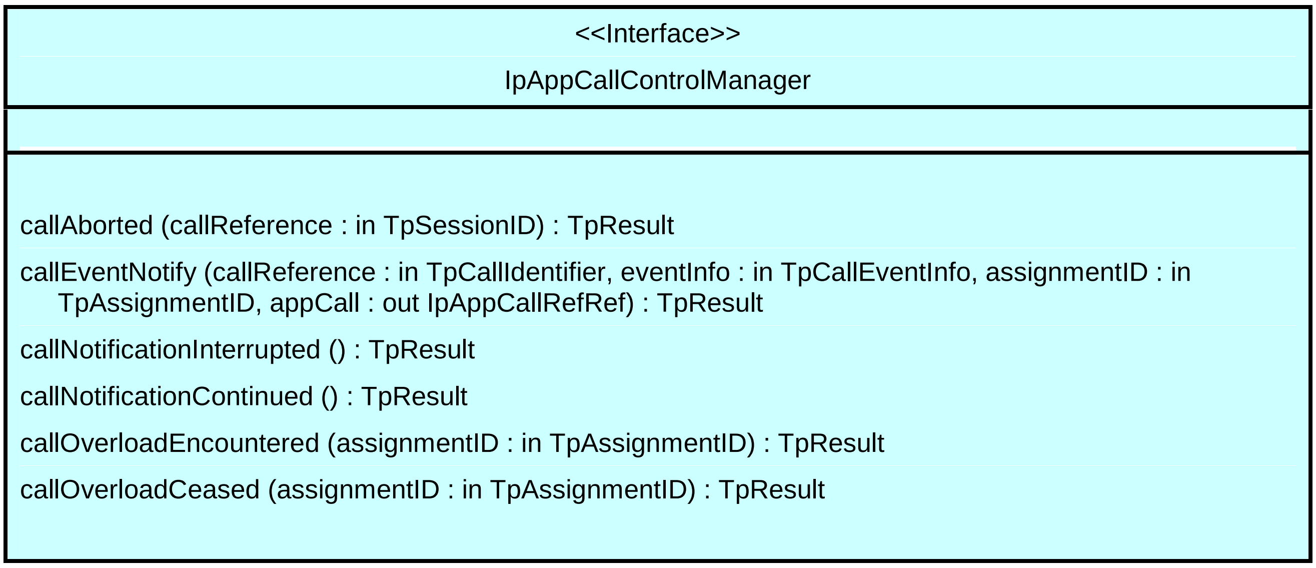
Raises

`TpGeneralException,TpGCCSException`

6.3.2 Interface Class IpAppCallControlManager

Inherits from: IpInterface

The generic call control manager application interface provides the application call control management functions to the generic call control service.



<<Interface>>

IpAppCallControlManager

callAborted (callReference : in TpSessionID) : TpResult

callEventNotify (callReference : in TpCallIdentifier, eventInfo : in TpCallEventInfo, assignmentID : in TpAssignmentID, appCall : out IpAppCallRefRef) : TpResult

callNotificationInterrupted () : TpResult

callNotificationContinued () : TpResult

callOverloadEncountered (assignmentID : in TpAssignmentID) : TpResult

callOverloadCeased (assignmentID : in TpAssignmentID) : TpResult

Method

`callAborted()`

This method indicates to the application that the call object (at the gateway) has aborted or terminated abnormally. No further communication will be possible between the call and application.

Parameters

`callReference : in TpSessionID`

Specifies the sessionID of call that has aborted or terminated abnormally.

Raises

`TpGCCSException,TpGeneralException`

Method

`callEventNotify()`

This method notifies the application of the arrival of a call-related event.

If this method is invoked with a monitor mode of P_MONITOR_MODE_INTERRUPTED, then the APL has control of the call. If the APL does nothing with the call (including its associated legs) within a specified time period (the duration of which forms a part of the service level agreement), then the call in the network shall be released and callEnded() shall be invoked, giving a release cause of 102 (Recovery on timer expiry).

When this method is invoked with a monitor mode of P_MONITOR_MODE_INTERRUPT, the application writer should ensure that no routeReq() is performed until an IpAppCall has been passed to the gateway, either through an explicit setCallback() invocation on the supplied IpCall, or via the return of the callEventNotify() method.

Parameters

`callReference : in TpCallIdentifier`

Specifies the reference to the call interface to which the notification relates. This parameter will be null if the notification is in NOTIFY mode.

`eventInfo : in TpCallEventInfo`

Specifies data associated with this event.

`assignmentID : in TpAssignmentID`

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

`appCall : out IpAppCallRefRef`

Specifies a reference to the application interface which implements the callback interface for the new call. This parameter will be null if the notification is in NOTIFY mode.

Raises

`TpGCCSException,TpGeneralException`

Method

`callNotificationInterrupted()`

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

Note that more permanent failures are reported via the Framework (integrity management).

Parameters

No Parameters were identified for this method

Raises

`TpGCCSException,TpGeneralException`

Method

`callNotificationContinued()`

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

Parameters

No Parameters were identified for this method

Method

`callOverloadEncountered()`

This method indicates that the network has detected overload and may have automatically imposed load control on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

`assignmentID : in TpAssignmentID`

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been encountered.

Raises

`TpGeneralException,TpGCCSException`

Method

`callOverloadCeased()`

This method indicates that the network has detected that the overload has ceased and has automatically removed any load controls on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

`assignmentID : in TpAssignmentID`

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been ceased

Raises

`TpGeneralException,TpGCCSException`

6.3.3 Interface Class IpCall

Inherits from: IpService

The generic Call provides the possibility to control the call routing, to request information from the call, control the charging of the call, to release the call and to supervise the call. It does not give the possibility to control the legs directly and it does not allow control over the media. The first capability is provided by the multi-party call and the latter as well by the multi-media call. The call is limited to two party calls, although it is possible to provide 'follow-on' calls, meaning that the call can be rerouted after the terminating party has disconnected or routing to the terminating party has failed. Basically, this means that at most two legs can be in connected or routing state at any time.

<<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

setAdviceOfCharge (callSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tariffSwitch : in TpDuration) : TpResult

getMoreDialledDigitsReq (callSessionID : in TpSessionID, length : in TpInt32) : TpResult

superviseCallReq (callSessionID : in TpSessionID, time : in TpDuration, treatment : in TpCallSuperviseTreatment) : TpResult

Method
routeReq()

This asynchronous method requests routing of the call to the remote party indicated by the targetAddress.

The extra address information such as originatingAddress is optional. If not present (i.e., the plan is set to P_ADDRESS_PLAN_NOT_PRESENT), the information provided in corresponding addresses from the route is used, otherwise the network or gateway provided numbers will be used.

If this method in invoked, and call reports have been requested, yet no IpAppCall interface has been provided, this method shall throw the P_NO_CALLBACK_ADDRESS_SET exception.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

responseRequested : in TpCallReportRequestSet

Specifies the set of observed events that will result in zero or more routeRes() being generated.

E.g., when both answer and disconnect is monitored the result can be received two times.

If the application wants to control the call (in whatever sense) it shall enable event reports

targetAddress : in TpAddress

Specifies the destination party to which the call leg should be routed.

originatingAddress : in TpAddress

Specifies the address of the originating (calling) party.

originalDestinationAddress : in TpAddress

Specifies the original destination address of the call.

redirectingAddress : in TpAddress

Specifies the address from which the call was last redirected.

appInfo : in TpCallAppInfoSet

Specifies application-related information pertinent to the call (such as alerting method, tele-service type, service identities and interaction indicators).

callLegSessionID : out TpSessionIDRef

Specifies the sessionID assigned by the gateway. This is the sessionID of the implicitly created call leg. The same ID will be returned in the routeRes or Err. This allows the application to correlate the request and the result.

This parameter is only relevant when multiple routeReq() calls are executed in parallel, e.g., in the multi-party call control service.

Raises

TpGCCSException,TpGeneralException

Method
release()

This method requests the release of the call object and associated objects. The call will also be terminated in the network. If the application requested reports to be sent at the end of the call (e.g., by means of getCallInfoReq) these reports will still be sent to the application.

The application should always either release or deassign the call when it is finished with the call, unless a callFaultDetected is received by the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

cause : in TpCallReleaseCause

Specifies the cause of the release.

Raises

TpGCCSException,TpGeneralException

Method
deassignCall()

This method requests that the relationship between the application and the call and associated objects be de-assigned. It leaves the call in progress, however, it purges the specified call object so that the application has no further control of call processing. If a call is de-assigned that has event reports, call information reports or call Leg information reports requested, then these reports will be disabled and any related information discarded.

The application should always either release or deassign the call when it is finished with the call, unless callFaultDetected is received by the application.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

Raises

`TpGCCSException,TpGeneralException`

Method

`getCallInfoReq()`

This asynchronous method requests information associated with the call to be provided at the appropriate time (for example, to calculate charging). This method must be invoked before the call is routed to a target address. Two types of reports can be requested; a final report or intermediate reports.

A final call report is sent when the call is ended. The call object will exist after the call is ended if information is required to be sent to the application at the end of the call. The call information will be sent after any call event reports.

Intermediate reports are received when the destination leg or party terminates or when the call ends. In case the originating party is still available the application can still initiate a follow-on call using routeReq.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`callInfoRequested : in TpCallInfoType`

Specifies the call information that is requested.

Raises

`TpGCCSException,TpGeneralException`

Method

`setCallChargePlan()`

Set an operator specific charge plan for the call. The charge plan must be set before the call is routed to a target address. Depending on the operator the method can also be used to change the charge plan for ongoing calls.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`callChargePlan : in TpCallChargePlan`

Specifies the charge plan to use.

Raises

`TpGCCSException,TpGeneralException`

Method
`setAdviceOfCharge()`

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`aOCInfo : in TpAoCInfo`

Specifies two sets of Advice of Charge parameter.

`tariffSwitch : in TpDuration`

Specifies the tariff switch interval that signifies when the second set of AoC parameters becomes valid.

Raises

`TpGeneralException,TpGCCSException`

Method
`getMoreDialledDigitsReq()`

This asynchronous method requests the call control service to collect further digits and return them to the application. Depending on the administered data, the network may indicate a new call to the gateway if a caller goes off-hook or dialled only a few digits. The application then gets a new call event which contains no digits or only the few dialled digits in the event data.

The application should use this method if it requires more dialled digits, e.g. to perform screening.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`length : in TpInt32`

Specifies the maximum number of digits to collect.

Raises

`TpGeneralException, TpGCCSException`

Method
`superviseCallReq()`

The application calls this method to supervise a call. The application can set a granted connection time for this call. If an application calls this function before it calls a routeReq() or a user interaction function the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`time : in TpDuration`

Specifies the granted time in milliseconds for the connection.

`treatment : in TpCallSuperviseTreatment`

Specifies how the network should react after the granted connection time expired.

Raises

`TpGCCSException,TpGeneralException`

6.3.4 Interface Class IpAppCall

Inherits from: IpInterface

The generic call application interface is implemented by the client application developer and is used to handle call request responses and state reports.

<<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 getMoreDialledDigitsRes (callSessionID : in TpSessionID, digits : in TpString) : TpResult getMoreDialledDigitsErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult

callEnded (callSessionID : in TpSessionID, report : in TpCallEndedReport) : TpResult

Method

routeRes()

This asynchronous method indicates that the request to route the call to the destination was successful, and indicates the response of the destination party (for example, the call was answered, not answered, refused due to busy, etc.).

If this method is invoked with a monitor mode of P_MONITOR_MODE_INTERRUPTED,

then the APL has control of the call. If the APL does nothing with the call (including its associated legs) within a specified time period (the duration of which forms a part of the service level agreement), then the call in the network shall be released and callEnded() shall be invoked, giving a release cause of 102 (Recovery on timer expiry).

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

eventReport : in TpCallReport

Specifies the result of the request to route the call to the destination party. It also includes the network event, date and time, monitoring mode and event specific information such as release cause.

callLegSessionID : in TpSessionID

Specifies the sessionID of the associated call leg. This corresponds to the sesion ID returned at the routeReq() and can be used to correlate the response with the request.

Raises

TpGCCSException,TpGeneralException

Method

routeErr()

This asynchronous method indicates that the request to route the call to the destination party was unsuccessful - the call could not be routed to the destination party (for example, the network was unable to route the call, the parameters were incorrect, the request was refused, etc.).

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

callLegSessionID : in TpSessionID

Specifies the sessionID of the associated call leg. This corresponds to the sessionID returned at the routeReq() and can be used to correlate the error with the request.

Raises

TpGCCSException,TpGeneralException

Method
getCallInfoRes()

This asynchronous method reports time information of the finished call or call attempt as well as release cause depending on which information has been requested by getCallInfoReq. This information may be used e.g. for charging purposes. The call information will possibly be sent after routeRes in all cases where the call or a leg of the call has been disconnected or a routing failure has been encountered.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callInfoReport : in TpCallInfoReport

Specifies the call information requested.

Raises

TpGCCSException,TpGeneralException

Method
getCallInfoErr()

This asynchronous method reports that the original request was erroneous, or resulted in an error condition.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method
superviseCallRes()

This asynchronous method reports a call supervision event to the application when it has indicated it's interest in these kind of events.

It is also called when the connection is terminated before the supervision event occurs. Furthermore, this method is invoked as a response to the request also when a tariff switch happens in the network during an active call.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call

report : in TpCallSuperviseReport

Specifies the situation which triggered the sending of the call supervision response.

usedTime : in TpDuration

Specifies the used time for the call supervision (in milliseconds).

Raises

TpGCCSException,TpGeneralException

Method

superviseCallErr()

This asynchronous method reports a call supervision error to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Raises

TpGCCSException,TpGeneralException

Method

callFaultDetected()

This method indicates to the application that a fault in the network has been detected. The call may or may not have been terminated.

The system deletes the call object. Therefore, the application has no further control of call processing. No report will be forwarded to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call in which the fault has been detected.

fault : in TpCallFault

Specifies the fault that has been detected.

Raises

`TpGCCSException,TpGeneralException`

Method
`getMoreDialledDigitsRes()`

This asynchronous method returns the collected digits to the application.

Parameters

`callSessionID : in TpSessionID`
Specifies the call session ID of the call.

`digits : in TpString`
Specifies the additional dialled digits if the string length is greater than zero.

Raises

`TpGeneralException,TpGCCSException`

Method
`getMoreDialledDigitsErr()`

This asynchronous method reports an error in collecting digits to the application.

Parameters

`callSessionID : in TpSessionID`
Specifies the call session ID of the call.

`errorIndication : in TpCallError`
Specifies the error which led to the original request failing.

Raises

`TpGeneralException,TpGCCSException`

Method
`callEnded()`

This method indicates to the application that the call has terminated in the network. However, the application may still receive some results (e.g., getCallInfoRes) related to the call. The application is expected to deassign the call object after having received the callEnded.

Note that the event that caused the call to end might also be received separately if the application was monitoring for it.

Parameters

`callSessionID : in TpSessionID`

Specifies the call sessionID.

`report : in TpCallEndedReport`

Specifies the reason the call is terminated.

Raises

`TpGeneralException,TpGCCSException`

6.4 Generic Call Control Service State Transition Diagrams

6.4.1 State Transition Diagrams for IpCallControlManager

The state transition diagram shows the application view on the Call Control Manager object.

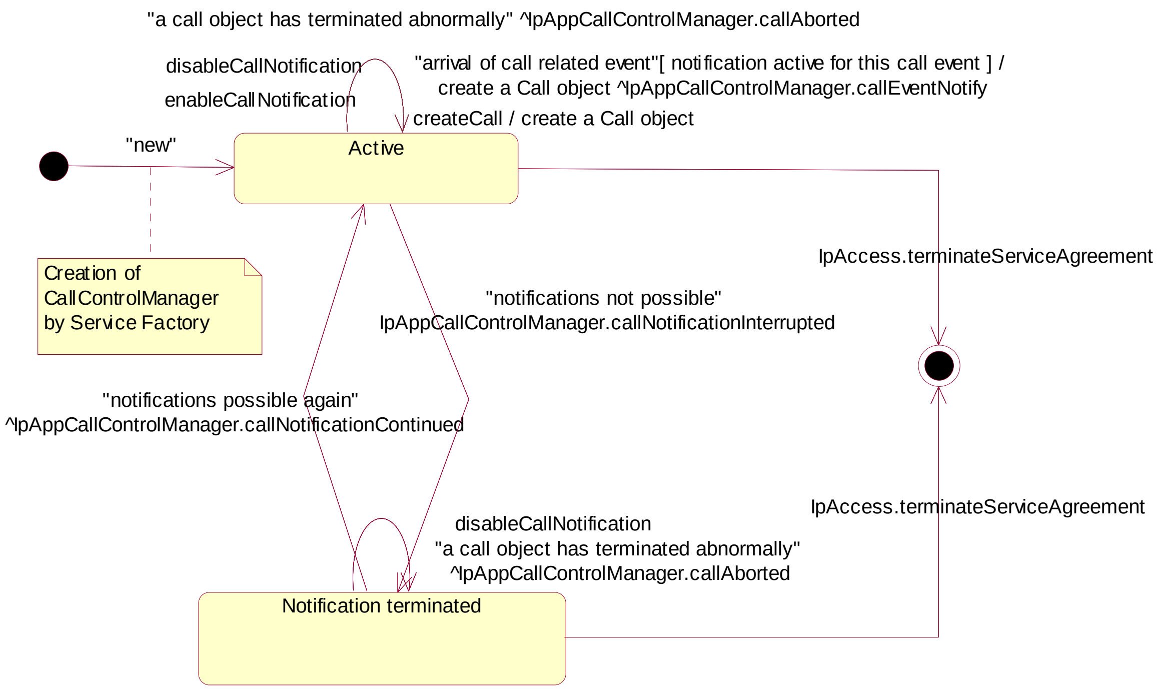

"a call object has terminated abnormally" ^IpAppCallControlManager.callAborted
disableCallNotification
enableCallNotification
"arrival of call related event"[notification active for this call event] / create a Call object ^IpAppCallControlManager.callEventNotify
createCall / create a Call object
"new"
Active
Creation of CallControlManager by Service Factory
IpAccess.terminateServiceAgreement
"notifications not possible" IpAppCallControlManager.callNotificationInterrupted
"notifications possible again" ^IpAppCallControlManager.callNotificationContinued
IpAccess.terminateServiceAgreement
disableCallNotification
"a call object has terminated abnormally" ^IpAppCallControlManager.callAborted
Notification terminated

Figure : Application view on the Call Control Manager

6.4.1.1 Active State

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

6.4.1.2 Notification terminated State

When the Call Control Manager is in the Notification terminated state, events requested with enableCallNotification() will not be forwarded to the application. There can be multiple reasons for this: for instance it might be that the application receives more notifications from the network than defined in the Service Level Agreement. Another example is that the Service has detected it receives no notifications from the network due to e.g. a link failure. In this state no requests for new notifications will be accepted.

6.4.2 State Transition Diagrams for IpCall

The state transition diagram shows the application view on the Call object. This diagram shows only the part of the state transition diagram valid for 3GPP (UMTS) release 99.

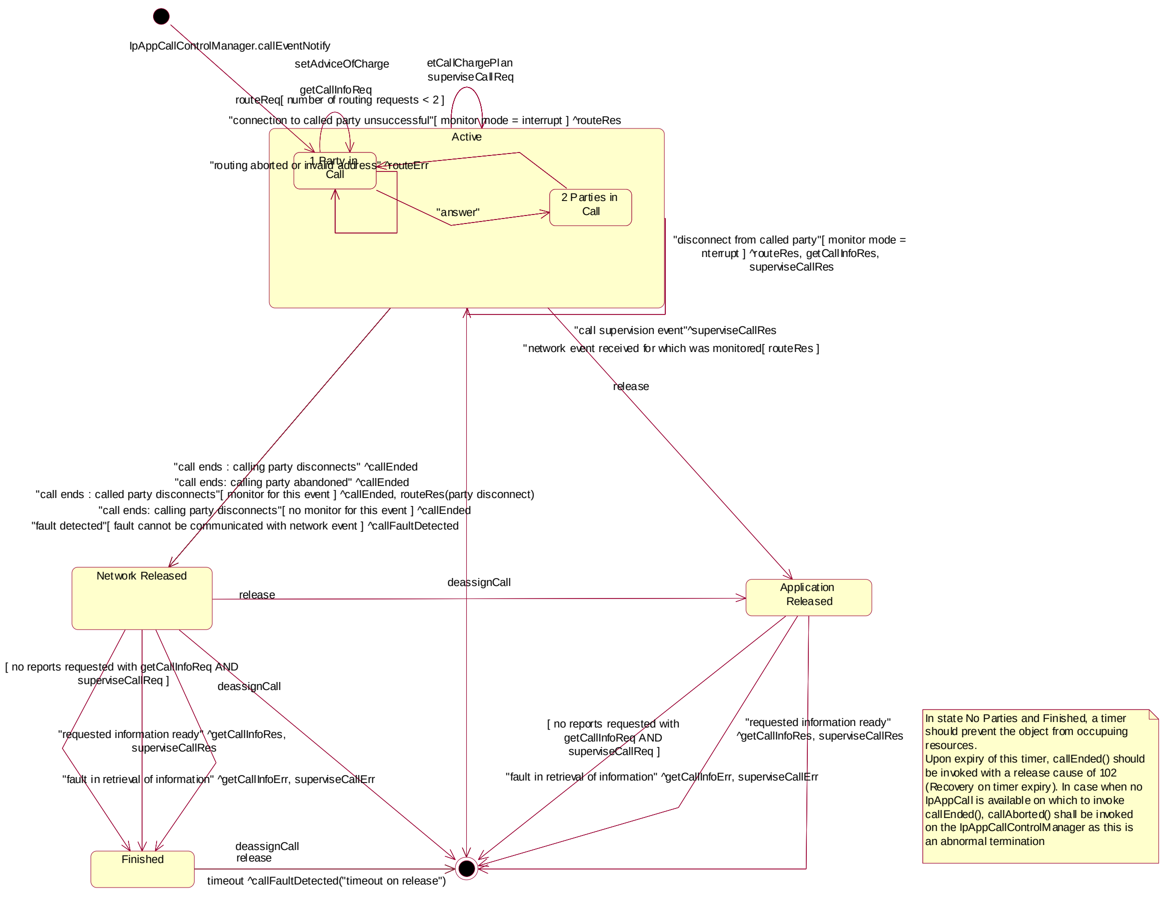

IpAppCallControlManager.callEventNotify
setAdviceOfCharge
etCallChargePlan
superviseCallReq
getCallInfoReq
routeReq[number of routing requests < 2]
"connection to called party unsuccessful"[monitor mode = interrupt] ^routeRes
Active
"routing aborted or invalid address" ^routeErr
1 Party in Call
"answer"
2 Parties in Call
"disconnect from called party"[monitor mode = nterrupt] ^routeRes, getCallInfoRes, superviseCallRes
"call supervision event"^superviseCallRes
"network event received for which was monitored[routeRes]
release
"call ends : calling party disconnects" ^callEnded
"call ends: calling party abandoned" ^callEnded
"call ends : called party disconnects"[monitor for this event] ^callEnded, routeRes(party disconnect)
"call ends: calling party disconnects"[no monitor for this event] ^callEnded
"fault detected"[fault cannot be communicated with network event] ^callFaultDetected
Network Released
release
deassignCall
Application Released
[no reports requested with getCallInfoReq AND superviseCallReq]
deassignCall
"requested information ready" ^getCallInfoRes, superviseCallRes
"fault in retrieval of information" ^getCallInfoErr, superviseCallErr
[no reports requested with getCallInfoReq AND superviseCallReq]
"requested information ready" ^getCallInfoRes, superviseCallRes
"fault in retrieval of information" ^getCallInfoErr, superviseCallErr
Finished
deassignCall
release
timeout ^callFaultDetected("timeout on release")
In state No Parties and Finished, a timer should prevent the object from occuping resources.
Upon expiry of this timer, callEnded() should be invoked with a release cause of 102 (Recovery on timer expiry). In case when no IpAppCall is available on which to invoke callEnded(), callAborted() shall be invoked on the IpAppCallControlManager as this is an abnormal termination

Figure : 3GPP

6.4.2.1 Network Released State

In this state the call has ended and the Gateway collects the possible call information requested with getCallInfoReq() and / or superviseCallReq(). The information will be returned to the application by invoking the methods getCallInfoRes() and / or superviseCallRes() on the application. Also when a call was unsuccessful these methods are used.In case the application has not requested additional call related information immediately a transition is made to state No Parties.

6.4.2.2 Finished State

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

6.4.2.3 Application Released State

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

6.4.2.4 No Parties State

In this state the Call object has been created. The application can request the gateway for a certain type of charging of the call by calling setCallChargePlan(). The application can request for charging related information by calling getCallInfoReq(). Furthermore the application can request supervision of the call by calling superviseCallReq(). It is also allowed to request Advice of Charge information to be sent by calling setAdviceOfCharge().

6.4.2.5 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 supervision of the call by calling superviseCallReq(). It is also allowed to send Advice of Charge information by calling setAdviceOfCharge().

6.4.2.6 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().

In this state the application can also request the gateway for a certain type of charging of the call by calling setCallChargePlan(). The application can also request for charging related information by calling getCallInfoReq(). The setCallChargePlan() and getCallInfoReq() should be issued before requesting a connection to a called party by means of routeReq().

When the calling party abandons the call before the application has invoked the routeReq() operation, the gateway informs the application by invoking callFaultDetected() and also the operation callEnded() will be invoked. When the calling party abandons the call after the application has invoked routeReq() but before the call has actually been established, the gateway informs the application by invoking callEnded().

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

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

6.4.2.7 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.

In this state user interaction is possible, depending on the underlying network.

6.4.2.8 Routing to Destination(s) State

In this state there is at least one outstanding routeReq.

6.5 Generic Call Control Service Properties

6.5.1 List of Service Properties

The following table lists properties relevant for the GCC API.

Property	Type	Description / Interpretation
P_TRIGGERING_EVENT_TYPES	INTEGER_SET	Indicates the static event types supported by the SCS. Static events are the events by which applications are initiated.
P_DYNAMIC_EVENT_TYPES	INTEGER_SET	Indicates the dynamic event types supported by the SCS. Dynamic events are the events the application can request for during the context of a call.
P_ADDRESSPLAN	INTEGER_SET	Indicates the supported address plan (defined in TpAddressPlan.) e.g. {P_ADDRESS_PLAN_E164, P_ADDRESS_PLAN_IP})
P_UI_CALL_BASED	BOOLEAN_SET	Value = TRUE : User interaction can be performed on call level and a reference to a Call object can be used in the IpUIManager.createUICall() operation. Value = FALSE: No User interaction on call level is supported.
P_UI_AT_ALL_STAGES	BOOLEAN_SET	Value = TRUE: User Interaction can be performed at any stage during a call . Value = FALSE: User Interaction can be performed in case there is only one party in the call.
P_MEDIA_TYPE	INTEGER_SET	Specifies the media type used by the Service. Values are defined by data-type TpMediaType : P_AUDIO, P_VIDEO, P_DATA

The previous table lists properties related to capabilities of the SCS itself. The following table lists properties that are used in the context of the Service Level Agreement, e.g. to restrict the access of applications to the capabilities of the SCS.

Property	Type	Description
P_TRIGGERING_ADDRESSES	ADDRESS_RANGE_SET	Indicates for which numbers the notification may be set. For terminating notifications it applies to the terminating number, for originating notifications it applies only to the originating number.
P_NOTIFICATION_TYPES	INTEGER_SET	Indicates whether the application is allowed to set originating and/or terminating triggers in the ECN. Set is: P_ORIGINATING P_TERMINATING
P_MONITOR_MODE	INTEGER_SET	Indicates whether the application is allowed to monitor in interrupt and/or notify mode. Set is: P_INTERRUPT P_NOTIFY
P_NUMBERS_TO_BE_CHANGED	INTEGER_SET	Indicates which numbers the application is allowed to change or fill for legs in an incoming call. Allowed value set: {P_ORIGINAL_CALLED_PARTY_NUMBER, P_REDIRECTING_NUMBER, P_TARGET_NUMBER, P_CALLING_PARTY_NUMBER}.
P_CHARGEPLAN_ALLOWED	INTEGER_SET	Indicates which charging is allowed in the setCallChargePlan indicator. Allowed values: {P_CHARGE_PER_TIME, P_TRANSPARANT_CHARGING, P_CHARGE_PLAN}
P_CHARGEPLAN_MAPPING	INTEGER_INTEGER_MAP	Indicates the mapping of chargeplans (we assume they can be indicated with integers) to a logical network chargeplan indicator. When the chargeplan supports indicates P_CHARGE_PLAN then only chargeplans in this mapping are allowed.

6.5.2 Service Property values for the CAMEL Service Environment.

Implementations of the Generic Call Control API relying on the CSE shall have the Service Properties outlined above set to the indicated values :

```
P_OPERATION_SET = {
“IpCallControlManager.enableCallNotification”,
“IpCallControlManager.disableCallNotification”,
“IpCallControlManager.changeCallNotification”,
“IpCallControlManager.getCriteria”,
“IpCallControlManager.setCallLoadControl”,
“IpCall.routeReq”,
“IpCall.release”,
“IpCall.deassignCall”,
“IpCall.getCallInfoReq”,
“IpCall.setCallChargePlan”,
“IpCall.setAdviceOfCharge”,
“IpCall.superviseCallReq”,
}

P_TRIGGERING_EVENT_TYPES = {
P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT,
P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT,
P_EVENT_GCCS_CALLED_PARTY_BUSY,
P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE,
P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY,
P_EVENT_GCCS_ROUTE_SELECT_FAILURE,
}

P_DYNAMIC_EVENT_TYPES = {
P_CALL_REPORT_ANSWER,
P_CALL_REPORT_BUSY,
P_CALL_REPORT_NO_ANSWER,
P_CALL_REPORT_DISCONNECT,
P_CALL_REPORT_ROUTING_FAILURE
}

P_ADDRESS_PLAN = {
P_ADDRESS_PLAN_E164
}

P_UI_CALL_BASED = {
TRUE
}

P_UI_AT_ALL_STAGES = {
FALSE
}

P_MEDIA_TYPE = {
P_AUDIO
}
```

6.6 Generic Call Control Data Definitions

The present document provides the GCC data definitions necessary to support the API specification.

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

The general format of a Data Definition specification is described below.

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

6.6.1 Generic Call Control Event Notification Data Definitions

TpCallEventName

Defines the names of event being notified. The following events are supported. The values may be combined by a logical 'OR' function when requesting the notifications. Additional events that can be requested / received during the call process are found in the TpCallReportType data-type.

Name	Value	Description
P_EVENT_NAME_UNDEFINED	0	Undefined
P_EVENT_GCCS_OFFHOOK_EVENT	1	GCCS – Offhook event This can be used for hot-line features. In case this event is set in the TpCallEventCriteria, only the originating address(es) may be specified in the criteria.
P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT	2	GCCS – Address information collected The network has collected the information from the A-party, but not yet analysed the information. The number can still be incomplete. Applications might set notifications for this event when part of the number analysis needs to be done in the application (see also the getMoreDialledDigits method on the call class).
P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT	4	GCCS – Address information is analysed The dialled number is a valid and complete number in the network.
P_EVENT_GCCS_CALLED_PARTY_BUSY	8	GCCS – Called party is busy
P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE	16	GCCS – Called party is unreachable (e.g. the called party has a mobile telephone that is currently switched off).
P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY	32	GCCS – No answer from called party
P_EVENT_GCCS_ROUTE_SELECT_FAILURE	64	GCCS – Failure in routing the call
P_EVENT_GCCS_ANSWER_FROM_CALL_PARTY	128	GCCS – Party answered call.

TpCallNotificationType

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

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

P_TERMINATING	2	Indicates that the notification is related to the terminating user in the call.

~~TpCallMonitorMode~~

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

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

TpCallEventCriteria

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

Of the addresses only the Plan and the AddrString are used for the purpose of matching the notifications against the criteria.

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

TpCallEventInfo

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

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

6.6.2 Generic Call Control Data Definitions

IpCall

Defines the address of an IpCall Interface.

IpCallRef

Defines a Reference to type IpCall.

IpAppCall

Defines the address of an IpAppCall Interface.

IpAppCallRef

Defines a Reference to type IpAppCall

IpAppCallRefRef

Defines a Reference to type IpAppCallRef.

TpCallIdentifierRef

Defines a Reference to type TpCallIdentifier.

TpCallIdentifier

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

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

IpAppCallControlManager

Defines the address of an IpAppCallControlManager Interface.

IpAppCallControlManagerRef

Defines a Reference to type IpAppCallControlManager.

IpCallControlManager

Defines the address of an IpCallControlManager Interface.

IpCallControlManagerRef

Defines a Reference to type IpCallControlManager.

~~TpCallAlertingMechanism~~

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

TpCallAppInfo

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

	Tag Element Type	
	TpCallAppInfoType	

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

TpCallAppInfoType

Defines the type of call application-related specific information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)
P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. telephony)
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64kbit/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified service-service information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address

TpCallAppInfoSet

Defines a Numbered Set of Data Elements of TpCallAppInfo.

~~TpCallBearerService~~

~~This data type defines the type of call application-related specific information (Q.931: Information Transfer Capability, and 3GPP TS 22.002 [4]).~~

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

TpCallChargePlan

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

Sequence Element Name	Sequence Element Type	Description
ChargeOrderType	TpCallChargeOrderCategory	Type of charging to be performed: time based charging or transparent charging or pre-defined charge plan.
ChargePerTime	TpChargePerTime	Charge per time. Only applicable when time based charging is selected.
TransparentCharge	TpOctetSet	Operator specific charge plan specification, e.g. charging table name / charging table entry. The associated charge plan data will be send transparently to the charging records. Only applicable when transparent charging is selected.

ChargePlan	TpInt32	Pre-defined charge plan. Example of the charge plan set from which the application can choose could be : (0 = normal user, 1 = silver card user, 2 = gold card user). Only applicable when transparent charging is selected.
Currency	TpString	Currency unit according to ISO-4217:1995
AdditionalInfo	TpOctetSet	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.
PartyToCharge	TpCallPartyToCharge	Party to be charged.
~~Sequence Element Name~~	~~Sequence Element Type~~	~~Description~~
~~ChargeOrderType~~	~~TpCallChargeOrder~~	~~Charge order~~
~~Currency~~	~~TpString~~	~~Currency unit according to ISO-4217:1995 [5]~~
~~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.

TpCallPartyToCharge

Defines the party to be charged

Name	Value	Description
P_CALL_PARTY_ORIGINATING	0	Calling party, i.e. party that initiated the call. For application initiated calls this indicates that the first party requested to be in the call will be charged.
P_CALL_PARTY_DESTINATION	1	Called party, i.e. destination party

~~TpCallChargeOrder~~

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

	~~Tag Element Type~~	
	~~`TpCallChargeOrderCategory`~~	

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

~~TpCallChargeOrderCategory~~

~~Defines the type of charging to be applied~~

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

~~TpCallError~~

~~Defines the Sequence of Data Elements that specify the additional information relating to a call error.~~

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

~~TpCallAdditionalErrorInfo~~

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

	~~Tag Element Type~~	
	~~TpCallErrorType~~	

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

~~TpCallErrorType~~

~~Defines a specific call error.~~

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

~~TpCallFault~~

~~Defines the cause of the call fault detected.~~

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

TpCallEndedReport

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

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

TpCallInfoReport

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

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

A callInfoReport will be generated at the end of user interaction and at the end of the connection with the associated address. This means that either the destination related information is present or the resource related information, but not both.

~~TpCallInfoType~~

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

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

~~TpCallNetworkAccessType~~

~~This data defines the bearer capabilities associated with the call. (3GPP TS 24.002 [6]) This information is network operator specific and may not always be available because there is no standard protocol to retrieve the information.~~

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

~~TpCallPartyCategory~~

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

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

TpCallReleaseCause

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

Sequence Element Name	Sequence Element Type
Value	TpInt32
Location	TpInt32
NOTE: The Value and Location are specified as in ITU-T Recommendation Q.850.	

The following example was taken from Q.850 to aid understanding:

Equivalent Call Report	Cause Value Set by Application	Cause Value from Network
P_CALL_REPORT_BUSY	17	17
P_CALL_REPORT_NO_ANSWER	19	18,19,21

P_CALL_REPORT_DISCONNECT	16	16
P_CALL_REPORT_REDIRECTED	23	23
P_CALL_REPORT_SERVICE_CODE	31	NA
P_CALL_REPORT_ROUTING_FAILURE	3	Any other value

~~TpCallServiceCode~~

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

~~Sequence Element Name~~	~~Sequence Element Type~~
~~CallServiceCodeType~~	~~TpCallServiceCodeType~~
~~ServiceCodeValue~~	~~TpString~~

~~TpCallServiceCodeType~~

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

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

~~TpCallTeleService~~

~~This data type defines the tele-service associated with the call. (Q.763: User Teleservice Information, Q.931: High Layer Compatibility Information, and 3GPP TS 22.003 [7]).~~

~~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~~

~~TpCallSuperviseReport~~

~~Defines the responses from the CC service for calls that are supervised. The values may be combined by a logical 'OR' function.~~

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

~~TpCallSuperviseTreatment~~

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

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

TpCallReport

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

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

TpCallAdditionalReportInfo

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

	Tag Element Type	
	TpCallReportType	

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

TpCallReportRequest

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

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

TpCallAdditionalReportCriteria

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

	Tag Element Type	
	TpCallReportType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_REPORT_UNDEFINED	NULL	Undefined
P_CALL_REPORT_PROGRESS	NULL	Undefined
P_CALL_REPORT_ALERTING	NULL	Undefined
P_CALL_REPORT_ANSWER	NULL	Undefined
P_CALL_REPORT_BUSY	NULL	Undefined
P_CALL_REPORT_NO_ANSWER	TpDuration	NoAnswerDuration
P_CALL_REPORT_DISCONNECT	NULL	Undefined
P_CALL_REPORT_REDIRECTED	NULL	Undefined
P_CALL_REPORT_SERVICE_CODE	TpCallServiceCode	ServiceCode
P_CALL_REPORT_ROUTING_FAILURE	NULL	Undefined

TpCallReportRequestSet

Defines a `Numbered Set of Data Elements` of TpCallReportRequest.

TpCallReportType

Defines a specific call event report type.

Name	Value	Description
P_CALL_REPORT_UNDEFINED	0	Undefined.
P_CALL_REPORT_PROGRESS	1	Call routing progress event:an indication from the network that progress has been made in routing the call to the requested call party. This message may be sent more than once, or may not be sent at all by the gateway with respect to routing a given call leg to a given address.
P_CALL_REPORT_ALERTING	2	Call is alerting at the call party.
P_CALL_REPORT_ANSWER	3	Call answered at address.
P_CALL_REPORT_BUSY	4	Called address refused call due to busy.
P_CALL_REPORT_NO_ANSWER	5	No answer at called address.
P_CALL_REPORT_DISCONNECT	6	The media stream of the called party has disconnected. This does not imply that the call has ended. When the call is ended, the callEnded method is called. This event can occur both when the called party hangs up, or when the application explicitly releases the leg using IpCallLeg::release() This cannot occur when the app explicitly releases the call leg and the call.~~The call party has disconnected.~~
P_CALL_REPORT_REDIRECTED	7	Call redirected to new address: an indication from the network that the call has been redirected to a new address.
P_CALL_REPORT_SERVICE_CODE	8	Mid-call service code received.
P_CALL_REPORT_ROUTING_FAILURE	9	Call routing failed - re-routing is possible.
P_CALL_REPORT_QUEUED	10	The call is being held in a queue. This event may be sent more than once during the routing of a call.

~~TpCallLoadControlMechanism~~

~~Defines the Tagged Choice of Data Elements that specify the applied mechanism and associated parameters.~~

	~~Tag Element Type~~	
	~~TpCallLoadControlMechanismType~~	

~~Tag Element Value~~	~~Choice Element Type~~	~~Choice Element Name~~
~~P_CALL_LOAD_CONTROL_PER_INTERVAL~~	~~TpCallLoadControlIntervalRate~~	~~CallLoadControlPerInterval~~

~~TpCallLoadControlIntervalRate~~

~~Defines the call admission rate of the call load control mechanism used. This data type indicates the interval (in milliseconds) between calls that are admitted.~~

~~Name~~	~~Value~~	~~Description~~
~~P_CALL_LOAD_CONTROL_ADMIT_NO_CALLS~~	~~0~~	~~Infinite interval (do not admit any calls)~~
	~~1 - 60000~~	~~Duration in milliseconds~~

~~TpCallLoadControlMechanismType~~

~~Defines the type of call load control mechanism to use.~~

~~Name~~	~~Value~~	~~Description~~
~~P_CALL_LOAD_CONTROL_PER_INTERVAL~~	~~1~~	~~admit one call per interval~~

TpCallTreatment

Defines the Sequence of Data Elements that specify the the treatment for calls that will be handled only by the network (for example, call which are not admitted by the call load control mechanism).

Sequence Element Name	Sequence Element Type
ReleaseCause	TpCallReleaseCause
AdditionalTreatmentInfo	TpCallAdditionalTreatmentInfo

~~TpCallAdditionalTreatmentInfo~~

~~Defines the Tagged Choice of Data Elements that specify the information to be sent to a call party.~~

	~~Tag Element Type~~	
	~~TpCallTreatmentType~~	

~~Tag Element Value~~	~~Choice Element Type~~	~~Choice Element Name~~
~~P_CALL_TREATMENT_DEFAULT~~	~~NULL~~	~~Undefined~~
~~P_CALL_TREATMENT_RELEASE~~	~~NULL~~	~~Undefined~~
~~P_CALL_TREATMENT_SIAR~~	~~TpUICallInfoID~~	~~InformationToSend~~

~~TpCallTreatmentType~~

~~Defines the treatment for calls that will be handled only by the network.~~

~~Name~~	~~Value~~	~~Description~~
~~P_CALL_TREATMENT_DEFAULT~~	~~0~~	~~Default treatment~~
~~P_CALL_TREATMENT_RELEASE~~	~~1~~	~~Release the call~~
~~P_CALL_TREATMENT_SIAR~~	~~2~~	~~Send information to the user, and release the call (Send Info & Release)~~

TpCallEventCriteriaResultSetRef

Defines a refernce to TpCallEventCriteriaResultSet.

TpCallEventCriteriaResultSet

Defines a set of TpCallEventCriteriaResult.

TpCallEventCriteriaResult

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

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

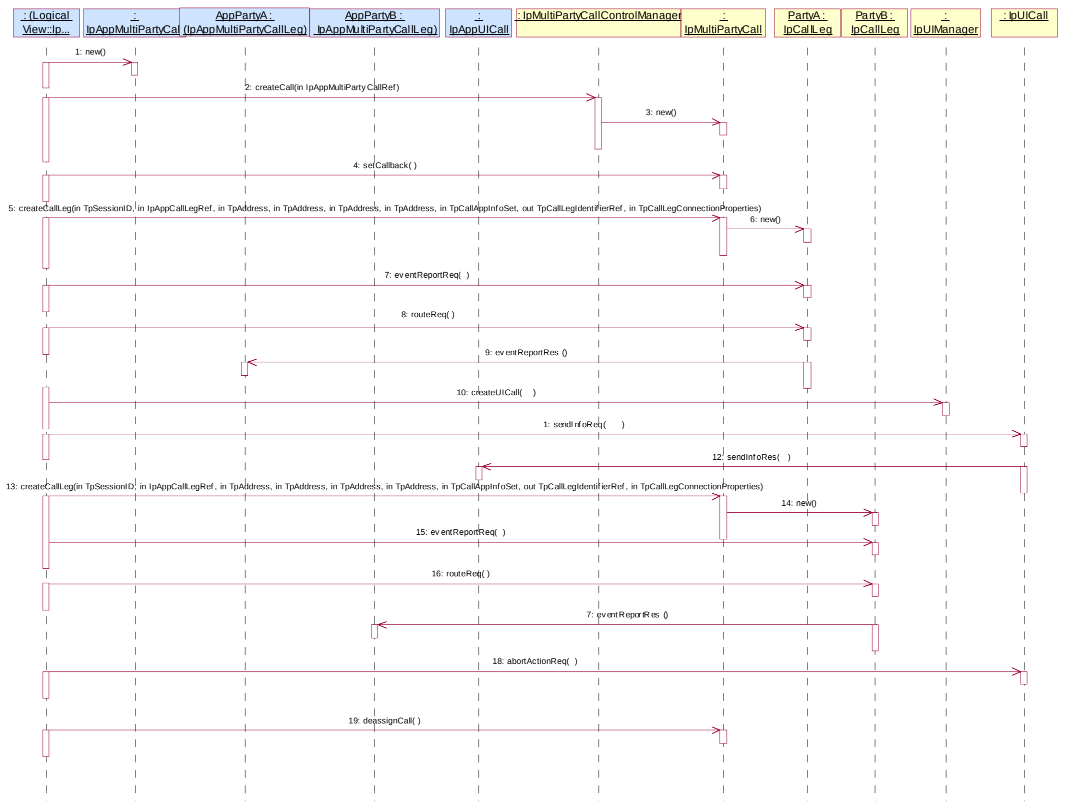
7 MultiParty Call Control Service

The Multi-Party Call Control API of 3GPP Rel4 relies on the CAMEL Service Environment (CSE). It should be noted that a number of restrictions exist because CAMEL phase 3 supports only two-party calls and no leg based operations. Furthermore application initiated calls are not supported in CAMEL phase 3. The detailed description of the supported methods is given in the chapter 7.5.

7.1 Sequence Diagrams

7.1.1 Application initiated call setup

The following sequence diagram shows an application creating a call between party A and party B. Here, a call is created first. Then party A's call leg is created before triggers are set on it for answer and then routed to the call. On answer, an announcement is played indicating that the call is being set up to party B. While the announcement is being played, party B's call leg is created and then triggers are set on it for answer. On answer the announcement is cancelled and party B is routed to the call.


: (Logical View::Ip...
: IpAppMultiPartyCal
AppPartyA : (IpAppMultiPartyCallLeg)
AppPartyB : IpAppMultiPartyCallLeg)
: IpAppUICall
: IpMultiPartyCallControlManager
: IpMultiPartyCall
PartyA : IpCallLeg
PartyB : IpCallLeg
: IpUIManager
: IpUICall
1: new()
2: createCall(in IpAppMultiPartyCallRef)
3: new()
4: setCallback( )
5: createCallLeg(in TpSessionID, in IpAppCallLegRef, in TpAddress, in TpAddress, in TpAddress, in TpAddress, in TpCallAppInfoSet, out TpCallLegIdentifierRef, in TpCallLegConnectionProperties)
6: new()
7: eventReportReq( )
8: routeReq( )
9: eventReportRes ()
10: createUICall( )
1: sendInfoReq( )
12: sendInfoRes( )
13: createCallLeg(in TpSessionID, in IpAppCallLegRef, in TpAddress, in TpAddress, in TpAddress, in TpAddress, in TpCallAppInfoSet, out TpCallLegIdentifierRef, in TpCallLegConnectionProperties)
14: new()
15: eventReportReq( )
16: routeReq( )
7: eventReportRes ()
18: abortActionReq( )
19: deassignCall( )

1: This message is used to create an object implementing the IpAppMultiPartyCall interface.

2: This message requests the object implementing the IpMultiPartyCallControlManager interface to create an object implementing the IpMultiPartyCall interface.

3: Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met it is created.

4: Once the object implementing the IpMultiPartyCall interface is created it is used to pass the reference of the object implementing the IpAppMultiPartyCall interface as the callback reference to the object implementing the

IpMultiPartyCall interface. Note that the reference to the callback interface could already have been passed in the createCall.

5: This message instructs the object implementing the IpMultiPartyCall interface to create a call leg for customer A.

6: Assuming that the criteria for creating an object implementing the IpCallLeg interface is met, message 6 is used to create it.

7: This message requests the call leg for customer A to inform the application when the call leg answers the call.

8: The call is then routed to the originating call leg.

9: Assuming the call is answered, the object implementing party A's IpCallLeg interface passes the result of the call being answered back to its callback object. This message is then forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

10: A UICall object is created and associated with the just created call leg.

11: This message is used to inform party A that the call is being routed to party B.

12: An indication that the dialogue with party A has commenced is returned via message 13 and eventually forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

13: This message instructs the object implementing the IpMultiPartyCall interface to create a call leg for customer B.

14: Assuming that the criteria for creating a second object implementing the IpCallLeg interface is met, it is created.

15: This message requests the call leg for customer B to inform the application when the call leg answers the call.

16: The call is then routed to the call leg.

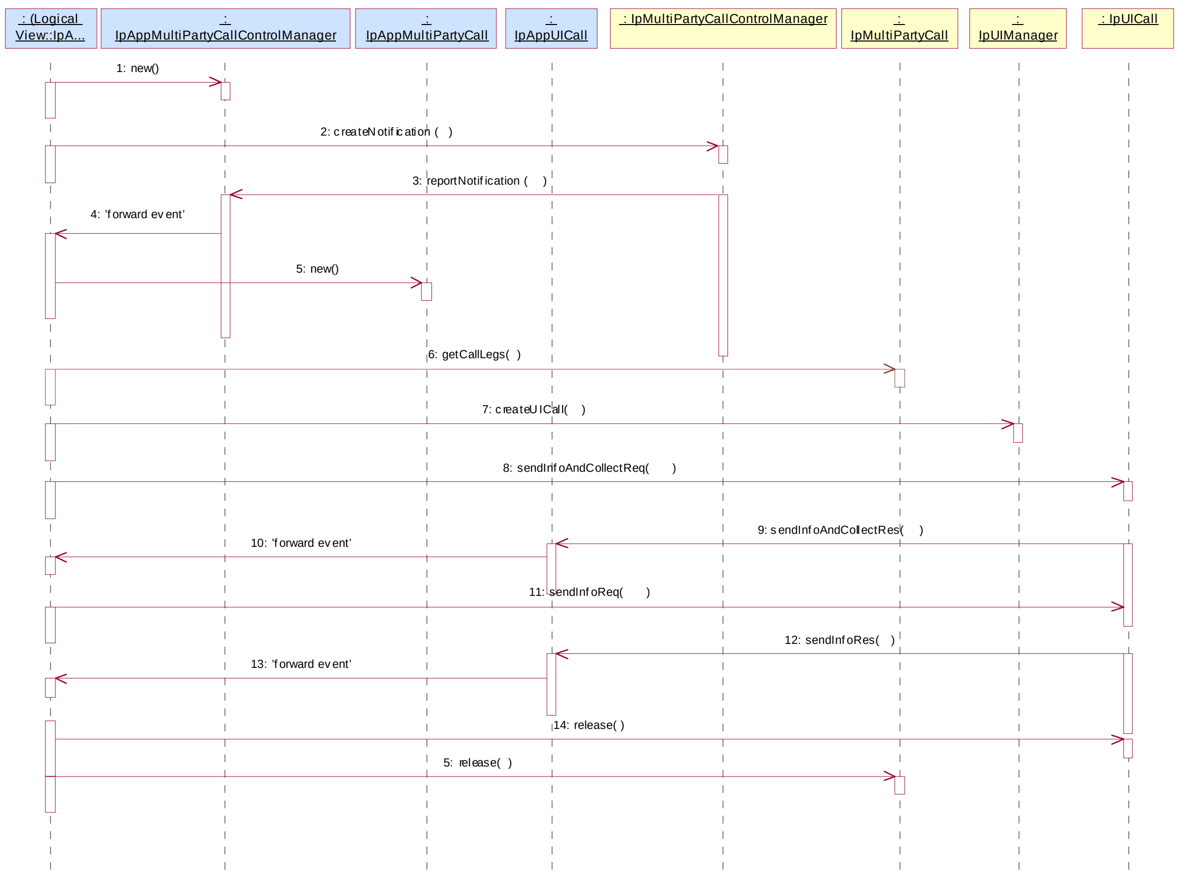
17: Assuming the call is answered, the object implementing party B's IpCallLeg interface passes the result of the call being answered back to its callback object. This message is then forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

18: This message then instructs the object implementing the IpUICall interface to stop sending announcements to party A.

19: The application deassigns the call. This will also deassign the associated user interaction.

7.1.2 Call Barring 2

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the framework. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is rejected and the call is cleared.



1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface.

4: This message is used to forward message 3 to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of the callEventNotify.

6: The application requests an list of all the legs currently in the call.

7: This message is used to create a UICall object that is associated with the incoming leg of the call.

8: The call barring service dialogue is invoked.

9: The result of the dialogue, which in this case is the PIN code, is returned to its callback object.

10: This message is used to forward the previous message to the IpAppLogic

11: Assuming an incorrect PIN is entered, the calling party is informed using additional dialogue of the reason why the call cannot be completed.

12: This message passes the indication that the additional dialogue has been sent.

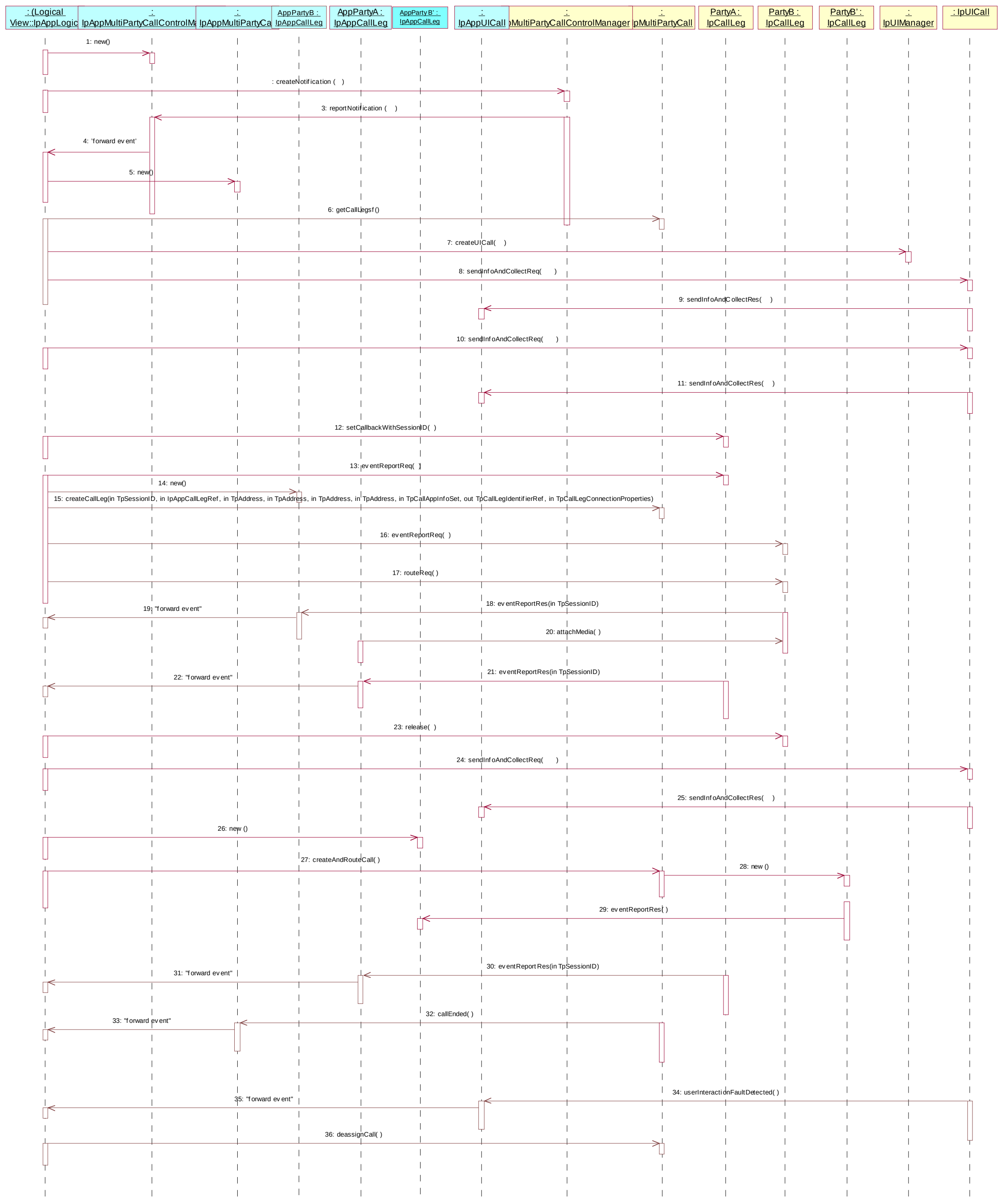
13: This message is used to forward the previous message to the IpAppLogic.

14: No more UI is required, so the UICall object is released.

15: This message is used by the application to clear the call.

7.1.3 Complex Card Service

The following sequence diagram shows an advanced card service, initiated as a result of a prearranged event being received by the framework. Before the call is made, the calling party is asked for an ID and PIN code. If the ID and PIN code are accepted, the calling party is prompted to enter the address of the destination party. A trigger of '#5' is then set on the controlling leg (the calling party's leg) such that if the calling party enters a '#5' an event will be sent to the application. The call is then routed to the destination party. Sometime during the call the calling party enters '#5' which causes the called leg to be released. The calling party is now prompted to enter the address of a new destination party, to which it is then routed.


: (Logical View::IpAppLogic
: IpAppMultiPartyCallControlMa
: IpAppMultiPartyCa
AppPartyB : IpAppCallLeg
AppPartyA : IpAppCallLeg
AppPartyB' : IpAppCallLeg
: IpAppUICall
: pMultiPartyCallControlManager
: pMultiPartyCall
PartyA : IpCallLeg
PartyB : IpCallLeg
PartyB' : IpCallLeg
: IpUIManager
: IpUICall
1: new()
: createNotification ( )
3: reportNotification ( )
4: 'forward event'
5: new()
6: getCallLegs()
7: createUICall( )
8: sendInfoAndCollectReq( )
9: sendInfoAndCollectRes( )
10: sendInfoAndCollectReq( )
11: sendInfoAndCollectRes( )
12: setCallbackWithSessionID( )
13: eventReportReq( )
14: new()
15: createCallLeg(in TpSessionID, in IpAppCallLegRef, in TpAddress, in TpAddress, in TpAddress, in TpAddress, in TpCallAppInfoSet, out TpCallLegIdentifierRef, in TpCallLegConnectionProperties)
16: eventReportReq( )
17: routeReq( )
18: eventReportRes(in TpSessionID)
19: "forward event"
20: attachMedia( )
21: eventReportRes(in TpSessionID)
22: "forward event"
23: release( )
24: sendInfoAndCollectReq( )
25: sendInfoAndCollectRes( )
26: new ()
27: createAndRouteCall( )
28: new ()
29: eventReportRes( )
30: eventReportRes(in TpSessionID)
31: "forward event"
32: callEnded( )
33: "forward event"
34: userInteractionFaultDetected( )
35: "forward event"
36: deassignCall( )

1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range result in the caller being prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

3: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface.

4: This message is used to forward message 3 to the IpAppLogic.

5: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of message 3.

6: This message retuns the call legs currently in the call. In principle a reference to the call leg of the calling party is already obtained by the application when it was notified of the new call event.

7: This message is used to associate a user interaction object with the calling party.

8: The initial card service dialogue is invoked using this message.

9: The result of the dialogue, which in this case is the ID and PIN code, is returned to its callback object using this message and eventually forwarded via another message (not shown) to the IpAppLogic.

10: Assuming the correct ID and PIN are entered, the final dialogue is invoked.

11: The result of the dialogue, which in this case is the destination address, is returned and eventually forwarded via another message (not shown) to the IpAppLogic.

12: This message is used to forward the address of the callback object.

13: The trigger for follow-on calls is set (on service code).

14: A new AppCallLeg is created to receive callbacks for another leg. Alternatively, the already existing AppCallLeg object could be passed in the subsequent createCallLeg(). In that case the application has to use the sessionIDs of the legs to distinguish between callbacks destined for the A-leg and callbacks destined for the B-leg.

15: This message is used to create a new call leg object. The object is created in the idle state and not yet routed in the network.

16: The application requests to be notified when the leg is answered.

17: The application routes the leg. As a result the network will try to reach the associated party.

18: When the B-party answers the call, the application is notified.

19: The event is forwarded to the application logic.

20: Legs that are created and routed explicitly are by default in state detached. This means that the media is not connected to the other parties in the call. In order to allow inband communication between the new party and the other parties in the call the media have to be explicitly attached.

21: At some time during the call the calling party enters '#5'. This causes this message to be sent to the object implementing the IpAppCallLeg interface, which forwards this event as a message (not shown) to the IpAppLogic.

22: The event is forwarded to the application.

23: This message releases the called party.

24: Another user interaction dialogue is invoked.

25: The result of the dialogue, which in this case is the new destination address is returned and eventually forwarded via another message (not shown) to the IpAppLogic.

26: A new AppCallLeg is created to receive callbacks for another leg.

27: The call is then forward routed to the new destination party.

28: As a result a new Callleg object is created.

29: This message passes the result of the call being answered to its callback object and is eventually forwarded via another message (not shown) to the IpAppLogic.

30: When the A-party terminates the application is informed.

31: The event is forwarded to the application logic.

32: Since the release of the A-party will in this case terminate the entire call, the application is also notified with this message.

33: The event is forwarded to the application logic.

34: Since the user interaction object were not released at the moment that the call terminated, the application receives this message to indicate that the UI resources are released in the gateway and no further communication is possible.

35: The event is forwarded to the application logic.

36: The application deassigns the call object.

7.2 Class Diagrams

The multiparty call control service consists of two packages, one for the interfaces on the application side and one for interfaces on the service side.

The class diagrams in the following figures show the interfaces that make up the multi party call control application package and the multi party call control service package. This class diagram shows the interfaces of the multi-party call control application package and their relations to the interfaces of the multi-party call control service package.

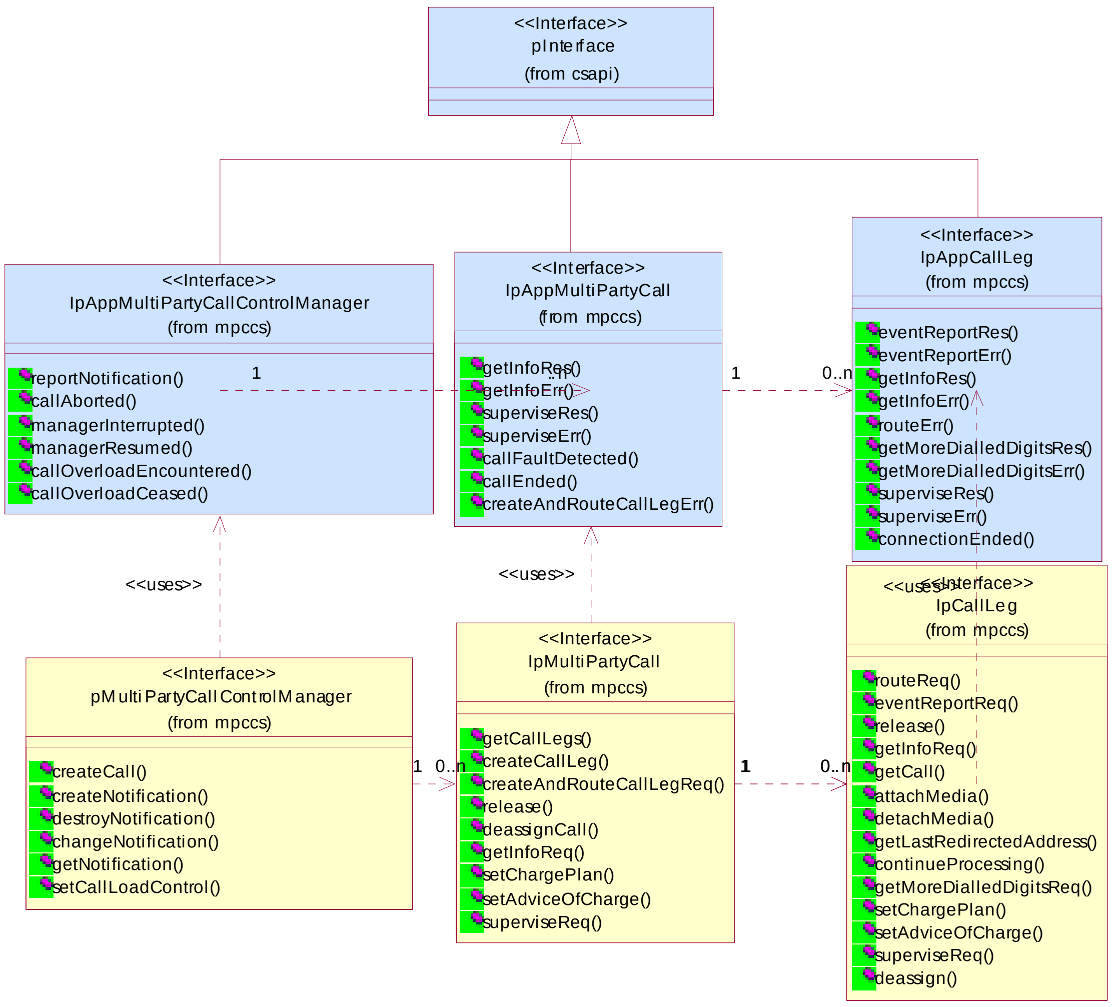


Figure: Application Interfaces

This class diagram shows the interfaces of the multi-party call control service package.

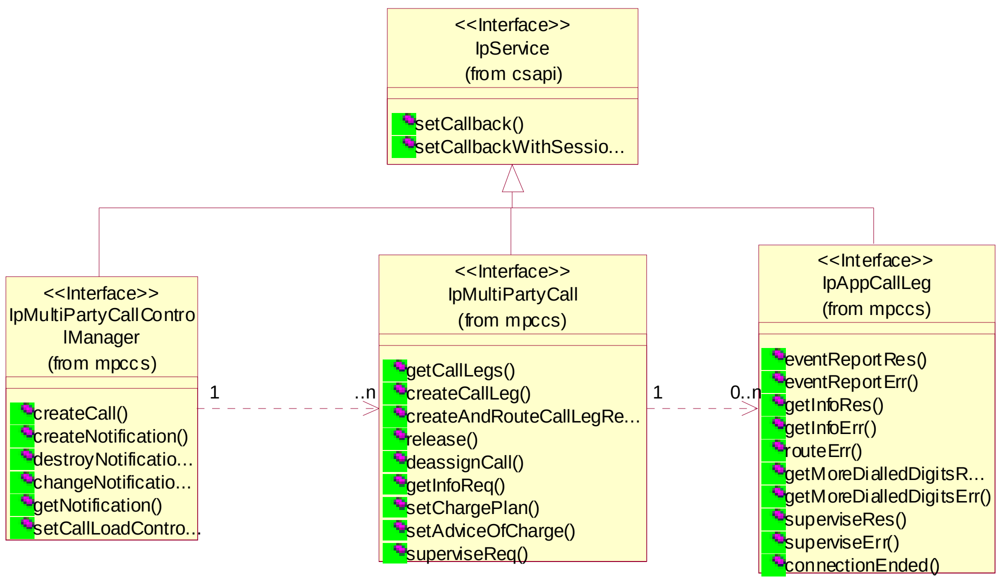


Figure: Service Interfaces

7.3 MultiParty Call Control Service Interface Classes

The Multi-party Call Control service enhances the functionality of the Generic Call Control Service with leg management. It also allows for multi-party calls to be established, i.e., up to a service specific number of legs can be connected simultaneously to the same call.

The Multi-party Call Control Service is represented by the IpMultiPartyCallControlManager, IpMultiPartyCall, IpCallLeg interfaces that interface to services provided by the network. Some methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls. To handle responses and reports, the developer must implement IpAppMultiPartyCallControlManager, IpAppMultiPartyCall and IpAppCallLeg to provide the callback mechanism.

7.3.1 Interface Class IpMultiPartyCallControlManager

Inherits from: IpService

This interface is the 'service manager' interface for the Multi-party Call Control Service. The multi-party call control manager interface provides the management functions to the multi-party call control service. The application programmer can use this interface to provide overload control functionality, create call objects and to enable or disable call-related event notifications. The action table associated with the STD shows in what state the IpMultiPartyCallControlManager must be if a method can successfully complete. In other words, if the IpMultiPartyCallControlManager is in another state the method will throw an exception immediately.

<<Interface>> IpMultiPartyCallControlManager
createCall (appCall : in IpAppMultiPartyCallRef, callReference : out TpMultiPartyCallIdentifierRef) : TpResult createNotification (appCallControlManager : in IpAppMultiPartyCallControlManagerRef, notificationRequest : in TpCallNotificationRequest, assignmentID : out TpAssignmentIDRef) : TpResult destroyNotification (assignmentID : in TpAssignmentID) : TpResult changeNotification (assignmentID : in TpAssignmentID, notificationRequest : in TpCallNotificationRequest) : TpResult getNotification (notificationsRequested : out TpNotificationRequestedSetRef) : TpResult setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange, assignmentID : out TpAssignmentIDRef) : TpResult

Method
createCall()

This method is used to create a new call object. An IpAppMultiPartyCallControlManager should already have been passed to the IpMultiPartyCallControlManager,

otherwise the call control will not be able to report a callAborted() to the application (the application should invoke setCallback() if it wishes to ensure this).

Parameters

appCall : in IpAppMultiPartyCallRef

Specifies the application interface for callbacks from the call created.

callReference : out TpMultiPartyCallIdentifierRef

Specifies the interface reference and sessionID of the call created.

Raises

TpCommonExceptions

Method
createNotification()

This method is used to enable call notifications so that events can be sent to the application. This is the first step an application has to do to get initial notifications of calls happening in the network. When such an event happens, the application will be informed by reportNotification(). In case the application is interested in other events during the context of a particular call session it has to use the createAndRouteCallLegReq() method on the call object or the eventReportReq() method on the call leg object. The application will get access to the call object when it receives thye reportNotification(). (Note that createNotification() is not applicable if the call is setup by the application).

The createNotification method is purely intended for applications to indicate their interest to be notified when certain call events take place. It is possible to subscribe to a certain event for a whole range of addresses, e.g. the application can indicate it wishes to be informed when a call is made to any number starting with 800.

If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P_GCCS_INVALID_CRITERIA. The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used and the same NotificationCallType is used.

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. Both notifications will share the same assignmentID. The gateway will always use the most recent callback. In case this most recent callback fails the second most recent is used. In case the enableCallNotification contains no callback, at the moment the application needs to be informed the gateway will use as

callback the callback that has been registered by setCallback().

Parameters

appCallControlManager : in IpAppMultiPartyCallControlManagerRef

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the setCallback() method.

notificationRequest : in TpCallNotificationRequest

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported. Examples of events are "incoming call attempt reported by network", "answer", "no answer", "busy". Individual addresses or address ranges may be specified for destination and/or origination.

assignmentID : out TpAssignmentIDRef

Specifies the ID assigned by the generic call control manager interface for this newly-enabled event notification.

Raises

TpCommonExceptions, P_INVALID_CRITERIA, P_INVALID_INTERFACE_TYPE, P_INVALID_EVENT_TYPE

Method

destroyNotification()

This method is used by the application to disable call notifications.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID given by the generic call control manager interface when the previous enableNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID. If two callbacks have been registered under this assignment ID both of them will be disabled.

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

Method
changeNotification()

This method is used by the application to change the event criteria introduced with createNotification. Any stored criteria associated with the specified assignementID will be replaced with the specified criteria.

Parameters

assignmentID : in TpAssignmentID

Specifies the ID assigned by the generic call control manager interface for the event notification. If two callbacks have been registered under this assigment ID both of them will be disabled.

notificationRequest : in TpCallNotificationRequest

Specifies the new set of event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

Method
getNotification()

This method is used by the application to query the event criteria set with createNotification or changeNotification.

Parameters

notificationsRequested : out TpNotificationRequestedSetRef

Specifies the nofications that have been requested by the application.

Raises

TpCommonExceptions

Method
setCallLoadControl()

This method imposes or removes load control on calls made to a particular address range within the call control service. The address matching mechanism is similar as defined for TpCallEventCriteria.

Parameters

duration : in TpDuration

Specifies the duration for which the load control should be set.

A duration of 0 indicates that the load control should be removed.

A duration of -1 indicates an infinite duration (i.e., until disabled by the application)

A duration of -2 indicates the network default duration.

mechanism : in TpCallLoadControlMechanism

Specifies the load control mechanism to use (for example, admit one call per interval), and any necessary parameters, such as the call admission rate. The contents of this parameter are ignored if the load control duration is set to zero.

treatment : in TpCallTreatment

Specifies the treatment of calls that are not admitted. The contents of this parameter are ignored if the load control duration is set to zero.

addressRange : in TpAddressRange

Specifies the address or address range to which the overload control should be applied or removed.

assignmentID : out TpAssignmentIDRef

Specifies the assignmentID assigned by the gateway to this request. This assignementID can be used to correlate the callOverlloadEncountered and callOverloadCeased methods with the request.

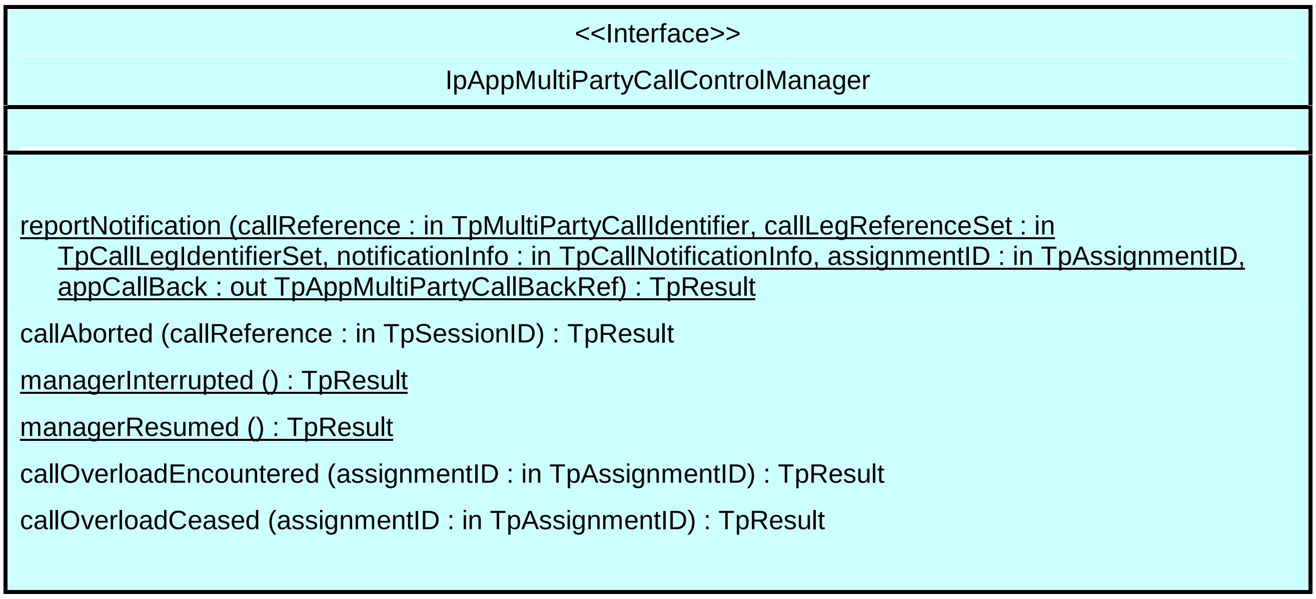
Raises

TpCommonExceptions, P_INVALID_ADDRESS, P_UNSUPPORTED_ADDRESS_PLAN

7.3.2 Interface Class IpAppMultiPartyCallControlManager

Inherits from: IpInterface

The Multi-Party call control manager application interface provides the application call control management functions to the Multi-Party call control service.



<<Interface>>

IpAppMultiPartyCallControlManager

reportNotification (callReference : in TpMultiPartyCallIdentifier, callLegReferenceSet : in TpCallLegIdentifierSet, notificationInfo : in TpCallNotificationInfo, assignmentID : in TpAssignmentID, appCallBack : out TpAppMultiPartyCallBackRef) : TpResult

callAborted (callReference : in TpSessionID) : TpResult

managerInterrupted () : TpResult

managerResumed () : TpResult

callOverloadEncountered (assignmentID : in TpAssignmentID) : TpResult

callOverloadCeased (assignmentID : in TpAssignmentID) : TpResult

Method

reportNotification()

This method notifies the application of the arrival of a call-related event.

If this method is invoked with a monitor mode of P_MONITOR_MODE_INTERRUPTED, then the APL has control of the call. If the APL does nothing with the call (including its associated legs) within a specified time period (the duration of which forms a part of the service level agreement), then the call in the network shall be released and callEnded() shall be invoked, giving a release cause of P_TIMER_EXPIRY.

Parameters

callReference : in TpMultiPartyCallIdentifier

Specifies the reference to the call interface to which the notification relates. This parameter will be null if the notification is being given in NOTIFY mode.

callLegReferenceSet : in TpCallLegIdentifierSet

Specifies the set of all call leg references. First in the set is the reference to the originating callLeg. It indicates the call leg related to the originating party. In case there is a destination call leg this will be the second leg in the set. from the notificationInfo can be found on who's behalf the notification was sent.

However, this parameter will be null if the notification is being given in NOTIFY mode.

notificationInfo : in TpCallNotificationInfo

Specifies data associated with this event (e.g. the originating or terminating leg which reports the notification).

assignmentID : in TpAssignmentID

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

appCallBack : out TpAppMultiPartyCallBackRef

Specifies references to the application interface which implements the callback interface for the new call and/or new call leg. This parameter may be null if the notification is being given in NOTIFY mode.

Method

callAborted()

This method indicates to the application that the call object has aborted or terminated abnormally. No further communication will be possible between the call and application.

Parameters

callReference : in TpSessionID

Specifies the sessionID of call that has aborted or terminated abnormally.

Method

managerInterrupted()

This method indicates to the application that event notifications and method invocations have been temporary interrupted (for example, due to network resources unavailable).

Note that more permanent failures are reported via the Framework (integrity management).

Parameters

No Parameters were identified for this method

Method

managerResumed()

This method indicates to the application that event notifications possibleand method invocations are enabled.

Parameters

No Parameters were identified for this method

Method

callOverloadEncountered()

This method indicates that the network has detected overload and may have automatically imposed load control on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been encountered.

Method

callOverloadCeased()

This method indicates that the network has detected that the overload has ceased and has automatically removed any load controls on calls requested to a particular address range or calls made to a particular destination within the call control service.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been ceased

7.3.3 Interface Class IpMultiPartyCall

Inherits from: IpService

The Multi-Party Call provides the possibility to control the call routing, to request information from the call, control the charging of the call, to release the call and to supervise the call. It also gives the possibility to manage call legs explicitly. An application may create more then one call leg.

<<Interface>>

IpMultiPartyCall

getCallLegs (callSessionID : in TpSessionID, callLegList : out TpCallLegIdentifierSetRef) : TpResult

createCallLeg (callSessionID : in TpSessionID, appCallLeg : in IpAppCallLegRef, callLeg : out TpCallLegIdentifierRef) : TpResult

createAndRouteCallLegReq (callSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet, targetAddress : in TpAddress, originatingAddress : in TpAddress, appInfo : in TpCallAppInfoSet, appLegInterface : in IpAppCallLegRef, callLegReference : out TpCallLegIdentifierRef) : TpResult

release (callSessionID : in TpSessionID, cause : in TpCallReleaseCause) : TpResult

deassignCall (callSessionID : in TpSessionID) : TpResult

getInfoReq (callSessionID : in TpSessionID, callInfoRequested : in TpCallInfoType) : TpResult

setChargePlan (callSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : TpResult

setAdviceOfCharge (callSessionID : in TpSessionID, aOCInfo : in TpAocInfo, tariffSwitch : in TpDuration) : TpResult

superviseReq (callSessionID : in TpSessionID, time : in TpDuration, treatment : in TpCallSuperviseTreatment) : TpResult

Method
getCallLegs()

This method requests the identification of the call leg objects associated with the call object. Returns the legs in the order of creation.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callLegList : out TpCallLegIdentifierSetRef

Specifies the call legs associated with the call. The set contains both the sessionIDs and the interface references.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
createCallLeg()

This method requests the creation of a new call leg object.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

appCallLeg : in IpAppCallLegRef

Specifies the application interface for callbacks from the call leg created.

callLeg : out TpCallLegIdentifierRef

Specifies the interface and sessionID of the call leg created.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_ADDRESS, P_UNSUPPORTED_ADDRESS_PLAN

Method

createAndRouteCallLegReq()

This asynchronous operation requests creation and routing of a new callLeg. In case the connection to the destination party is established successfully the CallLeg is attached to the call, i.e. no explicit setMedia() operation is needed. Requested events will be reported on the IpAppCallLeg interface. This interface the application must provide through the appLegInterface parameter.

The extra address information such as originatingAddress is optional. If not present (i.e., the plan is set to P_ADDRESS_PLAN_NOT_PRESENT), the information provided in corresponding addresses from the route is used, otherwise the network or gateway provided numbers will be used.

If this method in invoked, and call reports have been requested, yet the IpAppCallLeg interface parameter is NULL, this method shall throw the P_NO_CALLBACK_ADDRESS_SET exception.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

eventsRequested : in TpCallEventRequestSet

Specifies the event specific criteria used by the application to define the events required. Only events that meet these criteria are reported. Examples of events are "adress analysed", "answer", "release".

targetAddress : in TpAddress

Specifies the destination party to which the call should be routed.

originatingAddress : in TpAddress

Specifies the address of the originating (calling) party.

appInfo : in TpCallAppInfoSet

Specifies application-related information pertinent to the call (such as alerting method, tele-service type, service identities and interaction indicators).

appLegInterface : in IpAppCallLegRef

Specifies a reference to the application interface that implements the callback interface for the new call leg. Requested events will be reported by the eventReportRes() operation on this interface.

callLegReference : out TpCallLegIdentifierRef

Specifies the reference to the CallLeg interface that was created.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_ADDRESS , P_UNSUPPORTED_ADDRESS_PLAN, P_INVALID_NETWORK_STATE, P_INVALID_CRITERIA

Method
release()

This method requests the release of the call object and associated objects. The call will also be terminated in the network. If the application requested reports to be sent at the end of the call (e.g., by means of getInfoReq) these reports will still be sent to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

cause : in TpCallReleaseCause

Specifies the cause of the release.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

Method
deassignCall()

This method requests that the relationship between the application and the call and associated objects be de-assigned. It leaves the call in progress, however, it purges the specified call object so that the application has no further control of call processing. If a call is de-assigned that has call information reports, call leg event reports or call Leg information reports requested, then these reports will be disabled and any related information discarded.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
getInfoReq()

This asynchronous method requests information associated with the call to be provided at the appropriate time (for example, to calculate charging). This method must be invoked before the call is routed to a target address. Two types of reports can be requested; a final report or intermediate reports.

A final call report is sent when the call is ended. The call object will exist after the call is ended if information is required to be sent to the application at the end of the call. The call information will be sent after any call event reports.

Intermediate reports are received when the destination leg or party terminates or when the call ends.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callInfoRequested : in TpCallInfoType

Specifies the call information that is requested.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
setChargePlan()

Set an operator specific charge plan for the call. The charge plan must be set before the call is routed to a target address. Depending on the operator the method can also be used to change the charge plan for ongoing calls.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callChargePlan : in TpCallChargePlan

Specifies the charge plan to use.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
setAdviceOfCharge()

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

aOCInfo : in TpAoCInfo

Specifies two sets of Advice of Charge parameter.

tariffSwitch : in TpDuration

Specifies the tariff switch interval that signifies when the second set of AoC parameters becomes valid.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method

superviseReq()

The application calls this method to supervise a call. The application can set a granted connection time for this call. If an application calls this operation before it routes a call or a user interaction operation the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

time : in TpDuration

Specifies the granted time in milliseconds for the connection.

treatment : in TpCallSuperviseTreatment

Specifies how the network should react after the granted connection time expired.

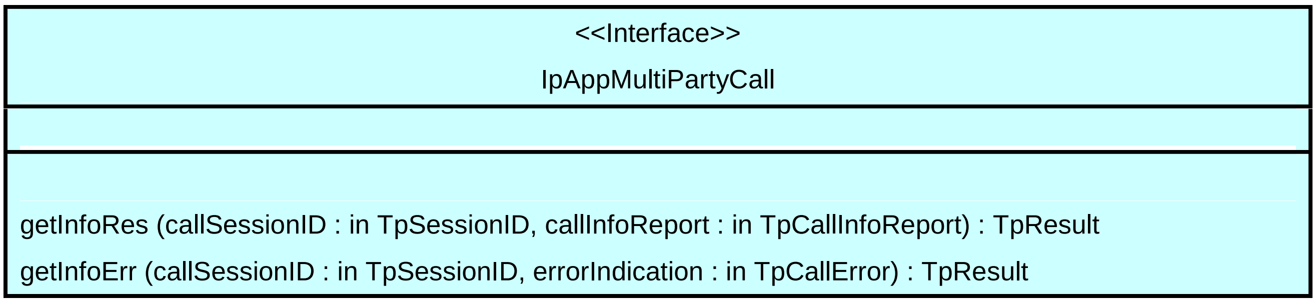
Raises

TpCommonExceptions, P_INVALID_SESSION_ID

7.3.4 Interface Class IpAppMultiPartyCall

Inherits from: IpInterface

The Multi-Party call application interface is implemented by the client application developer and is used to handle call request responses and state reports.



<<Interface>>
IpAppMultiPartyCall
getInfoRes (callSessionID : in TpSessionID, callInfoReport : in TpCallInfoReport) : TpResult
getInfoErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult

superviseRes (callSessionID : in TpSessionID, report : in TpCallSuperviseReport, usedTime : in TpDuration) : TpResult

superviseErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult

callFaultDetected (callSessionID : in TpSessionID, fault : in TpCallFault) : TpResult

callEnded (callSessionID : in TpSessionID, report : in TpCallEndedReport) : TpResult

createAndRouteCallLegErr (callSessionID : in TpSessionID, callLegReference : in TpCallLegIdentifier, errorIndication : in TpCallError) : TpResult

Method
getInfoRes()

This asynchronous method reports time information of the finished call or call attempt as well as release cause depending on which information has been requested by getInfoReq. This information may be used e.g. for charging purposes. The call information will possibly be sent after reporting of all cases where the call or a leg of the call has been disconnected or a routing failure has been encountered.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callInfoReport : in TpCallInfoReport

Specifies the call information requested.

Method
getInfoErr()

This asynchronous method reports that the original request was erroneous, or resulted in an error condition.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Method
superviseRes()

This asynchronous method reports a call supervision event to the application when it has indicated it’s interest in these kind of events.

It is also called when the connection is terminated before the supervision event occurs. Furthermore, this method is invoked as a response to the request also when a tariff switch happens in the network during an active call.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call

report : in TpCallSuperviseReport

Specifies the situation which triggered the sending of the call supervision response.

usedTime : in TpDuration

Specifies the used time for the call supervision (in milliseconds).

Method

superviseErr()

This asynchronous method reports a call supervision error to the application.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Method

callFaultDetected()

This method indicates to the application that a fault in the network has been detected. The call may or may not have been terminated.

The system deletes the call object. Therefore, the application has no further control of call processing.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call in which the fault has been detected.

fault : in TpCallFault

Specifies the fault that has been detected.

Method

callEnded()

This method indicates to the application that the call has terminated in the network.

Note that the event that caused the call to end might have been received separately if the application was monitoring for it.

Parameters

callSessionID : in TpSessionID

Specifies the call sessionID.

report : in TpCallEndedReport

Specifies the reason the call is terminated.

Method

createAndRouteCallLegErr()

This asynchronous method indicates that the request to route the call to the destination party was unsuccessful - the call could not be routed to the destination party (for example, the network was unable to route the call, the parameters were incorrect, the request was refused, etc.). Note that the event cases that can be monitored and correspond to an unsuccessful setup of a connection (e.g. busy, no_answer) will be reported by eventReportRes() and not by this operation.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callLegReference : in TpCallLegIdentifier

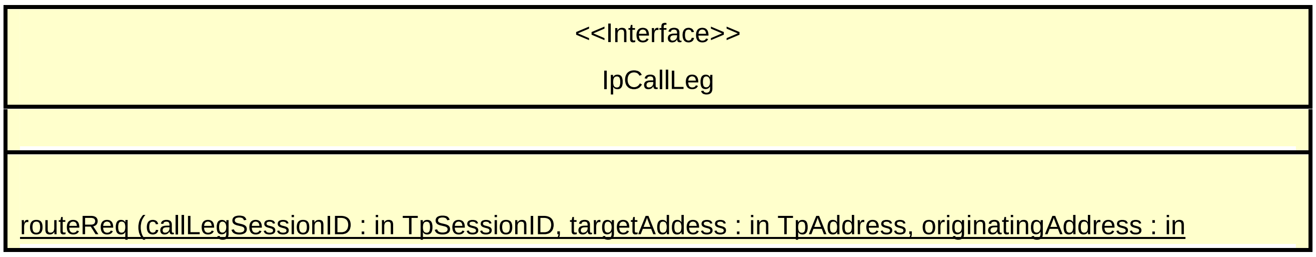
Specifies the reference to the CallLeg interface that was created.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

7.3.5 Interface Class IpCallLeg

Inherits from: The call leg interface represents the logical call leg associating a call with an address. The call leg tracks its own states and allows charging summaries to be accessed. The leg represents the signalling relationship between the call and an address. An application that uses the IpCallLeg interface to set up connections has more control, e.g. by defining leg specific event request and can obtain call leg specific report and events.



<<Interface>>

IpCallLeg

routeReq (callLegSessionID : in TpSessionID, targetAddess : in TpAddress, originatingAddress : in

TpAddress, appInfo : in TpCallAppInfoSet, connectionProperties : in TpCallLegConnectionProperties) : TpResult

eventReportReq (callLegSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet) : TpResult

release (callLegSessionID : in TpSessionID, cause : in TpCallReleaseCause) : TpResult

getInfoReq (callLegSessionID : in TpSessionID, callLegInfoRequested : in TpCallLegInfoType) : TpResult

getCall (callLegSessionID : in TpSessionID, callReference : out TpMultiPartyCallIdentifierRef) : TpResult

attachMedia (callLegSessionID : in TpSessionID) : TpResult

detachMedia (callLegSessionID : in TpSessionID) : TpResult

getLastRedirectedAddress (callLegSessionID : in TpSessionID, redirectedAddress : out TpAddressRef) : TpResult

continueProcessing (callLegSessionID : in TpSessionID) : TpResult

getMoreDialledDigitsReq (callLegSessionID : in TpSessionID, length : in TpInt32) : TpResult

setChargePlan (callLegSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : TpResult

setAdviceOfCharge (callLegSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tarrifSwitch : in TpDuration) : TpResult

superviseReq (callLegSessionID : in TpSessionID, time : in TpDuration, treatment : in TpCallSuperviseTreatment) : TpResult

deassign (callLegSessionID : in TpSessionID) : TpResult

Method

routeReq()

This asynchronous method requests routing of the call leg to the remote party indicated by the targetAddress.

In case the connection to the destination party is established successfully the CallLeg will be either detached or attached to the call based on the attach Mechanism values specified in the connectionProperties parameter.

The extra address information such as originatingAddress is optional. If not present (i.e. the plan is set to P_ADDRESS_PLAN_NOT_PRESENT), the information provided in the corresponding addresses from the route is used, otherwise network or gateway provided addresses will be used.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

targetAddess : in TpAddress

Specifies the destination party to which the call leg should be routed

originatingAddress : in TpAddress

Specifies the address of the originating (calling) party.

appInfo : in TpCallAppInfoSet

Specifies application-related information pertinent to the call leg (such as alerting method, tele-service type, service identities and interaction indicators).

connectionProperties : in TpCallLegConnectionProperties

Specifies the properties of the connection.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

Method

eventReportReq()

This asynchronous method sets, clears or changes the criteria for the events that the call leg object will be set to observe.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

eventsRequested : in TpCallEventRequestSet

Specifies the event specific criteria used by the application to define the events required. Only events that meet these criteria are reported. Examples of events are "address analysed", "answer", "release".

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_EVENT_TYPE, P_INVALID_CRITERIA

Method

release()

This method requests the release of the call leg. If successful, the associated address (party) will be released from the call, and the call leg deleted. Note that in some cases releasing the party may lead to release of the complete call in the network. The application will be informed of this with callEnded().

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

cause : in TpCallReleaseCause

Specifies the cause of the release.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

Method

getInfoReq()

This asynchronous method requests information associated with the call leg to be provided at the appropriate time (for example, to calculate charging). Note: in the call leg information must be accessible before the objects of concern are deleted.

Parameters

`callLegSessionID : in TpSessionID`

Specifies the call leg session ID of the call leg.

`callLegInfoRequested : in TpCallLegInfoType`

Specifies the call leg information that is requested.

Raises

`TpCommonExceptions, P_INVALID_SESSION_ID`

Method

getCall()

This method requests the call associated with this call leg.

Parameters

`callLegSessionID : in TpSessionID`

Specifies the call leg session ID of the call leg.

`callReference : out TpMultiPartyCallIdentifierRef`

Specifies the interface and sessionID of the call associated with this call leg.

Raises

`TpCommonExceptions, P_INVALID_SESSION_ID`

Method

attachMedia()

This method requests that the call leg be attached to its call object. This will allow transmission on all associated bearer connections or media channels to and from other parties in the call. The call leg must be in the connected state for this method to complete successfully.

Parameters

`callLegSessionID : in TpSessionID`

Specifies the sessionID of the call leg to attach to the call.

Raises

`TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE`

Method
detachMedia()

This method will detach the call leg from its call, i.e., this will prevent transmission on any associated bearer connections or media channels to and from other parties in the call. The call leg must be in the connected state for this method to complete successfully.

Parameters

callLegSessionID : in TpSessionID

Specifies the sessionID of the call leg to detach from the call.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

Method
getLastRedirectedAddress()

Queries the last address the leg has been redirected to.

Parameters

callLegSessionID : in TpSessionID

Specifies the call session ID of the call leg.

redirectedAddress : out TpAddressRef

Specifies the last address where the call leg was redirected to.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
continueProcessing()

This operation continues processing of the call leg. Applications can invoke this operation after call leg processing was interrupted due to detection of a notification or event the application subscribed it's interest in.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

Method
getMoreDialledDigitsReq()

This asynchronous method requests to collect further digits and return them to the application. Depending on the administered data, the network may indicate a new call to the gateway if a caller goes off-hook or dialled only a few digits. The application then gets a new call event which contains no digits or only the few dialled digits in the event data. The application should then use this method if it requires more dialled digits, e.g. to perform screening.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call.

length : in TpInt32

Specifies the maximum number of digits to collect.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
setChargePlan()

Set an operator specific charge plan for the cal leg. The charge plan must be set before the call leg is routed to a target address. Depending on the operator the method can also be used to change the charge plan for ongoing calls.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

callChargePlan : in TpCallChargePlan

Specifies the charge plan to use.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
setAdviceOfCharge()

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

aOCInfo : in TpAoCInfo

Specifies two sets of Advice of Charge parameter.

tarrifSwitch : in TpDuration

Specifies the tariff switch interval that signifies when the second set of AoC parameters becomes valid.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
superviseReq()

The application calls this method to supervise a call leg. The application can set a granted connection time for this call. If an application calls this function before it calls a routeReq() or a user interaction function the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

time : in TpDuration

Specifies the granted time in milliseconds for the connection.

treatment : in TpCallSuperviseTreatment

Specifies how the network should react after the granted connection time expired.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Method
deassign()

This method requests that the relationship between the application and the call leg and associated objects be de-assigned. It leaves the call leg in progress, however, it purges the specified call leg object so that the application has no further control of call leg processing. If a call leg is de-assigned that has event reports or call leg information reports requested, then these reports will be disabled and any related information discarded.

The application should always either release or deassign the call leg when it is finished with the call, leg unless callFaultDetected is received by the application.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

7.3.6 Interface Class IpAppCallLeg

Inherits from: IpInterface

IpService

The application call leg interface is implemented by the client application developer and is used to handle responses and errors associated with requests on the call leg in order to be able to receive leg specific information and events.

<<Interface>> IpAppCallLeg
eventReportRes (callLegSessionID : in TpSessionID, eventInfo : in TpCallEventInfo) : TpResult eventReportErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult getInfoRes (callLegSessionID : in TpSessionID, callLegInfoReport : in TpCallLegInfoReport) : TpResult getInfoErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult routeErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult getMoreDialledDigitsRes (callSessionID : in TpSessionID, digits : in TpString) : TpResult getMoreDialledDigitsErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult superviseRes (callLegSessionID : in TpSessionID, report : in TpCallSuperviseReport, usedTime : in TpDuration) : TpResult superviseErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult connectionEnded (callLegSessionID : in TpSessionID, cause : in TpCallReleaseCause) : TpResult

Method
eventReportRes()

This asynchronous method reports that an event has occurred that was requested to be reported (for example, a mid-call event, the party has requested to disconnect, etc.).

Depending on the type of event received, outstanding requests for events are discarded. The exact details of these so-called disarming rules are captured in the data definition of

the event type.

If this method is invoked with a monitor mode of P_MONITOR_MODE_INTERRUPTED, then the APL has control of the call. If the APL does nothing with the call (including its associated legs) within a specified time period (the duration which forms a part of the service level agreement), then the call in the network shall be released and callEnded() shall be invoked, giving a release cause of P_TIMER_EXPIRY.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg on which the event was detected.

eventInfo : in TpCallEventInfo

Specifies data associated with this event.

Method

eventReportErr()

This asynchronous method indicates that the request to manage call leg event reports was unsuccessful, and the reason (for example, the parameters were incorrect, the request was refused, etc.).

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Method

getInfoRes()

This asynchronous method reports all the necessary information requested by the application, for example to calculate charging.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg to which the information relates.

callLegInfoReport : in TpCallLegInfoReport

Specifies the call leg information requested.

Method

getInfoErr()

This asynchronous method reports that the original request was erroneous, or resulted in an error condition.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

Method
`routeErr()`

Parameters

`callLegSessionID : in TpSessionID`

Specifies the call leg session ID of the call leg.

`errorIndication : in TpCallError`

Specifies the error which led to the original request failing.

Method
`getMoreDialledDigitsRes()`

This asynchronous method returns the collected digits to the application.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`digits : in TpString`

Specifies the additional dialled digits if the string length is greater than zero.

Method
`getMoreDialledDigitsErr()`

This asynchronous method reports an error in collecting digits to the application.

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`errorIndication : in TpCallError`

Specifies the error which led to the original request failing.

Method
`superviseRes()`

This asynchronous method reports a call leg supervision event to the application when it has indicated its interest in these kind of events.

It is also called when the connection to a party is terminated before the supervision event occurs. Furthermore, this method is invoked as a response to the request also when a tariff switch happens in the network during an active call.

Parameters

`callLegSessionID : in TpSessionID`

Specifies the call leg session ID of the call leg

`report : in TpCallSuperviseReport`

Specifies the situation which triggered the sending of the call leg supervision response.

`usedTime : in TpDuration`

Specifies the used time for the call leg supervision (in milliseconds).

Method

`superviseErr()`

Parameters

`callLegSessionID : in TpSessionID`

Specifies the call leg session ID of the call leg.

`errorIndication : in TpCallError`

Specifies the error which led to the original request failing.

Method

`connectionEnded()`

This method indicates to the application that the connection has terminated in the network. However, the application may still receive some results (e.g., getInfoRes) related to the call leg. The application is expected to deassign the call leg object after having received the connectionEnded.

Note that the event that caused the connection to end might also be received separately if the application was monitoring for it.

Parameters

`callLegSessionID : in TpSessionID`

Specifies the call leg session ID of the call leg.

`cause : in TpCallReleaseCause`

Specifies the reason the connection is terminated.

7.4 MultiParty Call Control Service State Transition Diagrams

7.4.1 State Transition Diagrams for IpMultiPartyCallControlManager

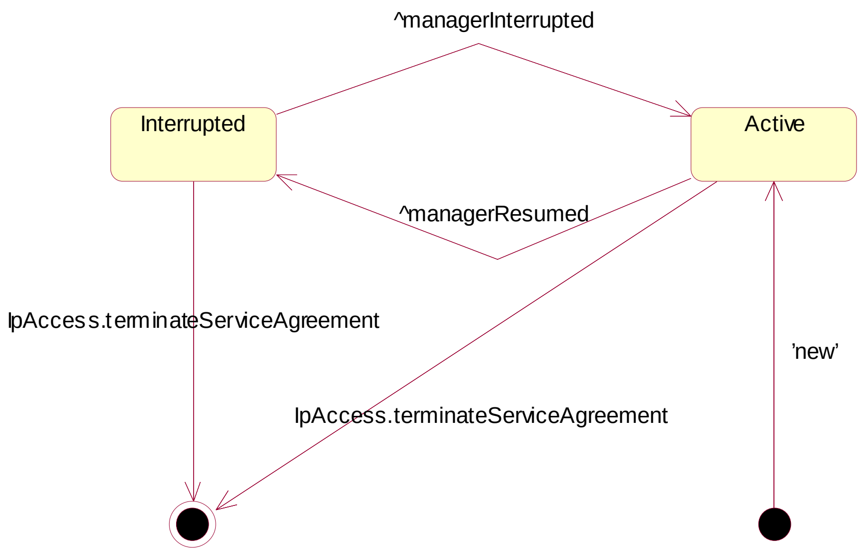

^managerInterrupted
Interrupted
Active
^managerResumed
IpAccess.terminateServiceAgreement
'new'
IpAccess.terminateServiceAgreement

Figure : Application view and the Multi-Party Call Control Manager

7.4.1.1 Active State

In this state a relation between the Application and the Service has been established. The state allows the application to indicate that it is interested in call related events. In case such an event occurs, the Manager will create a Call object with the appropriate number of Call Leg objects and inform the application. The application can also indicate it is no longer interested in certain call related events by calling destroyNotification().

7.4.1.2 Interrupted State

When the Manager is in the Interrupted state it is temporarily unavailable for use. Events requested cannot be forwarded to the application and methods in the API cannot successfully be executed. A number of reasons can cause this: for instance the application receives more notifications from the network than defined in the Service Agreement. Another example is that the Service has detected it receives no notifications from the network due to e.g. a link failure.

7.4.1.3 Overview of allowed methods

Call Control Manager State	Methods applicable

Active	createCall, createNotification, destroyNotification, changeNotification, getNotification, setCallLoadControl
Interrupted	getNotification

7.4.2 State Transition Diagrams for IpMultiPartyCall

The state transition diagram shows the application view on the MultiParty Call object.

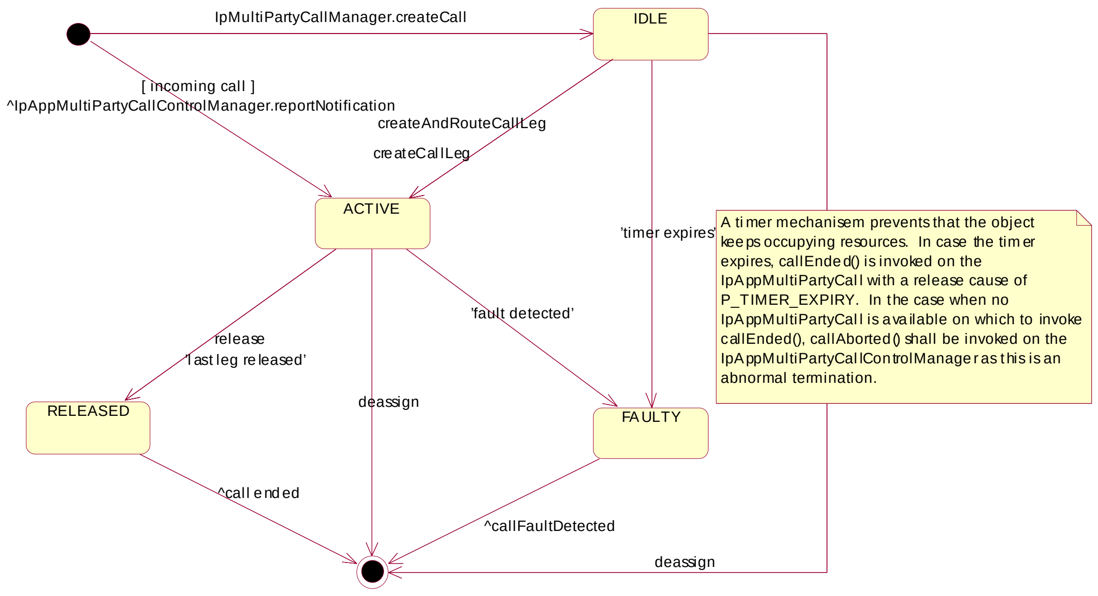

IpMultiPartyCallManager.createCall
IDLE
[incoming call]
^IpAppMultiPartyCallControlManager.reportNotification
createAndRouteCallLeg
createCallLeg
ACTIVE
'timer expires'
'fault detected'
release
'last leg released'
RELEASED
deassign
FAULTY
^call ended
^callFaultDetected
deassign
A timer mechanisem prevents that the object keeps occupying resources. In case the timer expires, callEnded() is invoked on the IpAppMultiPartyCall with a release cause of P_TIMER_EXPIRY. In the case when no IpAppMultiPartyCall is available on which to invoke callEnded(), callAborted() shall be invoked on the IpAppMultiPartyCallControlManager as this is an abnormal termination.

Figure : Application view on the MultiParty Call object

7.4.2.1 IDLE State

In this state the Call object has no Call Leg object associated to it.

The application can request for charging related information reports, call supervision, set the charge plan and set Advice Of Charge indicators. When the first Call Leg object is requested to be created a state transition is made to the Active state.

7.4.2.2 ACTIVE State

In this state the Call object has one or more Call Leg objects associated to it. The application is allowed to create additional Call Leg objects.

Furthermore, the application can request for call supervision. The Application can request charging related information reports, set the charge plan and set Advice Of Charge indicators in this state prior to call establishment.

7.4.2.3 FAULTY State

A transition to this state is made when the Call object is in state IDLE and no requests from the application have been received during a certain period or when a non-recoverable fault was detected during the ACTIVE state.

In case the application requested for call related information previously, the application will be informed that this information is not available through getInfoError or SuperviseError and additionally the application is informed that the call object is transitioning to end state.

7.4.2.4 RELEASED State

In this state the last Call leg object has released or the call itself was released. While the call is in this state, the requested call information will be collected and returned through getInfoReq() and / or superviseReq(). As soon as all information is returned, the application will be informed that the call has ended and Call object transition to the end state.

7.4.2.5 Overview of allowed methods

Methods applicable	**Call Control Call State**	**Call Control Manager State**	**Call Control Call Leg state**
getCallLegs,	Idle, Active, Released	-	
createCallLegs, createAndRouteCallLegReq, setAdviceOfCharge, superviseReq,	Idle, Active	Active	
Release	Active	Active	
Deassign	Idle, Active	-	
GetInfoReq	Idle	Active	
SetChargePlan	Idle, Active	Active	Alerting, Connected

7.4.3 State Transition Diagrams for IpCallLeg

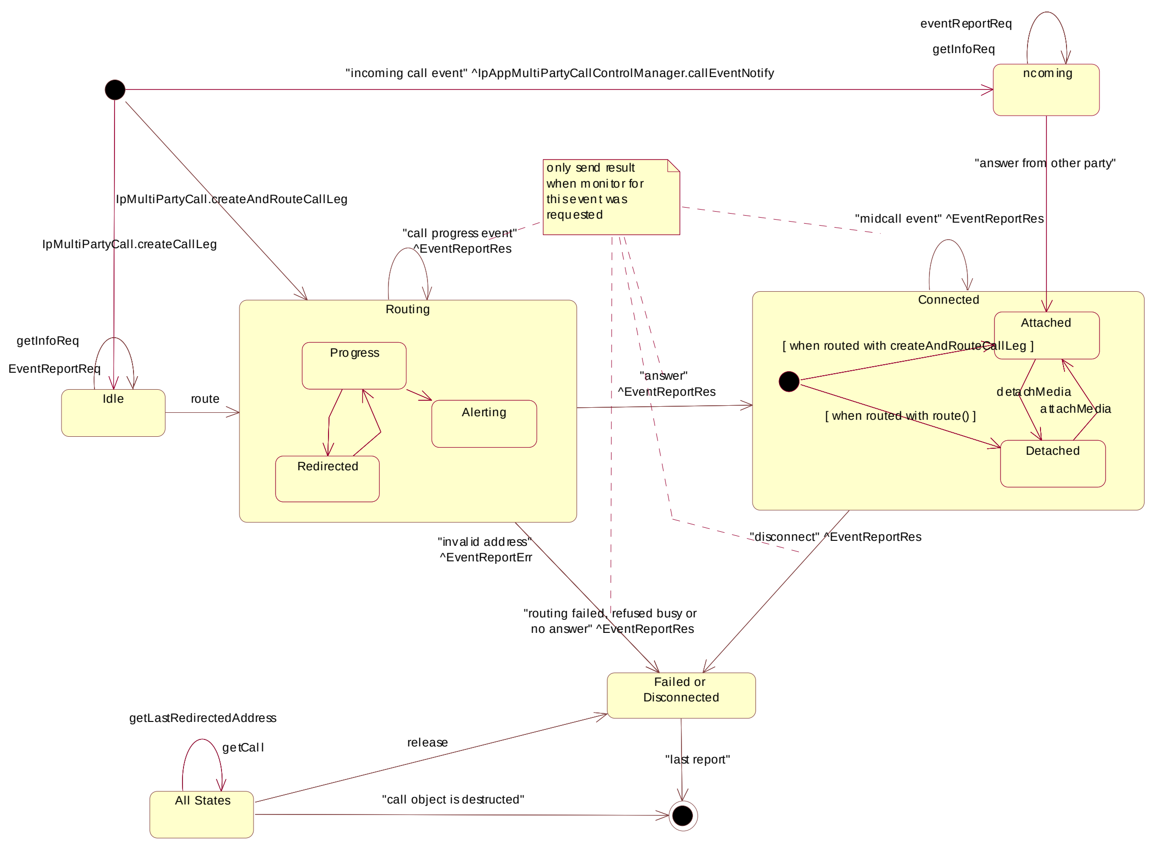

eventReportReq
getInfoReq
"incoming call event" ^IpAppMultiPartyCallControlManager.callEventNotify
ncoming
"answer from other party"
only send result when monitor for this event was requested
"midcall event" ^EventReportRes
IpMultiPartyCall.createAndRouteCallLeg
IpMultiPartyCall.createCallLeg
"call progress event" ^EventReportRes
Routing
Progress
Alerting
Redirected
Connected
Attached
[when routed with createAndRouteCallLeg]
detachMedia
attachMedia
[when routed with route()]
Detached
getInfoReq
EventReportReq
Idle
route
"answer" ^EventReportRes
"invalid address" ^EventReportErr
"disconnect" ^EventReportRes
"routing failed, refused busy or no answer" ^EventReportRes
Failed or Disconnected
getLastRedirectedAddress
getCall
release
All States
"last report"
"call object is destructed"

Figure : Application view on the CallLeg object

7.4.3.1 Idle State

In this state a new CallLeg object has been created and the application has not yet issued a routing request.

7.4.3.2 Routing State

In this state a connection to the call party is being established.

7.4.3.3 Connected State

In this state a connection to the call party is established.

In case the request for the connection was made by createAndRouteCallLeg on the Call object, the call party is also attached to the Call.

In case the request was made by route() the call party still needs to be attached to the Call.

7.4.3.4 Failed or Disconnected State

In this state no connection to the call party could be established or the call party has disconnected.

The reason that no connection could be established can be that an invalid address was specified, the network aborted routing or the call party was busy.

7.4.3.5 Incoming State

This state is only valid for an incoming Call Leg in case and there is no call established to another party.

7.4.3.6 Progress State

In this sub-state the network has indicated there is progress in routing the CallLeg.

7.4.3.7 Alerting State

In this sub-state the network has indicated there the terminal of the party is alerting.

7.4.3.8 Redirected State

In this sub-state the network has indicated the call party has redirected calls to another address.

7.4.3.9 Attached State

In this sub-state the media of the Call Leg object is attached to a Call object.

7.4.3.10 Detached State

In this sub-state the media of the Call Leg object is not attached to a Call object.

7.4.3.11 Overview of allowed methods

State	methods applicable
Idle	routeReq, eventReportReq, release, getInfoReq, getCall, setChargePlan, setAdviceOfCharge, superviseReq, deassign,
Collect_Address	eventReportReq, release, getInfoReq, getCall, continueProcessing, setChargePlan, setAdviceOfCharge, superviseReq, deassign,

Analyse_Address	eventReportReq, release, getInfoReq, getCall, continueProcessing, getMoreDialledDigitsReq, setChargePlan, setAdviceOfCharge, superviseReq, deassign,
Progressing	eventReportReq, release, getInfoReq, getCall, continueProcessing, setChargePlan, setAdviceOfCharge, superviseReq, deassign,
Alerting	eventReportReq, release, getInfoReq, getCall, continueProcessing, setChargePlan, setAdviceOfCharge, superviseReq, deassign,
Active	eventReportReq, release, getInfoReq, getCall, attachMedia, detachMedia,

	getLastRedirectedAddress, continueProcessing, setChargePlan, setAdviceOfCharge, superviseReq, deassign,
Released	getCall, deassign,
Faulty	deassign

7.5 Multi-Party Call Control Service Properties

7.5.1 List of Service Properties

The following table lists properties relevant for the MPCC API. These properties are additional to the properties of the GCC, from which the MPCC is an extension.

Property	Type	Description
P_MAX_CALLLEGS_PER_CALL	INTEGER_SET	Indicates how many parties can be in one call.
P_UI_CALLLEG_BASED	BOOLEAN_SET	Value = TRUE : User interaction can be performed on leg level and a reference to a CallLeg object can be used in the IpUIManager.createUICall() operation. Value = FALSE : No user interaction on leg level is supported.
P_ROUTING_WITH_CALLLEG_OPERATIONS	BOOLEAN_SET	Value = TRUE : the atomic operations for routing a CallLeg are supported {IpMultiPartyCall.createCallLeg(), IpCallLeg.eventReportReq(), IpCallLeg.route(), IpCallLeg.attachMedia()} Value = FALSE : the convenience function has to be used for routing a CallLeg.
P_MEDIA_ATTACH_EXPLICIT	BOOLEAN_SET	Value = TRUE : the CallLeg shall be explicitly attached to a Call. Value = FALSE : the CallLeg is automatically attached to a Call, no IpCallLeg.attachMedia() is needed when a party answers.

7.5.2 Service Property values for the CAMEL Service Environment.

Implementations of the MultiParty Call Control API relying on the CSE shall have the Service Properties outlined above set to the indicated values :

```
P_OPERATION_SET = {
"IpMultiPartyCallControlManager.createNotification",
"IpMultiPartyCallControlManager.destroyNotification",
"IpMultiPartyCallControlManager.changeNotification",
"IpMultiPartyCallControlManager.getNotification",
"IpMultiPartyCallControlManager.setCallLoadControl"
"IpMultiPartyCall.getCallLegs",
"IpMultiPartyCall.createCallLeg",
"IpMultiPartyCall.createAndRouteCallLegReq",
"IpMultiPartyCall.release",
"IpMultiPartyCall.deassignCall",
"IpMultiPartyCall.getInfoReq",
"IpMultiPartyCall.setChargePlan",
"IpMultiPartyCall.setAdviceOfCharge",
"IpMultiPartyCall.superviseReq",
“IpCallLeg.routeReq”,
“IpCallLeg.eventReportReq”,
“IpCallLeg.release”,
“IpCallLeg.getInfoReq”,
“IpCallLeg.getCall”,
“IpCallLeg.continueProcessing”
}

P_TRIGGERING_EVENT_TYPES = {
P_CALL_EVENT_CALL_ATTEMPT,
P_CALL_EVENT_ADDRESS_COLLECTED,
P_CALL_EVENT_ADDRESS_ANALYSED,
P_CALL_EVENT_RELEASE,
}

P_DYNAMIC_EVENT_TYPES = {
P_CALL_EVENT_ANSWER,
P_CALL_EVENT_RELEASE
}

P_ADDRESS_PLAN = {
P_ADDRESS_PLAN_E164
}

P_UI_CALL_BASED = {
TRUE
}

P_UI_AT_ALL_STAGES = {
FALSE
}

P_MEDIA_TYPE = {
P_AUDIO
}

P_MAX_CALLLEGS_PER_CALL = {
0,
2
}

P_UI_CALLLEG_BASED = {
FALSE
}

P_MEDIA_ATTACH_EXPLICIT = {
FALSE
}
```

7.6 Multi-Party Call Control Data Definitions

The present document provides the GCC data definitions necessary to support the API specification.

The general format of a data definition specification is described below.

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

7.6.1 Event Notification Data Definitions

No specific event notification data defined.

7.6.2 Multi-Party Call Control Data Definitions

IpCallLeg

Defines the address of an `IpCallLeg` Interface.

IpCallLegRef

Defines a `Reference` to type IpCallLeg.

IpCallLegRefRef

Defines a `Reference` to type IpCallLegRef.

IpAppCallLeg

Defines the address of an `IpAppCallLeg` Interface.

IpAppCallLegRef

Defines a `Reference` to type IpAppCallLeg.

IpMultiPartyCall

Defines the address of an `IpMultiPartyCall` Interface.

IpMultiPartyCallRef

Defines a `Reference` to type IpMultiPartyCall.

IpAppMultiPartyCall

Defines the address of an `IpAppMultiPartyCall` Interface.

IpAppMultiPartyCallRef

Defines a `Reference` to type IpAppMultiPartyCall.

IpMultiPartyCallControlManager

Defines the address of an `IpMultiPartyCallControlManager` Interface.

IpMultiPartyCallControlManagerRef

Defines a `Reference` to type IpMultiPartyCallControlManager~~Call~~.

IpAppMultiPartyCallControlManager

Defines the address of an `IpAppMultiPartyCallControlManager` Interface.

IpAppMultiPartyCallControlManagerRef

Defines a `Reference` to type IpAppMultiPartyCall ControlManager..

TpAppCallLegRefSet

Defines a `Numbered Set of Data Elements` of IpAppCallLegRef.

IpAppCallLegRef

Defines a `Reference` to type IpAppCallLegRef.

IpAppMultiPartyCallRef

Defines a `Reference` to type IpAppMultiPartyCallRef.

TpMultiPartyCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Call ~~Leg~~ object

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallReference	IpMultiPartyCallRef	This element specifies the interface reference for the Multi-party call object.
CallSessionID	TpSessionID	This element specifies the call session ID.

TpMultiPartyCallIdentifierRef

Defines a `Reference` to type TpMultiPartyCall~~Leg~~Identifier.

TpAppMultiPartyCallBack

Defines the Tagged Choice of Data Elements that references the application callback interfaces

	Tag Element Type	
	TpAppMultiPartyCallBackRefType	

Tag Element Value	Choice Element Type	Choice Element Name
P_APP_CALLBACK_UNDEFINED	NULL	Undefined
P_APP_MULTIPARTY_CALL_CALLBACK	IpAppMultiPartyCallRef	appMultiPartyCall

P_APP_CALL_LEG_CALLBACK	IpAppCallLegRef	appCallLeg
P_APP_CALL_AND_CALL_LEG_CALLBACK	TpAppCallLegCallBack	appMultiPartyCallAndCallLeg

TpAppMultiPartyCallBackRefType

Defines the type application call back interface.

Name	Value	Description
P_APP_CALLBACK_UNDEFINED	0	Application Call back interface undefined
P_APP_MULTIPARTY-CALL_CALLBACK	1	Application Multi-Party Call interface referenced
P_APP_CALL_LEG_CALLBACK	2	Application CallLeg interface referenced
P_APP_CALL_AND_CALL_LEG_CALLBACK	3	Application Multi-Party Call and CallLeg interface referenced

TpAppCallLegCallBack

Defines the Sequence of Data Elements that references a call and a call leg application interface.

Sequence Element Name	Sequence Element Type	
appMultiPartyCall	IpAppMultiPartyCallRef	
appCallLegSet	TpAppCallLegRefSet	Specifies the set of all call leg call back references. First in the set is the reference to the call back of the originating callLeg. In case there is a call back to a destination call leg this will be second in the set.

TpMultiPartyCallIdentifierSet

Defines a Numbered Set of Data Elements of TpMultiPartyCallIdentifier.

TpMultiPartyCallIdentifierSetRef

Defines a Reference to type TpMultiPartyCallIdentifierSet.

TpCallAppInfo

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

	Tag Element Type	
	TpCallAppInfoType	

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

TpCallAppInfoType

Defines the type of call application-related specific information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)
P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. telephony)
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64 kbit/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified service-service information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address
P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS	9	Contains the original address specified by the originating user when launching the call.
P_CALL_APP_REDIRECTING_ADDRESS	10	Contains the address of the user from which the call is diverting.

TpCallEventRequest

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

Sequence Element Name	Sequence Element Type
CallEventType	TpCallEventType
AdditionalCallEventCriteria	TpAdditionalCallEventCriteria
CallMonitorMode	TpCallMonitorMode

TpCallEventRequestSet

Defines a `Numbered Set of Data Elements` of TpCallEventRequest.

TpCallEventType

Defines a specific call event report type.

Name	Value	Description
P_CALL_EVENT_UNDEFINED	0	Undefined
P_CALL_EVENT_CALL_ATTEMPT	1	A Call attempt takes place (e.g. Off-hook event).
P_CALL_EVENT_ADDRESS_COLLECTED	2	The destination address has been collected.
P_CALL_EVENT_ADDRESS_ANALYSED	3	The destination address has been analysed.
~~P_CALL_EVENT_PROGRESS~~	~~4~~	~~Call routing progress event:an indication from the network that progress has been made in routing the call to the requested call party.~~
P_CALL_EVENT_ALERTING	5	Call is alerting at the call party.
P_CALL_EVENT_ANSWER	6	Call answered at address.
P_CALL_EVENT_RELEASE	7	A Call has been released or the call could not be routed.
P_CALL_EVENT_REDIRECTED	8	Call redirected to new address: an indication from the network that the call has been redirected to a new address.
P_CALL_EVENT_SERVICE_CODE	9	Mid-call service code received.
P_CALL_EVENT_QUEUED	10	The Call Event has been queued. (no events are disarmed as a result of this)

The table below defines the disarming rules for dynamic events. In case such an event occurs the table shows which events are disarmed (are not monitored anymore) and should be re-armed by eventReportReq() in case the application is still interested in these events.

Event Occurred	Events Disarmed
P_CALL_EVENT_UNDEFINED	Not Applicable
P_CALL_EVENT_CALL_ATTEMPT	Not applicable, can only be armed as trigger
P_CALL_EVENT_ADDRESS_COLLECTED	P_CALL_EVENT_ADDRESS_COLLECTED
P_CALL_EVENT_ADDRESS_ANALYSED	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED
P_CALL_EVENT_PROGRESS	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED P_CALL_EVENT_PROGRESS
P_CALL_EVENT_ALERTING	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED P_CALL_EVENT_PROGRESS P_CALL_EVENT_ALERTING P_CALL_EVENT_RELEASE with criteria: P_USER_NOT_AVAILABLE P_BUSY P_NOT_REACHABLE P_ROUTING_FAILURE P_CALL_RESTRICTED P_UNAVAILABLE_RESOURCES
P_CALL_EVENT_ANSWER	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED P_CALL_EVENT_PROGRESS P_CALL_EVENT_ALERTING P_CALL_EVENT_RELEASE with criteria: P_USER_NOT_AVAILABLE P_BUSY P_NOT_REACHABLE P_ROUTING_FAILURE P_CALL_RESTRICTED P_UNAVAILABLE_RESOURCES P_NO_ANSWER P_PREMATURE_DISCONNECT P_CALL_EVENT_ANSWER
P_CALL_EVENT_RELEASE	All pending events are disarmed
P_CALL_EVENT_REDIRECTED	P_CALL_EVENT_REDIRECTED
P_CALL_EVENT_SERVICE_CODE	P_CALL_EVENT_SERVICE_CODE

TpAdditionalCallEventCriteria

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

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpInt32	MinAddressLength
P_CALL_EVENT_ADDRESS_ANALYSED	NULL	Undefined
P_CALL_EVENT_PROGRESS	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_RELEASE	TpCallReleaseCauseSet	ReleaseCauseSet
P_CALL_EVENT_REDIRECTED	NULL	Undefined

P_CALL_EVENT_SERVICE_CODE	TpCallServiceCode	ServiceCode

TpCallReleaseCauseSet

Defines a Numbered Set of Data Elements of TpCallReleaseCause.

TpCallEventInfo

Defines the `Sequence of Data Elements` that specify the event report specific information.

Sequence Element Name	Sequence Element Type
CallEventType	TpCallEventType
AdditionalCallEventInfo	TpCallAdditionalEventInfo~~TpAdditionalCallEventInfo~~
CallMonitorMode	TpCallMonitorMode
CallEventTime	TpDateAndTime

TpCallAdditionalEventInfo

Defines the `Tagged Choice of Data Elements` that specify additional call event information for certain types of events.

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpAddress	CollectedAddress
P_CALL_EVENT_ADDRESS_ANALYSED	TpAddress	CalledAddress
P_CALL_EVENT_PROGRESS	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_RELEASE	TpCallReleaseCause	ReleaseCause
P_CALL_EVENT_REDIRECTED	TpAddress	ForwardAddress
P_CALL_EVENT_SERVICE_CODE	TpCallServiceCode	ServiceCode

TpCallNotificationRequest

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

Sequence Element Name	Sequence Element Type	Description
CallNotificationScope	TpCallNotificationScope	Defines the scope of the notification request.
CallEventsRequested	TpCallEventRequestSet	Defines the events which are requested

TpCallNotificationScope

Defines a the sequence of Data elements that specify the scope of a notification request.

Of the addresses only the Plan and the AddrString are used for the purpose of matching the notifications against the criteria.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.
OriginatingAddress	TpAddressRange	Defines the origination address or address range for which the notification is requested.
NotificationCallType	TpNotificationCallType	Defines wheter the notification is requested for a originating or terminating call.

TpNotificationCallType

Defines the type of call for which the notification is requested or reported.

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

TpCallNotificationInfo

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

Sequence Element Name	Sequence Element Type	Description
CallNotificationReportScope	TpCallNotificationReportScope	Defines the scope of the notification report.
CallAppInfo	TpCallAppInfoSet	Contains additonal call info.
CallEventInfo	TpCallEventInfo	Contains the event which is reported.

TpCallNotificationReportScope

Defines the `Sequence of Data Elements` that specify the scope for which a notification report was sent.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddress	Contains the destination address of the call.
OriginatingAddress	TpAddress	Contains the origination address of the call
NotificationCallType	TpNotificationCallType	Indicates if the notification was reported for an originating or terminating call.

TpNotificationRequested

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

Sequence Element Name	Sequence Element Type
AppCallNotificationRequest	TpCallNotificationRequest
AssignmentID	TpInt32

TpNotificationsRequestedSet

Defines a numbered Set of Data Elements of TpNotificationRequested.

TpNotificationsRequestedSetRef

Defines a reference to the type TpNotificationsRequestSet.

TpCallReleaseCause

Defines the reason for which a call is released.

Name	Value	Description
P_UNDEFINED	0	The reason of release isn't known, because no info was received from the network.
P_USER_NOT_AVAILBLE	1	The user isn't available in the network. This means that the number isn't allocated or that the user isn't registered.
P_BUSY	2	The user is busy.
P_NO_ANSWER	3	No answer was received
P_NOT_REACHABLE	4	The user terminal isn't reachable
P_ROUTING_FAILURE	5	A routing failure occurred. For example an invalid address was received
P_PREMATURE_DISCONNECT	6	The user disconnected the call during setup phase.
P_DISCONNECTED	7	Call disconnect by the end user.
P_CALL_RESTRICTED	8	The call was subject of restrictions
P_UNAVAILABLE_RESOURCE	9	No resources where available to establisch the call.
P_GENERAL_FAILURE	10	A general network failure occurred.
P_TIMER_EXPIRY	11	The call was released because an activity timer expired.

TpCallLegIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Call Leg object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallLegReference	IpCallLegRef	This element specifies the interface reference for the callLeg object.
CallLegSessionID	TpSessionID	This element specifies the callLeg session ID.

TpCallLegIdentifierRef

Defines a Reference to type TpCallLegIdentifier.

TpCallLegIdentifierSet

Defines a Numbered Set of Data Elements of TpCallLegIdentifier.

TpCallLegIdentifierSetRef

Defines a Reference to type TpCallLegIdentifierSet.

TpCallLegAttachMechanism

Defines how a CallLeg should be attached to the call.

Name	Value	Description
P_CALLLEG_ATTACH_IMPLICITLY	0	CallLeg should be attached implicitly to the call.
P_CALLLEG_ATTACH_EXPLICITLY	1	CallLeg should be attached explicitly to the call by using the attachMedia() operation. This allows e.g. the application to do first user interaction to the party before he/she is placed in the call.

TpCallLegConnectionProperties

Defines the Sequence of Data Elements that specify the connection properties of the Call Leg object

Sequence Element Name	Sequence Element Type	Sequence Element Description
AttachMechanism	TpCallLegAttachMechanism	Defines how a CallLeg should be attached to the call.

TpCallLegInfoReport

Defines the `Sequence of Data Elements` that specify the call leg information requested.

Sequence Element Name	Sequence Element Type	Description
`CallLegInfoType`	TpCallLegInfoType	The type of the call leg.
`CallLegStartTime`	TpDateAndTime	The time and date when the call leg was started (i.e. the leg was routed).
`CallLegConnectedToResourceTime`	TpDateAndTime	The date and time when the call leg was connected to the resource. If no resource was connected the time is set to an empty string. Either this element is valid or the CallConnectedToAddressTime is valid, depending on whether the report is sent as a result of user interaction.
`CallLegConnectedToAddressTime`	TpDateAndTime	The date and time when the call leg was connected to the destination (i.e. when the destination answered the call). If the destination did not answer, the time is set to an empty string. Either this element is valid or the CallConnectedToResourceTime is valid, depending on whether the report is sent as a result of user interaction.
`CallLegEndTime`	TpDateAndTime	The date and time when the call leg was released.
`ConnectedAddress`	TpAddress	The address of the party associated with the leg. If during the call the connected address was received from the party then this is returned, otherwise the destination address (for legs connected to a destination) or the originating address (for legs connected to the origination) is returned.
`CallLegReleaseCause`	TpCallReleaseCause	The cause of the termination. May be present with P_CALL_LEG_INFO_RELEASE_CAUSE was specified.
`CallAppInfo`	TpCallAppInfoSet	Additional information for the leg. May be present with P_CALL_LEG_INFO_APPINFO was specified.

TpCallLegInfoType

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

Name	Value	Description
`P_CALL_LEG_INFO_UNDEFINED`	00h	Undefined
`P_CALL_LEG_INFO_TIMES`	01h	Relevant call times
`P_CALL_LEG_INFO_RELEASE_CAUSE`	02h	Call leg release cause
`P_CALL_LEG_INFO_ADDRESS`	04h	Call leg connected address
`P_CALL_LEG_INFO_APPINFO`	08h	Call leg application related information

8 Common Call Control Data Types

TpCallAlertingMechanism

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

TpCallBearerService

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

Name	Value	Description
`P_CALL_BEARER_SERVICE_UNKNOWN`	0	`Bearer capability information unknown at this time`
`P_CALL_BEARER_SERVICE_SPEECH`	1	`Speech`

P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTED	2	Unrestricted digital information
P_CALL_BEARER_SERVICE_DIGITALRESTRICTED	3	Restricted digital information
P_CALL_BEARER_SERVICE_AUDIO	4	3.1 kHz audio
P_CALL_BEARER_SERVICE_ DIGITALUNRESTRICTEDTONES	5	Unrestricted digital information with tomes/announcements
P_CALL_BEARER_SERVICE_VIDEO	6	Video

TpCallChargePlan

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

Sequence Element Name	Sequence Element Type	Description
ChargeOrderType	TpCallChargeOrderCategory	Charge order
ChargePerTime	TpChargePerTime	Charge per time. Only applicable when time based charging is selected.
TransparentCharge	TpOctetSet	Operator specific charge plan specification, e.g. charging table name / charging table entry. The associated charge plan data will be send transparently to the charging records. Only applicable when transparent charging is selected.
ChargePlan	TpInt32	Pre-defined charge plan. Example of the charge plan set from which the application can choose could be : (0 = normal user, 1 = silver card user, 2 = gold card user). Only applicable when transparent charging is selected.
Currency	TpString	Currency unit according to ISO-4217:1995
AdditionalInfo	TpOctetSet	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.

Valid Currencies are:

ADP, AED, AFA, ALL, AMD, ANG, AON, AOR, ARS, ATS, AUD, AWG, AZM, BAM,

BBD, BDT, BEF, BGL, BGN, BHD, BIF, BMD, BND, BOB, BOV, BRL, BSD, BTN,

BWP, BYB, BZD, CAD, CDF, CHF, CLF, CLP, CNY, COP, CRC, CUP, CVE, CYP,

CZK, DEM, DJF, DKK, DOP, DZD, ECS, ECV, EEK, EGP, ERN, ESP, ETB, EUR,

FIM, FJD, FKP, FRF, GBP, GEL, GHC, GIP, GMD, GNF, GRD, GTQ, GWP, GYD,

HKD, HNL, HRK, HTG, HUF, IDR, IEP, ILS, INR, IQD, IRR, ISK, ITL, JMD,

JOD, JPY, KES, KGS, KHR, KMF, KPW, KRW, KWD, KYD, KZT, LAK, LBP, LKR,

LRD, LSL, LTL, LUF, LVL, LYD, MAD, MDL, MGF, MKD, MMK, MNT, MOP, MRO,

MTL, MUR, MVR, MWK, MXN, MXV, MYR, MZM, NAD, NGN, NIO, NLG, NOK, NPR,

NZD, OMR, PAB, PEN, PGK, PHP, PKR, PLN, PTE, PYG, QAR, ROL, RUB, RUR,

RWF, SAR, SBD, SCR, SDD, SEK, SGD, SHP, SIT, SKK, SLL, SOS, SRG, STD,

SVC, SYP, SZL, THB, TJR, TMM, TND, TOP, TPE, TRL, TTD, TWD, TZS, UAH, UGX, USD, USN, USS, UYU, UZS, VEB, VND, VUV, WST, XAF, XAG, XAU, XBA, XBB, XBC, XBD, XCD, XDR, XFO, XFU, XOF, XPD, XPF, XPT, XTS, XXX, YER, YUM, ZAL, ZAR, ZMK, ZRN, ZWD.

XXX is used for transactions where no currency is involved.

TpCallChargeOrder

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

	Tag Element Type	
	TpCallChargeOrderCategory	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_CHARGE_PER_TIME	TpChargePerTime	ChargePerTime
P_CALL_CHARGE_TRANSPARENT	TpOctetSet	TransparentCharge
P_CALL_CHARGE_PREDEFINED_SET	TpInt32	ChargePlan

TpCallChargeOrderCategory

Defines the type of charging to be applied

Name	Value	Description
P_CALL_CHARGE_PER_TIME	0	Charge per time
P_CALL_CHARGE_TRANSPARENT	1	Operator specific charge plan specification, e.g. charging table name / charging table entry. The associated charge plan data will be send transparently to the charging records
P_CALL_CHARGE_PREDEFINED_SET	2	Pre-defined charge plan. Example of the charge plan set from which the application can choose could be : (0 = normal user, 1 = silver card user, 2 = gold card user).

TpCallAdditionalChargePlanInfo

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	Description
P_CALL_CHARGE_PER_TIME	TpOctetSet	TimeAdditionalInfo	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.
P_CALL_CHARGE_TRANSPARENT	NULL	Undefined	
P_CALL_CHARGE_PREDEFINED_SET	TpOctetSet	SetAdditionalInfo	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.

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.

TpCallError

Defines the `Sequence of Data Elements` that specify the additional information relating to acall error.

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

TpCallAdditionalErrorInfo

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

	Tag Element Type	
	TpCallErrorType	

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

TpCallErrorType

Defines a specific call error.

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

TpCallFault

Defines the cause of the call fault detected.

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

TpCallInfoReport

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

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

A callInfoReport will be generated at the end of user interaction and at the end of the connection with the associated address. This means that either the destination related information is present or the resource related information, but not both.

TpCallInfoType

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

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

TpCallLoadControlMechanism

Defines the Tagged Choice of Data Elements that specify the applied mechanism and associated parameters.

	Tag Element Type	
	TpCallLoadControlMechanismType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_LOAD_CONTROL_PER_INTERVAL	TpCallLoadControlIntervalRate	CallLoadControlPerInterval

TpCallLoadControlIntervalRate

Defines the call admission rate of the call load control mechanism used. This data type indicates the interval (in milliseconds) between calls that are admitted.

Name	Value	Description
P_CALL_LOAD_CONTROL_ADMIT_NO_CALLS	0	Infinite interval (do not admit any calls)
	1 - 60000	Duration in milliseconds

TpCallLoadControlMechanismType

Defines the type of call load control mechanism to use.

Name	Value	Description
P_CALL_LOAD_CONTROL_PER_INTERVAL	1	admit one call per interval

TpCallMonitorMode

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

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

		continues
P_CALL_MONITOR_MODE_DO_NOT_MONITOR	2	Do not monitor for the event

TpCallNetworkAccessType

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

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

TpCallPartyCategory

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

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

TpCallServiceCode

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

Sequence Element Name	Sequence Element Type
CallServiceCodeType	TpCallServiceCodeType
ServiceCodeValue	TpString

TpCallServiceCodeType

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

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

TpCallSuperviseReport

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

Name	Value	Description
P_CALL_SUPERVISE_TIMEOUT	01h	The call supervision timer has expired
P_CALL_SUPERVISE_CALL_ENDED	02h	The call has ended, either due to timer expiry or call party release. In case the called party disconnects but a follow-on call can still be made also this indication is used.
P_CALL_SUPERVISE_TONE_APPLIED	04h	A warning tone has been applied. This is only sent in combination with P_CALL_SUPERVISE_TIMEOUT
P_CALL_SUPERVISE_UI_FINISHED	0	The user interaction has finished.

TpCallSuperviseTreatment

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

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

TpCallTeleService

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

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

TpCallTreatment

Defines the `Sequence of Data Elements` that specify the the treatment for calls that will be handled only by the network (for example, call which are not admitted by the call load control mechanism).

Sequence Element Name	Sequence Element Type
ReleaseCause	TpCallReleaseCause
AdditionalTreatmentInfo	TpCallAdditionalTreatmentInfo

TpCallTreatmentType

Defines the treatment for calls that will be handled only by the network.

Name	Value	Description
P_CALL_TREATMENT_DEFAULT	0	Default treatment
P_CALL_TREATMENT_RELEASE	1	Release the call
P_CALL_TREATMENT_SIAR	2	Send information to the user, and release the call (Send Info & Release)

TpCallAdditionalTreatmentInfo

Defines the `Tagged Choice of Data Elements` that specify the information to be sent to a call party.

	Tag Element Type	
	TpCallTreatmentType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_TREATMENT_DEFAULT	NULL	Undefined
P_CALL_TREATMENT_RELEASE	NULL	Undefined
P_CALL_TREATMENT_SIAR	TpUIInfo	InformationToSend

Annex A (normative): OMG IDL Description of Call Control SCF

The OMG IDL representation of this interface specification is contained in files contained in archive 2919804IDL.ZIP which accompanies the present document.

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

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

B.1 Interface IpCallControlManager

enableCallNotification (appCallControlManager~~Interface~~ : in IpAppCallControlManagerRef, eventCriteria : in TpCallEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult

createCall (appCall : in IpAppCallRef, callReference : out TpCallIdentifierRef) : TpResult

setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange, assignmentID : out TpAssignmentIDRef) : TpResult

B.2 Interface IpAppCallControlManager

callEventNotify (callReference : in TpCallIdentifier, eventInfo : in TpCallEventInfo, assignmentID : in TpAssignmentID, appCall~~Interface~~ : out IpAppCallRefRef) : TpResult

callOverloadEncountered (assignmentID : in TpAssignmentID) : TpResult

callOverloadCeased (assignmentID : in TpAssignmentID) : TpResult

B.3 Interface IpCall

getMoreDialledDigitsReq (callSessionID : in TpSessionID, length : in TpInt32) : TpResult

B.4 Interface IpAppCall

getMoreDialledDigitsRes (callSessionID : in TpSessionID, digits : in TpString) : TpResult

getMoreDialledDigitsErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : TpResult

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Mar 2001	CN_11	NP-010134	047	--	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	1.0.0
Jun 2001	CN_12	NP-010327	--	--	Approved at TSG CN#12 and placed under Change Control	2.0.0	4.0.0

```
//Source file: common_cc_data.idl
//Date:  12 June 2001


#ifndef __COMMON_CC_DATA_DEFINED
#define __COMMON_CC_DATA_DEFINED



#include "ui_data.idl"
#include "osa.idl"

module org {

      module csapi {

            module cc {

                  module gccs {


                        exception TpGCCSException {
                              TpInt32 exceptionType;
                        };

                  };



                  enum TpCallReleaseCause {
                        P_UNDEFINED,
                        P_USER_NOT_AVAILABLE,
                        P_BUSY,
                        P_NO_ANSWER,
                        P_NOT_REACHABLE,
                        P_ROUTING_FAILURE,
                        P_PREMATURE_DISCONNECT,
                        P_DISCONNECTED,
                        P_CALL_RESTRICTED,
                        P_UNAVAILABLE_RESOURCE,
                        P_GENERAL_FAILURE,
                        P_TIMER_EXPIRY
                  };



                  enum TpCallMonitorMode {
                        P_CALL_MONITOR_MODE_INTERRUPT,
                        P_CALL_MONITOR_MODE_NOTIFY,
                        P_CALL_MONITOR_MODE_DO_NOT_MONITOR
                  };



                  typedef TpInt32 TpCallAlertingMechanism;



                  enum TpCallBearerService {
                        P_CALL_BEARER_SERVICE_UNKNOWN,
                        P_CALL_BEARER_SERVICE_SPEECH,
                        P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTED,

                        P_CALL_BEARER_SERVICE_DIGITALRESTRICTED,

                        P_CALL_BEARER_SERVICE_AUDIO,
                        P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTEDTONES,

                        P_CALL_BEARER_SERVICE_VIDEO
                  };



                  enum TpCallChargeOrderCategory {
                        P_CALL_CHARGE_PER_TIME,
                        P_CALL_CHARGE_TRANSPARENT,
                        P_CALL_CHARGE_PREDEFINED_SET
                  };



                  struct TpCallChargePlan {
                        TpCallChargeOrderCategory ChargeOrderType;
                        TpChargePerTime ChargePerTime;
                        TpOctetSet TransparentCharge;
                        TpInt32 ChargePlan;
                        TpString Currency;
                        TpOctetSet AdditionalInfo;
                  };



                  union TpCallChargeOrder switch(TpCallChargeOrderCategory) {
                        case P_CALL_CHARGE_PER_TIME: TpChargePerTime
ChargePerTime;
                        case P_CALL_CHARGE_TRANSPARENT: TpOctetSet
TransparentCharge;
                        case P_CALL_CHARGE_PREDEFINED_SET: TpInt32 ChargePlan;
                  };



                  enum TpCallErrorType {
                        P_CALL_ERROR_UNDEFINED,
                        P_CALL_ERROR_INVALID_ADDRESS,
                        P_CALL_ERROR_INVALID_STATE
                  };



                  union TpCallAdditionalErrorInfo switch(TpCallErrorType) {
                        case P_CALL_ERROR_INVALID_ADDRESS: TpAddressError
CallErrorInvalidAddress;
                        default: short Dummy;
                  };



                  struct TpCallError {
                        TpDateAndTime ErrorTime;
                        TpCallErrorType ErrorType;
                        TpCallAdditionalErrorInfo AdditionalErorInfo;
                  };



                  struct TpCallEndedReport {
                        TpSessionID CallLegSessionID;
                        TpCallReleaseCause Cause;
                  };



                  enum TpCallFault {
                        P_CALL_FAULT_UNDEFINED,
                        P_CALL_TIMEOUT_ON_RELEASE,
                        P_CALL_TIMEOUT_ON_INTERRUPT
                  };



                  const TpInt32 P_CALL_INFO_UNDEFINED = 0;


                  const TpInt32 P_CALL_INFO_TIMES = 1;


                  const TpInt32 P_CALL_INFO_RELEASE_CAUSE = 2;


                  const TpInt32 P_CALL_INFO_INTERMEDIATE = 4;


                  typedef TpInt32 TpCallInfoType;



                  struct TpCallInfoReport {
                        TpCallInfoType CallInfoType;
                        TpDateAndTime CallInitiationStartTime;
                        TpDateAndTime CallConnectedToResourceTime;
                        TpDateAndTime CallConnectedToDestinationTime;
                        TpDateAndTime CallEndTime;
                        TpCallReleaseCause Cause;
                  };



                  const TpInt32 P_CALL_LOAD_CONTROL_ADMIT_NO_CALLS = 0;


                  enum TpCallLoadControlMechanismType {
                        P_CALL_LOAD_CONTROL_PER_INTERVAL
                  };



                  typedef TpInt32 TpCallLoadControlIntervalRate;



                  union TpCallLoadControlMechanism
switch(TpCallLoadControlMechanismType) {
```

```
                              case P_CALL_LOAD_CONTROL_PER_INTERVAL:
TpCallLoadControlIntervalRate CallLoadControlPerInterval;
                  };



                  enum TpCallNetworkAccessType {
                        P_CALL_NETWORK_ACCESS_TYPE_UNKNOWN,
                        P_CALL_NETWORK_ACCESS_TYPE_POT,
                        P_CALL_NETWORK_ACCESS_TYPE_ISDN,
                        P_CALL_NETWORK_ACCESS_TYPE_DIALUPINTERNET,

                        P_CALL_NETWORK_ACCESS_TYPE_XDSL,
                        P_CALL_NETWORK_ACCESS_TYPE_WIRELESS
                  };



                  enum TpCallPartyCategory {
                        P_CALL_PARTY_CATEGORY_UNKNOWN,
                        P_CALL_PARTY_CATEGORY_OPERATOR_F,
                        P_CALL_PARTY_CATEGORY_OPERATOR_E,
                        P_CALL_PARTY_CATEGORY_OPERATOR_G,
                        P_CALL_PARTY_CATEGORY_OPERATOR_R,
                        P_CALL_PARTY_CATEGORY_OPERATOR_S,
                        P_CALL_PARTY_CATEGORY_ORDINARY_SUB,
                        P_CALL_PARTY_CATEGORY_PRIORITY_SUB,
                        P_CALL_PARTY_CATEGORY_DATA_CALL,
                        P_CALL_PARTY_CATEGORY_TEST_CALL,
                        P_CALL_PARTY_CATEGORY_PAYPHONE
                  };



                  enum TpCallServiceCodeType {
                        P_CALL_SERVICE_CODE_UNDEFINED,
                        P_CALL_SERVICE_CODE_DIGITS,
                        P_CALL_SERVICE_CODE_FACILITY,
                        P_CALL_SERVICE_CODE_U2U,
                        P_CALL_SERVICE_CODE_HOOKFLASH,
                        P_CALL_SERVICE_CODE_RECALL
                  };



                  struct TpCallServiceCode {
                        TpCallServiceCodeType CallServiceCodeType;
                        TpString ServiceCodeValue;
                  };



                  enum TpCallTeleService {
                        P_CALL_TELE_SERVICE_UNKNOWN,
                        P_CALL_TELE_SERVICE_TELEPHONY,
                        P_CALL_TELE_SERVICE_FAX_2_3,
                        P_CALL_TELE_SERVICE_FAX_4_I,
                        P_CALL_TELE_SERVICE_FAX_4_II_III,
                        P_CALL_TELE_SERVICE_VIDEOTEX_SYN,
                        P_CALL_TELE_SERVICE_VIDEOTEX_INT,
                        P_CALL_TELE_SERVICE_TELEX,
                        P_CALL_TELE_SERVICE_MHS,
                        P_CALL_TELE_SERVICE_OSI,
                        P_CALL_TELE_SERVICE_FTAM,
                        P_CALL_TELE_SERVICE_VIDEO,
                        P_CALL_TELE_SERVICE_VIDEO_CONF,
                        P_CALL_TELE_SERVICE_AUDIOGRAPH_CONF,

                        P_CALL_TELE_SERVICE_MULTIMEDIA,
                        P_CALL_TELE_SERVICE_CS_INI_H221,
                        P_CALL_TELE_SERVICE_CS_SUB_H221,
                        P_CALL_TELE_SERVICE_CS_INI_CALL,
                        P_CALL_TELE_SERVICE_DATATRAFFIC,
                        P_CALL_TELE_SERVICE_EMERGENCY_CALLS,

                        P_CALL_TELE_SERVICE_SMS_MT_PP,
                        P_CALL_TELE_SERVICE_SMS_MO_PP,
                        P_CALL_TELE_SERVICE_CELL_BROADCAST,
                        P_CALL_TELE_SERVICE_ALT_SPEECH_FAX_3,

                        P_CALL_TELE_SERVICE_AUTOMATIC_FAX_3,

                        P_CALL_TELE_SERVICE_VOICE_GROUP_CALL,

                        P_CALL_TELE_SERVICE_VOICE_BROADCAST
                  };


                  const TpInt32 P_CALL_SUPERVISE_TIMEOUT = 1;


                  const TpInt32 P_CALL_SUPERVISE_CALL_ENDED = 2;


                  const TpInt32 P_CALL_SUPERVISE_TONE_APPLIED = 4;


                  const TpInt32 P_CALL_SUPERVISE_UI_FINISHED = 8;


                  typedef TpInt32 TpCallSuperviseReport;


                  const TpInt32 P_CALL_SUPERVISE_RELEASE = 1;


                  const TpInt32 P_CALL_SUPERVISE_RESPOND = 2;


                  const TpInt32 P_CALL_SUPERVISE_APPLY_TONE = 4;


                  typedef TpInt32 TpCallSuperviseTreatment;


                  enum TpCallTreatmentType {
                        P_CALL_TREATMENT_DEFAULT,
                        P_CALL_TREATMENT_RELEASE,
                        P_CALL_TREATMENT_SIAR
```

```
                  };



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



                  struct TpCallTreatment {
                        TpCallTreatmentType CallTreatmentType;
                        TpCallReleaseCause ReleaseCause;
                        TpCallAdditionalTreatmentInfo AdditionalTreatmentInfo;
                  };

                  union TpCallAdditionalChargePlanInfo
switch(TpCallChargeOrderCategory) {
                        case P_CALL_CHARGE_PER_TIME: TpOctetSet
TimeAdditionalInfo;
                        case P_CALL_CHARGE_PREDEFINED_SET: TpOctetSet
SetAdditionalInfo;
                        default: any Dummy;
                  };

            };

      };

};

#endif
```

```
//Source file: gcc_data.idl
//Date:  12 June 2001

#ifndef __GCC_DATA_DEFINED
#define __GCC_DATA_DEFINED



#include "common_cc_data.idl"
#include "osa.idl"

module org {

      module csapi {



            module cc {

                  module gccs {


                        const TpInt32 P_EVENT_NAME_UNDEFINED = 0;


                        const TpInt32 P_EVENT_GCCS_OFFHOOK_EVENT = 1;


                        const TpInt32 P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT = 2;


                        const TpInt32 P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT = 4;


                        const TpInt32 P_EVENT_GCCS_CALLED_PARTY_BUSY = 8;


                        const TpInt32 P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE = 
16;


                        const TpInt32 P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY = 
32;


                        const TpInt32 P_EVENT_GCCS_ROUTE_SELECT_FAILURE = 64;


                        const TpInt32 P_EVENT_GCCS_ANSWER_FROM_CALL_PARTY = 128;



                        typedef TpInt32 TpCallEventName;



                        enum TpCallNotificationType {
                              P_ORIGINATING,
                              P_TERMINATING
                        };
                        struct TpCallEventCriteria {
                              TpAddressRange DestinationAddress;
                              TpAddressRange OriginatingAddress;
                              TpCallEventName CallEventName;
                              TpCallNotificationType CallNotificationType;

                              TpCallMonitorMode MonitorMode;
                        };

                        struct TpCallEventCriteriaResult {
                              TpCallEventCriteria CallEventCriteria;
                              TpInt32 AssignmentID;
                        };



                        typedef sequence <TpCallEventCriteriaResult>
TpCallEventCriteriaResultSet;

                        enum TpCallAppInfoType {
                              P_CALL_APP_UNDEFINED,
                              P_CALL_APP_ALERTING_MECHANISM,
                              P_CALL_APP_NETWORK_ACCESS_TYPE,
                              P_CALL_APP_TELE_SERVICE,
                              P_CALL_APP_BEARER_SERVICE,
                              P_CALL_APP_PARTY_CATEGORY,
                              P_CALL_APP_PRESENTATION_ADDRESS,
                              P_CALL_APP_GENERIC_INFO,
                              P_CALL_APP_ADDITIONAL_ADDRESS
                        };

                        union TpCallAppInfo switch(TpCallAppInfoType) {
                              case P_CALL_APP_ALERTING_MECHANISM:
TpCallAlertingMechanism CallAppAlertingMechanism;
                              case P_CALL_APP_NETWORK_ACCESS_TYPE:
TpCallNetworkAccessType CallAppNetworkAccessType;
                              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;
                              default: short Dummy;
                        };

                        typedef sequence<TpCallAppInfo> TpCallAppInfoSet;



                        struct TpCallReleaseCause {
                              TpInt32 Value;
                              TpInt32 Location;
                        };



                        enum TpCallReportType {
                              P_CALL_REPORT_UNDEFINED,

                              P_CALL_REPORT_PROGRESS,
                              P_CALL_REPORT_ALERTING,
                              P_CALL_REPORT_ANSWER,
                              P_CALL_REPORT_BUSY,
                              P_CALL_REPORT_NO_ANSWER,

                              P_CALL_REPORT_DISCONNECT,

                              P_CALL_REPORT_REDIRECTED,

                              P_CALL_REPORT_SERVICE_CODE,

                              P_CALL_REPORT_ROUTING_FAILURE,

                              P_CALL_REPORT_QUEUED
                        };



                        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: TpCallServiceCode
ServiceCode;
                              case P_CALL_REPORT_ROUTING_FAILURE:
TpCallReleaseCause RoutingFailure;
                              default: short Dummy;
                        };



                        struct TpCallReport {
                              TpCallMonitorMode MonitorMode;
                              TpDateAndTime CallEventTime;

                              TpCallReportType CallReportType;
                              TpCallAdditionalReportInfo AdditionalReportInfo;
                        };



                        union TpCallAdditionalReportCriteria
switch(TpCallReportType) {
                              case P_CALL_REPORT_NO_ANSWER: TpDuration
NoAnswerDuration;
                              case P_CALL_REPORT_SERVICE_CODE: TpCallServiceCode
ServiceCode;
                              default: short Dummy;
                        };
                        struct TpCallReportRequest {
                              TpCallMonitorMode MonitorMode;
                              TpCallReportType CallReportType;
                              TpCallAdditionalReportCriteria
AdditionalReportCriteria;
                        };



                        typedef sequence <TpCallReportRequest>
TpCallReportRequestSet;



                        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;
                        struct TpCallEventInfo {
                              TpAddress DestinationAddress;
                              TpAddress OriginatingAddress;
                              TpAddress OriginalDestinationAddress;
                              TpAddress RedirectingAddress;
                              TpCallAppInfoSet CallAppInfo;
                              TpCallEventName CallEventName;
                              TpCallNotificationType CallNotificationType;
                              TpCallMonitorMode MonitorMode;
                        };

                        enum TpCallPartyToCharge {
                              P_CALL_PARTY_ORIGINATING,
                              P_CALL_PARTY_DESTINATION
                        };

                        struct TpCallChargePlan {
                              TpCallChargeOrderCategory ChargeOrderType;
                              TpChargePerTime ChargePerTime;
                              TpCallPartyToCharge PartyToCharge;
                              TpOctetSet TransparentCharge;
                              TpInt32 ChargePlan;
                              TpString Currency;
                              TpOctetSet AdditionalInfo;
                        };

                        struct TpCallEndedReport {
                              TpSessionID CallLegSessionID;
                              TpCallReleaseCause Cause;
                        };

                        struct TpCallInfoReport {
                              TpCallInfoType CallInfoType;
                              TpDateAndTime CallInitiationStartTime;
                              TpDateAndTime CallConnectedToResourceTime;
                              TpDateAndTime CallConnectedToDestinationTime;
                              TpDateAndTime CallEndTime;
                              TpCallReleaseCause Cause;
                        };

                        struct TpCallTreatment {
                              TpCallTreatmentType CallTreatmentType;
                              TpCallReleaseCause ReleaseCause;
                              TpCallAdditionalTreatmentInfo
AdditionalTreatmentInfo;
                        };

                  };




            };

      };

};

#endif
```

```
//Source file: gcc_interfaces.idl
//Date:  12 June 2001


#ifndef __GCC_INTERFACES_DEFINED
#define __GCC_INTERFACES_DEFINED



#include "osa.idl"
#include "common_cc_data.idl"
#include "gcc_data.idl"


module org {

      module csapi {

            module cc {

                  module gccs {


                        interface IpAppCall : IpInterface {

                              void routeRes (
                                    in TpSessionID callSessionID,
                                    in TpCallReport eventReport,
                                    in TpSessionID callLegSessionID
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void routeErr (
                                    in TpSessionID callSessionID,
                                    in TpCallError errorIndication,
                                    in TpSessionID callLegSessionID
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void getCallInfoRes (
                                    in TpSessionID callSessionID,
                                    in TpCallInfoReport callInfoReport
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void getCallInfoErr (
                                    in TpSessionID callSessionID,
                                    in TpCallError errorIndication
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void superviseCallRes (
                                    in TpSessionID callSessionID,
                                    in TpCallSuperviseReport report,
                                    in TpDuration usedTime
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void superviseCallErr (
                                    in TpSessionID callSessionID,
                                    in TpCallError errorIndication
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void callFaultDetected (
                                    in TpSessionID callSessionID,
                                    in TpCallFault fault
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void getMoreDialledDigitsRes (
                                    in TpSessionID callSessionID,
                                    in TpString digits
                                    )
                                    raises (TpGeneralException,TpGCCSException);


                              void getMoreDialledDigitsErr (
                                    in TpSessionID callSessionID,
                                    in TpCallError errorIndication
                                    )
                                    raises (TpGeneralException,TpGCCSException);


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

                        };



                        interface IpCall : IpService {

                              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);


                              void release (
                                    in TpSessionID callSessionID,
                                    in TpCallReleaseCause cause
                                    )
```

```
                                    raises (TpGCCSException,TpGeneralException);


                              void deassignCall (
                                    in TpSessionID callSessionID
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void getCallInfoReq (
                                    in TpSessionID callSessionID,
                                    in TpCallInfoType callInfoRequested
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void setCallChargePlan (
                                    in TpSessionID callSessionID,
                                    in TpCallChargePlan callChargePlan
                                    )
                                    raises (TpGCCSException,TpGeneralException);


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


                              void getMoreDialledDigitsReq (
                                    in TpSessionID callSessionID,
                                    in TpInt32 length
                                    )
                                    raises (TpGeneralException,
TpGCCSException);


                              void superviseCallReq (
                                    in TpSessionID callSessionID,
                                    in TpDuration time,
                                    in TpCallSuperviseTreatment treatment
                                    )
                                    raises (TpGCCSException,TpGeneralException);

                        };


                        struct TpCallIdentifier {
                              TpSessionID CallSessionID;
                              IpCall CallReference;
                        };



                        interface IpAppCallControlManager : IpInterface {

                              void callAborted (
                                    in TpSessionID callReference
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void callEventNotify (
                                    in TpCallIdentifier callReference,
                                    in TpCallEventInfo eventInfo,
                                    in TpAssignmentID assignmentID,
                                    out IpAppCall appCall
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void callNotificationInterrupted ()

                                    raises (TpGCCSException,TpGeneralException);


                              void callNotificationContinued ();


                              void callOverloadEncountered (
                                    in TpAssignmentID assignmentID
                                    )
                                    raises (TpGeneralException,TpGCCSException);


                              void callOverloadCeased (
                                    in TpAssignmentID assignmentID
                                    )
                                    raises (TpGeneralException,TpGCCSException);

                        };



                        interface IpCallControlManager : IpService {

                              void createCall (
                                    in IpAppCall appCall,
                                    out TpCallIdentifier callReference
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void enableCallNotification (
                                    in IpAppCallControlManager
appCallControlManager,

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


                              void disableCallNotification (
                                    in TpAssignmentID assignmentID
                                    )
                                    raises (TpGCCSException,TpGeneralException);


                              void setCallLoadControl (
                                    in TpDuration duration,
                                    in TpCallLoadControlMechanism mechanism,
                                    in TpCallTreatment treatment,
                                    in TpAddressRange addressRange,
                                    out TpAssignmentID assignmentID
                                    )
                                    raises (TpGeneralException,TpGCCSException);


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


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

                        };


                  };

            };

      };

};

#endif
```

```
//Source file: mpcc_data.idl
//Date:  12 June 2001


#ifndef __MPCC_DATA_DEFINED
#define __MPCC_DATA_DEFINED



#include "osa.idl"
#include "common_cc_data.idl"

module org {

      module csapi {

            module cc {


                  enum TpCallAppInfoType {
                        P_CALL_APP_UNDEFINED,
                        P_CALL_APP_ALERTING_MECHANISM,
                        P_CALL_APP_NETWORK_ACCESS_TYPE,
                        P_CALL_APP_TELE_SERVICE,
                        P_CALL_APP_BEARER_SERVICE,
                        P_CALL_APP_PARTY_CATEGORY,
                        P_CALL_APP_PRESENTATION_ADDRESS,
                        P_CALL_APP_GENERIC_INFO,
                        P_CALL_APP_ADDITIONAL_ADDRESS,
                        P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS,
                        P_CALL_APP_REDIRECTING_ADDRESS
                  };



                  union TpCallAppInfo switch(TpCallAppInfoType) {
                        case P_CALL_APP_ALERTING_MECHANISM:
TpCallAlertingMechanism CallAppAlertingMechanism;
                        case P_CALL_APP_NETWORK_ACCESS_TYPE:
TpCallNetworkAccessType CallAppNetworkAccessType;
                        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_ORIGINAL_DESTINATION_ADDRESS: TpAddress
CallAppOriginalDestinationAddress;
                        case P_CALL_APP_REDIRECTING_ADDRESS: TpAddress
CallAppRedirectingAddress;
                        default: short Dummy;
                  };
                  typedef sequence <TpCallAppInfo> TpCallAppInfoSet;



                  enum TpCallEventType {
                        P_CALL_EVENT_UNDEFINED,
                        P_CALL_EVENT_CALL_ATTEMPT,
                        P_CALL_EVENT_ADDRESS_COLLECTED,
                        P_CALL_EVENT_ADDRESS_ANALYSED,
                        P_CALL_EVENT_ALERTING,
                        P_CALL_EVENT_ANSWER,
                        P_CALL_EVENT_RELEASE,
                        P_CALL_EVENT_REDIRECTED,
                        P_CALL_EVENT_SERVICE_CODE,
                        P_CALL_EVENT_QUEUED
                  };



                  union TpCallAdditionalEventInfo switch(TpCallEventType) {
                        case P_CALL_EVENT_ADDRESS_COLLECTED: TpAddress 
CollectedAddress;
                        case P_CALL_EVENT_ADDRESS_ANALYSED: TpAddress 
CalledAddress;
                        case P_CALL_EVENT_RELEASE: TpCallReleaseCause 
ReleaseCause;
                        case P_CALL_EVENT_REDIRECTED: TpAddress ForwardAddress;
                        case P_CALL_EVENT_SERVICE_CODE: TpCallServiceCode 
ServiceCode;
                        default: short Dummy;
                  };



                  enum TpNotificationCallType {
                        P_ORIGINATING,
                        P_TERMINATING
                  };



                  struct TpCallNotificationScope {
                        TpAddressRange DestinationAddress;
                        TpAddressRange OriginatingAddress;
                        TpNotificationCallType NotificationCallType;
                  };



                  struct TpCallNotificationReportScope {
                        TpAddress DestinationAddress;
                        TpAddress OriginatingAddress;
                        TpNotificationCallType NotificationCallType;
                  };



                  typedef sequence<TpCallReleaseCause> TpCallReleaseCauseSet;
                  union TpAdditionalCallEventCriteria switch(TpCallEventType) {
                        case P_CALL_EVENT_ADDRESS_COLLECTED: TpInt32
MinAddressLength;
                        case P_CALL_EVENT_RELEASE: TpCallReleaseCauseSet
ReleaseCauseSet;
                        case P_CALL_EVENT_SERVICE_CODE: TpCallServiceCode
ServiceCode;
                        default: short Dummy;
                  };



                  struct TpCallEventRequest {
                        TpCallEventType CallEventType;
                        TpAdditionalCallEventCriteria
AdditionalCallEventCriteria;
                        TpCallMonitorMode CallMonitorMode;
                  };



                  typedef sequence <TpCallEventRequest> TpCallEventRequestSet;



                  struct TpCallNotificationRequest {
                        TpCallNotificationScope CallNotificationScope;
                        TpCallEventRequestSet CallEventsRequested;
                  };



                  struct TpNotificationRequested {
                        TpCallNotificationRequest AppCallNotificationRequest;
                        TpInt32 AssignmentID;
                  };



                  typedef sequence <TpNotificationRequested>
TpNotificationRequestedSet;

                  enum TpCallLegAttachMechanism {
                        P_CALLLEG_ATTACH_IMPLICITLY,
                        P_CALLLEG_ATTACH_EXPLICITLY
                  };



                  struct TpCallLegConnectionProperties {
                        TpCallLegAttachMechanism AttachMechanism;
                  };



                  const TpInt32 P_CALL_LEG_INFO_UNDEFINED = 0;


                  const TpInt32 P_CALL_LEG_INFO_TIMES = 1;
                  const TpInt32 P_CALL_LEG_INFO_RELEASE_CAUSE = 2;


                  const TpInt32 P_CALL_LEG_INFO_ADDRESS = 4;


                  const TpInt32 P_CALL_LEG_INFO_APPINFO = 8;


                  typedef TpInt32 TpCallLegInfoType;



                  struct TpCallLegInfoReport {
                        TpCallLegInfoType CallLegInfoType;
                        TpDateAndTime CallLegStartTime;
                        TpDateAndTime CallLegConnectedToResourceTime;
                        TpDateAndTime CallLegConnectedToAddressTime;
                        TpDateAndTime CallLegEndTime;
                        TpAddress ConnectedAddress;
                        TpCallReleaseCause CallLegReleaseCause;
                        TpCallAppInfoSet CallAppInfo;
                  };

                  struct TpCallEventInfo {
                        TpCallEventType CallEventType;
                        TpCallAdditionalEventInfo AdditionalCallEventInfo;
                        TpCallMonitorMode CallMonitorMode;
                        TpDateAndTime CallEventTime;
                  };



                  struct TpCallNotificationInfo {
                        TpCallNotificationReportScope
CallNotificationReportScope;
                        TpCallAppInfoSet CallAppInfo;
                        TpCallEventInfo CallEventInfo;
                  };

            };

      };

};

#endif
```

```
//Source file: mpcc_interfaces.idl
//Date:  12 June 2001


#ifndef __MPCC_INTERFACES_DEFINED
#define __MPCC_INTERFACES_DEFINED



#include "osa.idl"
#include "common_cc_data.idl"
#include "mpcc_data.idl"

module org {

      module csapi {

            module cc {

                  module mpccs {

                        interface IpMultiPartyCall;
                        interface IpCallLeg;
                        interface IpAppMultiPartyCall;
                        interface IpAppCallLeg;


                        struct TpCallLegIdentifier {
                              IpCallLeg CallLegReference;

                              TpSessionID CallLegSessionID;
                        };

                        typedef sequence <TpCallLegIdentifier>
TpCallLegIdentifierSet;

                        struct TpMultiPartyCallIdentifier {
                              IpMultiPartyCall CallReference;
                              TpSessionID CallSessionID;
                        };


                        typedef sequence <TpMultiPartyCallIdentifier>
TpMultiPartyCallIdentifierSet;

                        enum TpAppMultiPartyCallBackRefType {
                              P_APP_CALLBACK_UNDEFINED,
                              P_APP_MULTIPARTY_CALL_CALLBACK,
                              P_APP_CALL_LEG_CALLBACK,
                              P_APP_CALL_AND_CALL_LEG_CALLBACK
                        };

                        typedef sequence <IpAppCallLeg> TpAppCallLegRefSet;

                        struct TpAppCallLegCallBack {
                              IpAppMultiPartyCall appMultiPartyCall;
                              TpAppCallLegRefSet appCallLegSet;
                        };

                        union TpAppMultiPartyCallBack
switch(TpAppMultiPartyCallBackRefType) {
```

```
                              case P_APP_MULTIPARTY_CALL_CALLBACK:
IpAppMultiPartyCall appMultiPartyCall;
                              case P_APP_CALL_LEG_CALLBACK: IpAppCallLeg
appCallLeg;
                              case P_APP_CALL_AND_CALL_LEG_CALLBACK:
TpAppCallLegCallBack appMultiPartyCallAndCallLeg;
                              default: short Dummy;
                        };



                        interface IpAppCallLeg : IpInterface, IpService {

                              void eventReportRes (
                                    in TpSessionID callLegSessionID,
                                    in TpCallEventInfo eventInfo
                                    );


                              void eventReportErr (
                                    in TpSessionID callLegSessionID,
                                    in TpCallError errorIndication
                                    );


                              void getInfoRes (
                                    in TpSessionID callLegSessionID,
                                    in TpCallLegInfoReport callLegInfoReport
                                    );


                              void getInfoErr (
                                    in TpSessionID callLegSessionID,
                                    in TpCallError errorIndication
                                    );


                              void routeErr (
                                    in TpSessionID callLegSessionID,
                                    in TpCallError errorIndication
                                    );


                              void getMoreDialledDigitsRes (
                                    in TpSessionID callSessionID,
                                    in TpString digits
                                    );


                              void getMoreDialledDigitsErr (
                                    in TpSessionID callSessionID,
                                    in TpCallError errorIndication
                                    );


                              void superviseRes (
                                    in TpSessionID callLegSessionID,
                                    in TpCallSuperviseReport report,
                                    in TpDuration usedTime
                                    );
                              void superviseErr (
                                    in TpSessionID callLegSessionID,
                                    in TpCallError errorIndication
                                    );


                              void connectionEnded (
                                    in TpSessionID callLegSessionID,
                                    in TpCallReleaseCause cause
                                    );

                        };



                        interface IpCallLeg {

                              void routeReq (
                                    in TpSessionID callLegSessionID,
                                    in TpAddress targetAddess,
                                    in TpAddress originatingAddress,
                                    in TpCallAppInfoSet appInfo,
                                    in TpCallLegConnectionProperties
connectionProperties
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE);


                              void eventReportReq (
                                    in TpSessionID callLegSessionID,
                                    in TpCallEventRequestSet eventsRequested
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_EVENT_TYPE, P_INVALID_CRITERIA);


                              void release (
                                    in TpSessionID callLegSessionID,
                                    in TpCallReleaseCause cause
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE);


                              void getInfoReq (
                                    in TpSessionID callLegSessionID,
                                    in TpCallLegInfoType callLegInfoRequested
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void getCall (
                                    in TpSessionID callLegSessionID,
                                    out TpMultiPartyCallIdentifier callReference

                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void attachMedia (
                                    in TpSessionID callLegSessionID
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE);


                              void detachMedia (
                                    in TpSessionID callLegSessionID
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE);


                              void getLastRedirectedAddress (
                                    in TpSessionID callLegSessionID,
                                    out TpAddress redirectedAddress
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void continueProcessing (
                                    in TpSessionID callLegSessionID
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE);


                              void getMoreDialledDigitsReq (
                                    in TpSessionID callLegSessionID,
                                    in TpInt32 length
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void setChargePlan (
                                    in TpSessionID callLegSessionID,
                                    in TpCallChargePlan callChargePlan
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void setAdviceOfCharge (
                                    in TpSessionID callLegSessionID,
                                    in TpAoCInfo aOCInfo,
                                    in TpDuration tarrifSwitch
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void superviseReq (
                                    in TpSessionID callLegSessionID,
                                    in TpDuration time,
                                    in TpCallSuperviseTreatment treatment
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void deassign (
                                    in TpSessionID callLegSessionID
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);

                        };



                        interface IpAppMultiPartyCall : IpInterface {

                              void getInfoRes (
                                    in TpSessionID callSessionID,
                                    in TpCallInfoReport callInfoReport
                                    );


                              void getInfoErr (
                                    in TpSessionID callSessionID,
                                    in TpCallError errorIndication
                                    );


                              void superviseRes (
                                    in TpSessionID callSessionID,
                                    in TpCallSuperviseReport report,
                                    in TpDuration usedTime
                                    );


                              void superviseErr (
                                    in TpSessionID callSessionID,
                                    in TpCallError errorIndication
                                    );


                              void callFaultDetected (
                                    in TpSessionID callSessionID,
                                    in TpCallFault fault
                                    );


                              void callEnded (
                                    in TpSessionID callSessionID,
                                    in TpCallEndedReport report
                                    );


                              void createAndRouteCallLegErr (
                                    in TpSessionID callSessionID,
                                    in TpCallLegIdentifier callLegReference,
                                    in TpCallError errorIndication
                                    );
                        };


                        interface IpMultiPartyCall : IpService {

                              void getCallLegs (
                                    in TpSessionID callSessionID,
                                    out TpCallLegIdentifierSet callLegList
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void createCallLeg (
                                    in TpSessionID callSessionID,
                                    in IpAppCallLeg appCallLeg,
                                    out TpCallLegIdentifier callLeg
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_ADDRESS,
P_UNSUPPORTED_ADDRESS_PLAN);


                              void createAndRouteCallLegReq (
                                    in TpSessionID callSessionID,
                                    in TpCallEventRequestSet eventsRequested,
                                    in TpAddress targetAddress,
                                    in TpAddress originatingAddress,
                                    in TpCallAppInfoSet appInfo,
                                    in IpAppCallLeg appLegInterface,
                                    out TpCallLegIdentifier callLegReference
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_ADDRESS ,
P_UNSUPPORTED_ADDRESS_PLAN, P_INVALID_NETWORK_STATE, P_INVALID_CRITERIA);


                              void release (
                                    in TpSessionID callSessionID,
                                    in TpCallReleaseCause cause
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE);


                              void deassignCall (
                                    in TpSessionID callSessionID
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void getInfoReq (
                                    in TpSessionID callSessionID,
                                    in TpCallInfoType callInfoRequested
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);
```

```
                              void setChargePlan (
                                    in TpSessionID callSessionID,
                                    in TpCallChargePlan callChargePlan
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void setAdviceOfCharge (
                                    in TpSessionID callSessionID,
                                    in TpAoCInfo aOCInfo,
                                    in TpDuration tariffSwitch
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);


                              void superviseReq (
                                    in TpSessionID callSessionID,
                                    in TpDuration time,
                                    in TpCallSuperviseTreatment treatment
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_SESSION_ID);

                        };



                        interface IpAppMultiPartyCallControlManager :
IpInterface {

                              void reportNotification (
                                    in TpMultiPartyCallIdentifier callReference,

                                    in TpCallLegIdentifierSet
callLegReferenceSet,
                                    in TpCallNotificationInfo notificationInfo,

                                    in TpAssignmentID assignmentID,
                                    out TpAppMultiPartyCallBack appCallBack
                                    );


                              void callAborted (
                                    in TpSessionID callReference
                                    );


                              void managerInterrupted ();


                              void managerResumed ();


                              void callOverloadEncountered (
                                    in TpAssignmentID assignmentID
                                    );
                              void callOverloadCeased (
                                    in TpAssignmentID assignmentID
                                    );

                        };



                        interface IpMultiPartyCallControlManager : IpService {

                              void createCall (
                                    in IpAppMultiPartyCall appCall,
                                    out TpMultiPartyCallIdentifier callReference

                                    )
                                    raises (TpCommonExceptions);


                              void createNotification (
                                    in IpAppMultiPartyCallControlManager
appCallControlManager,
                                    in TpCallNotificationRequest
notificationRequest,
                                    out TpAssignmentID assignmentID
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_CRITERIA, P_INVALID_INTERFACE_TYPE, P_INVALID_EVENT_TYPE);


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


                              void changeNotification (
                                    in TpAssignmentID assignmentID,
                                    in TpCallNotificationRequest
notificationRequest
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_ASSIGNMENT_ID, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE);


                              void getNotification (
                                    out TpNotificationRequestedSet
notificationsRequested
                                    )
                                    raises (TpCommonExceptions);


                              void setCallLoadControl (
                                    in TpDuration duration,
                                    in TpCallLoadControlMechanism mechanism,
                                    in TpCallTreatment treatment,
                                    in TpAddressRange addressRange,
                                    out TpAssignmentID assignmentID
                                    )
                                    raises (TpCommonExceptions,
P_INVALID_ADDRESS, P_UNSUPPORTED_ADDRESS_PLAN);
```

```
                        };

                  };
            };
      };
};
#endif
```

```
//Source file: osa.idl
//Date:  12 June 2001


#ifndef __OSA_DEFINED
#define __OSA_DEFINED



module org {

      module csapi {



            enum TpAddressError {
                  P_ADDRESS_INVALID_UNDEFINED,
                  P_ADDRESS_INVALID_MISSING,
                  P_ADDRESS_INVALID_MISSING_ELEMENT,
                  P_ADDRESS_INVALID_OUT_OF_RANGE,
                  P_ADDRESS_INVALID_INCOMPLETE,
                  P_ADDRESS_INVALID_CANNOT_DECODE
            };



            enum TpAddressPlan {
                  P_ADDRESS_PLAN_NOT_PRESENT,
                  P_ADDRESS_PLAN_UNDEFINED,
                  P_ADDRESS_PLAN_IP,
                  P_ADDRESS_PLAN_MULTICAST,
                  P_ADDRESS_PLAN_UNICAST,
                  P_ADDRESS_PLAN_E164,
                  P_ADDRESS_PLAN_AESA,
                  P_ADDRESS_PLAN_URL,
                  P_ADDRESS_PLAN_NSAP,
                  P_ADDRESS_PLAN_SMTP,
                  P_ADDRESS_PLAN_MSMAIL,
                  P_ADDRESS_PLAN_X400
            };



            enum TpAddressPresentation {
                  P_ADDRESS_PRESENTATION_UNDEFINED,
                  P_ADDRESS_PRESENTATION_ALLOWED,
                  P_ADDRESS_PRESENTATION_RESTRICTED,
                  P_ADDRESS_PRESENTATION_ADDRESS_NOT_AVAILABLE
            };



            enum TpAddressScreening {
                  P_ADDRESS_SCREENING_UNDEFINED,
                  P_ADDRESS_SCREENING_USER_VERIFIED_PASSED,
                  P_ADDRESS_SCREENING_USER_NOT_VERIFIED,
                  P_ADDRESS_SCREENING_USER_VERIFIED_FAILED,
                  P_ADDRESS_SCREENING_NETWORK
            };
```

```
typedef boolean TpBoolean;


typedef float TpFloat;


typedef long TpInt32;


const TpInt32 P_METHOD_NOT_SUPPORTED = 22;

const TpInt32 P_NO_CALLBACK_ADDRESS_SET = 17;

const TpInt32 P_RESOURCES_UNAVAILABLE = 13;

const TpInt32 P_TASK_CANCELLED = 15;

const TpInt32 P_TASK_REFUSED = 14;

typedef TpInt32 TpAssignmentID;


typedef TpInt32 TpDuration;

exception TpGeneralException {
      TpInt32 exceptionType;
};


typedef string TpLongString;


typedef TpInt32 TpSessionID;

typedef sequence <TpSessionID> TpSessionIDSet;


typedef string TpString;


exception P_INVALID_ASSIGNMENT_ID {
      TpString extraInformation;
};


exception P_INVALID_TIME_AND_DATE_FORMAT {
      TpString extraInformation;
};



exception P_INVALID_EVENT_TYPE {
      TpString extraInformation;
};



exception P_INVALID_INTERFACE_NAME {
      TpString extraInformation;
};



exception P_INVALID_INTERFACE_TYPE {
      TpString extraInformation;
};



exception P_UNKNOWN_SUBSCRIBER {
      TpString extraInformation;
};



exception P_INFORMATION_NOT_AVAILABLE {
      TpString extraInformation;
};



struct TpAddress {

      TpAddressPlan Plan;

      TpString AddrString;
      TpString Name;

      TpAddressPresentation Presentation;

      TpAddressScreening Screening;

      TpString SubAddressString;
};


typedef TpAddress TpAddressRange;



typedef sequence <TpAddress> TpAddressSet;



typedef TpString TpPrice;
typedef TpString TpDate;


typedef TpString TpDateAndTime;


typedef TpString TpTime;


typedef TpString TpURL;


typedef TpString TpLanguage;


enum TpCallAoCOrderCategory {
      P_CHARGE_ADVICE_INFO,
      P_CHARGE_PER_TIME,
      P_CHARGE_NETWORK
};


struct TpCAIElements {
      TpInt32 InitialSecsPerTimeInterval;
      TpInt32 SecondsPerTimeInterval;
      TpInt32 SegmentsPerDataInterval;
      TpInt32 ScalingFactor;
      TpInt32 UnitIncrement;
      TpInt32 UnitsPerDataInterval;
      TpInt32 UnitsPerInterval;


};


struct TpChargeAdviceInfo {

      TpCAIElements CurrentCAI;

      TpCAIElements NextCAI;


};


struct TpChargePerTime {
      TpInt32 NextChargePerMinute;
      TpInt32 InitialCharge;
      TpInt32 CurrentChargePerMinute;

};
```

```
            union TpAoCOrder switch(TpCallAoCOrderCategory) {
                  case P_CHARGE_ADVICE_INFO: TpChargeAdviceInfo
ChargeAdviceInfo;
                  case P_CHARGE_PER_TIME: TpChargePerTime ChargePerTime;
                  case P_CHARGE_NETWORK: TpString NetworkCharge;
            };



            struct TpAoCInfo {

                  TpAoCOrder ChargeOrder;
                  TpString Currency;


            };



            exception P_INVALID_NETWORK_STATE {
                  TpString extraInformation;
            };



            exception P_INVALID_CRITERIA {
                  TpString extraInformation;
            };



            const TpInt32 P_INVALID_STATE = 744;
            exception P_INVALID_SESSION_ID {
                  TpString extraInformation;
            };

            exception P_SET_LENGTH_EXCEEDED {
                  TpString extraInformation;
            };

            exception TpCommonExceptions {
                  TpInt32 exceptionType;
                  TpString extraInformation;
            };

            exception P_INVALID_CURRENCY {
                  TpString extraInformation;
            };

            exception P_INVALID_AMOUNT {
                  TpString extraInformation;
            };



            struct TpTimeInterval {
                  TpDateAndTime StartTime;
                  TpDateAndTime StopTime;
            };
            exception P_APPLICATION_NOT_ACTIVATED {
                  TpString extraInformation;
            };

            exception P_INVALID_ADDRESS {
                  TpString extraInformation;
            };

            typedef octet TpOctet;

            typedef sequence <TpOctet> TpOctetSet;

            exception P_UNSUPPORTED_ADDRESS_PLAN {
                  TpString extraInformation;
            };


            interface IpInterface {
            };



            interface IpService : IpInterface {

                  void setCallback (
                        in IpInterface appInterface
                        )
                        raises (TpCommonExceptions);


                  void setCallbackWithSessionID (
                        in IpInterface appInterface,
                        in TpSessionID sessionID
                        )
                        raises (TpCommonExceptions, P_INVALID_SESSION_ID);

            };



      };

};

#endif
```