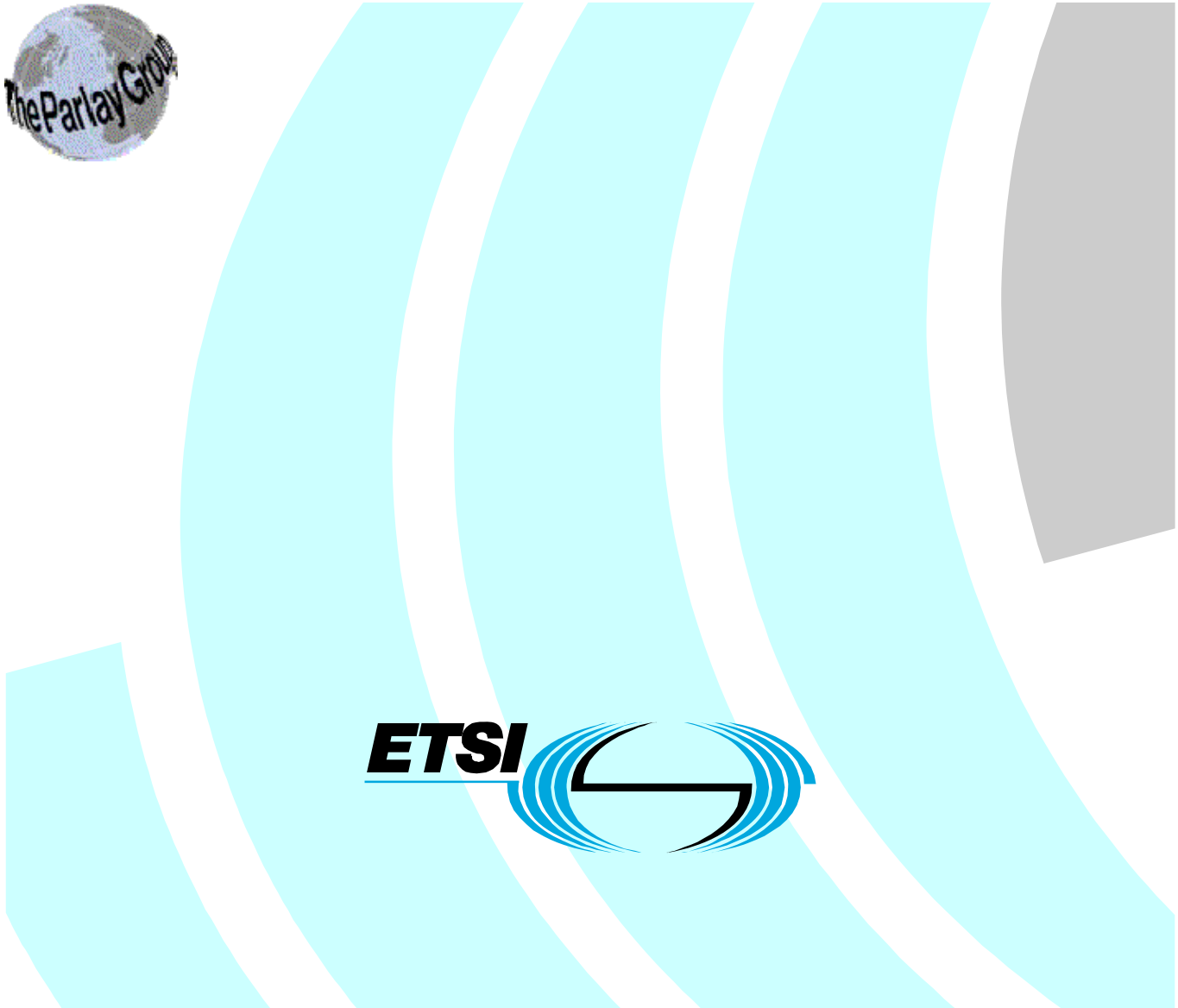
Final draft ETSI ES 204 915-4-2 V1.1.1 (2008-02)

ETSI Standard

Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control; Sub-part 2: Generic Call Control SCF (Parlay 6)



Reference
DES/TISPAN-01032-4-02-OSA

Keywords
API, IDL, OSA, UML

ETSI

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

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

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

Contents

Intellectual Property Rights

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

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

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 4, sub-part 2 of a multi-part deliverable covering Open Service Access (OSA); Application Programming Interface (API), as identified below. The API specification (ES 204 915) is structured in the following parts:

Part 1: "Overview";

Part 2: "Common Data Definitions";

Part 3: "Framework";

Part 4: "Call Control";

Sub-part 1: "Call Control Common Definitions";

Sub-part 2: "Generic Call Control SCF";

Sub-part 3: "Multi-Party Call Control SCF";

Sub-part 4: "Multi-Media Call Control SCF";

Sub-part 5: "Conference 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";

Part 10: "Connectivity Manager SCF";

Part 11: "Account Management SCF";

Part 12: "Charging SCF";

Part 13: "Policy Management SCF";

Part 14: "Presence and Availability Management SCF";

Part 15: "Multi-Media Messaging SCF"

Part 16: "Service Broker SCF".

The present document has been defined jointly between ETSI, The Parlay Group (http://www.parlay.org) and the 3GPP, in co-operation with a number of JAIN™ Community (http://www.java.sun.com/products/jain) member companies.

The present document forms part of the Parlay 6.0 set of specifications.

The present document is equivalent to 3GPP TS 29.198-4-2 V7.0.0 (Release 7).

1 Scope

The present document is part 4, sub-part 2 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 present document specifies the Generic Call Control Service Capability Feature (SCF) aspects of the interface. All aspects of the Generic 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.
- WSDL Description of the interfaces.
- Reference to the Java™ API 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).

2 References

The references listed in clause 2 of ES 204 915-1 contain provisions which, through reference in this text, constitute provisions of the present document.

ETSI ES 204 915-1: "Open Service Access (OSA); Application Programming Interface (API); Part 1: Overview (Parlay 6)".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ES 204 915-1 apply.

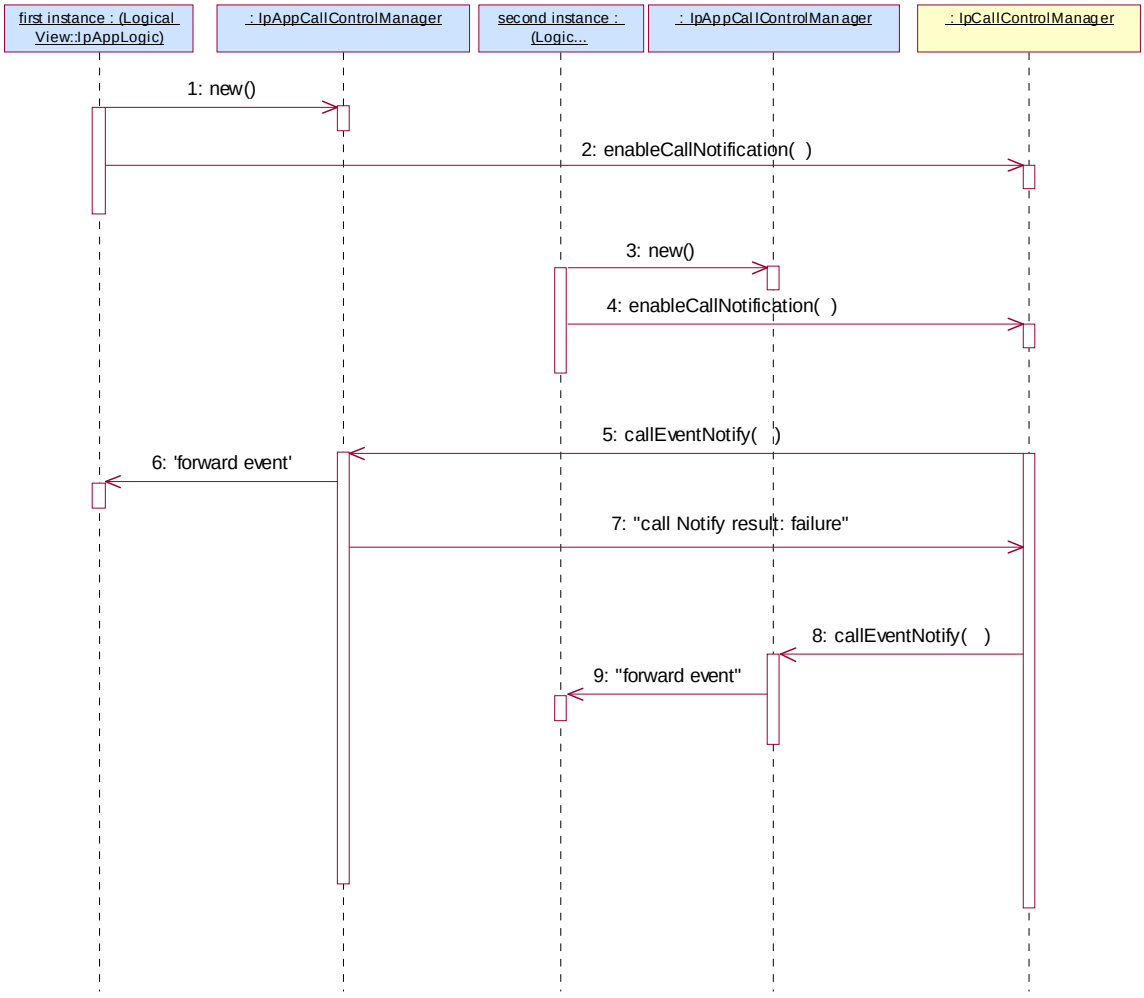
3.2 Abbreviations

For the purposes of the present document, the abbreviations given in ES 204 915-1 apply.

4 Generic Call Control Service Sequence Diagrams

4.1 Additional Callbacks

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



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

2: The enableCallNotification 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 enableCallNotification 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.

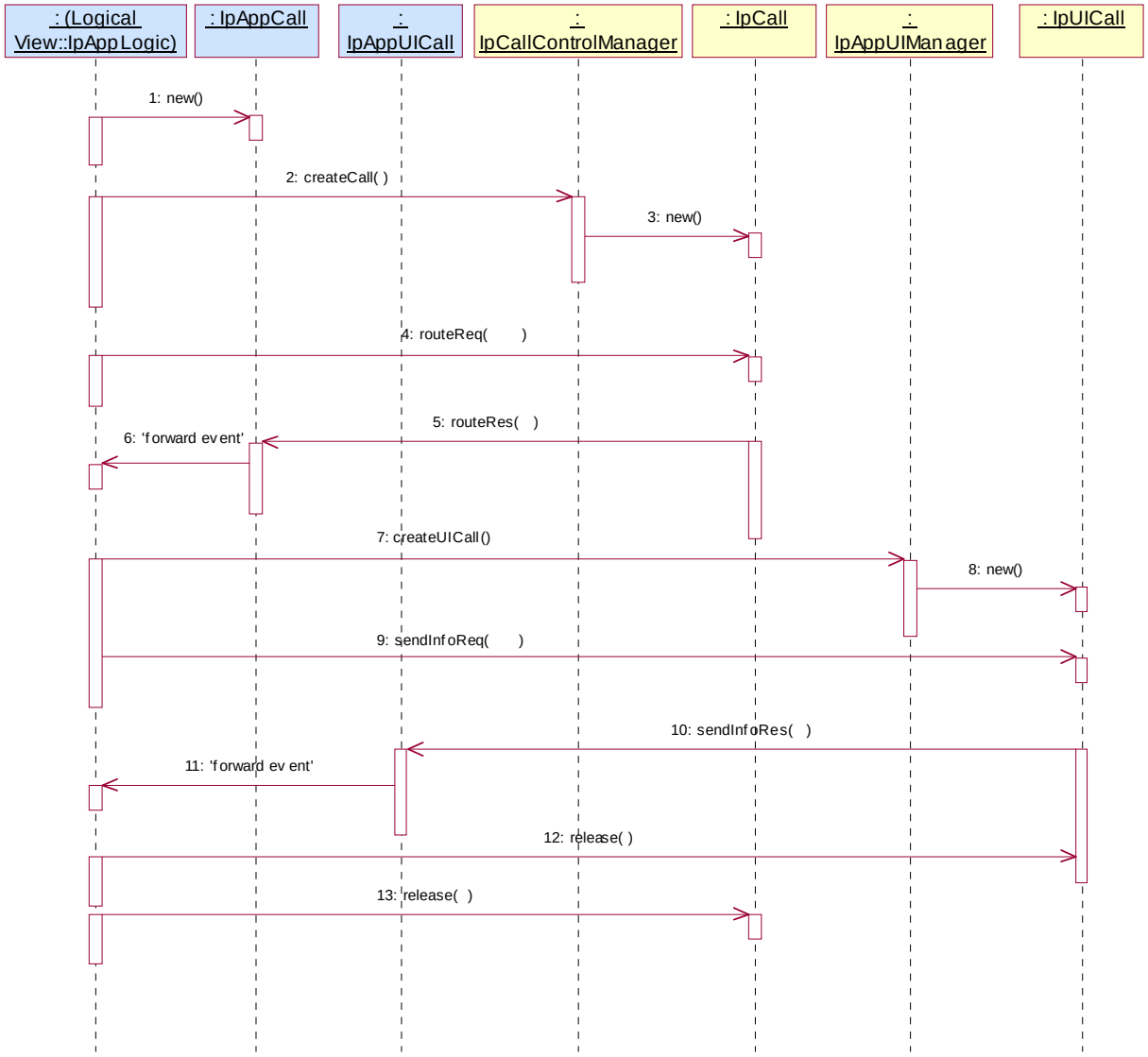
7: When the first instance of the application is overloaded or unavailable this is communicated with an exception to the call control manager.

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.

4.2 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.

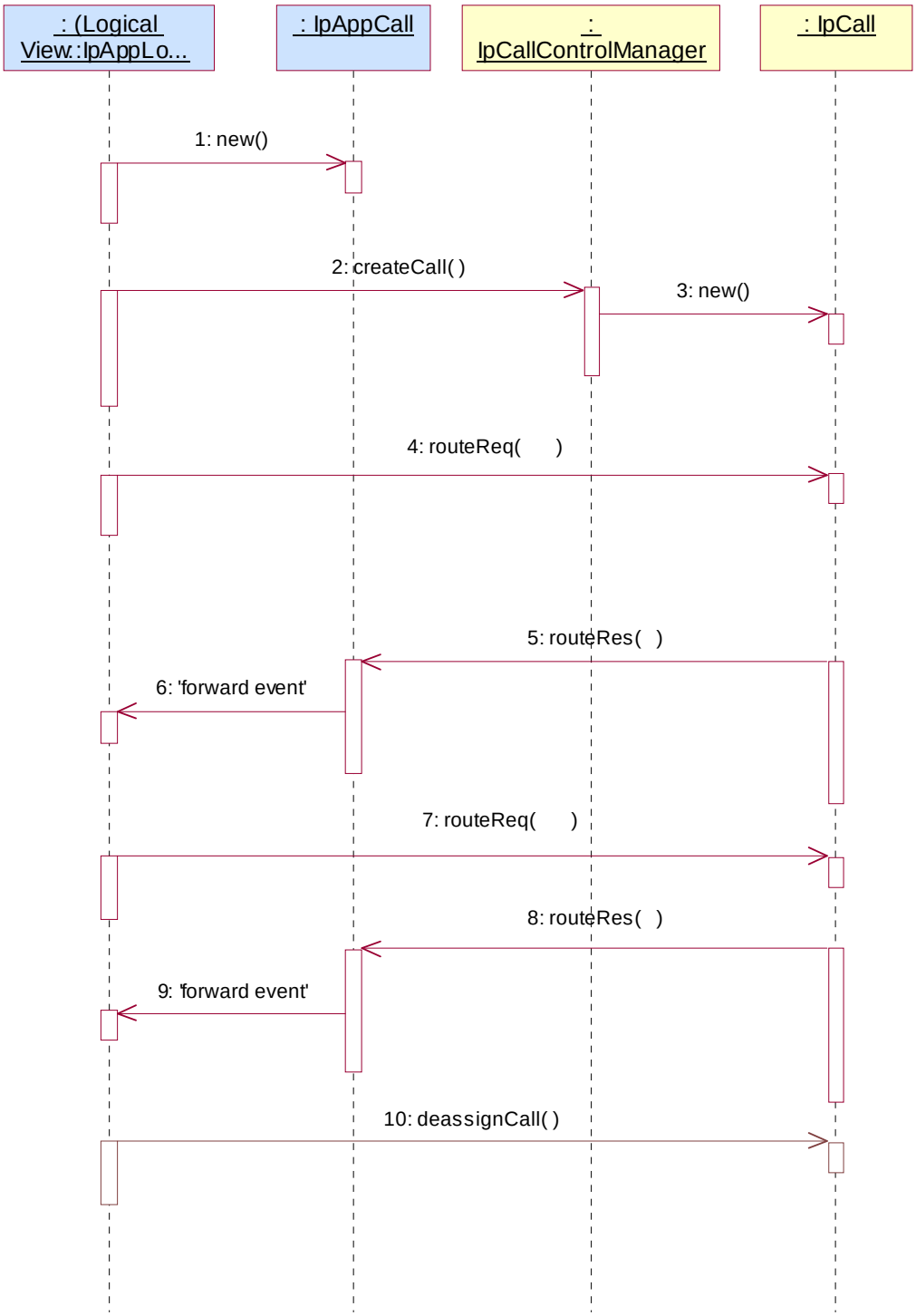
11: The event is forwarded to the application logic.

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.

4.3 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.



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.

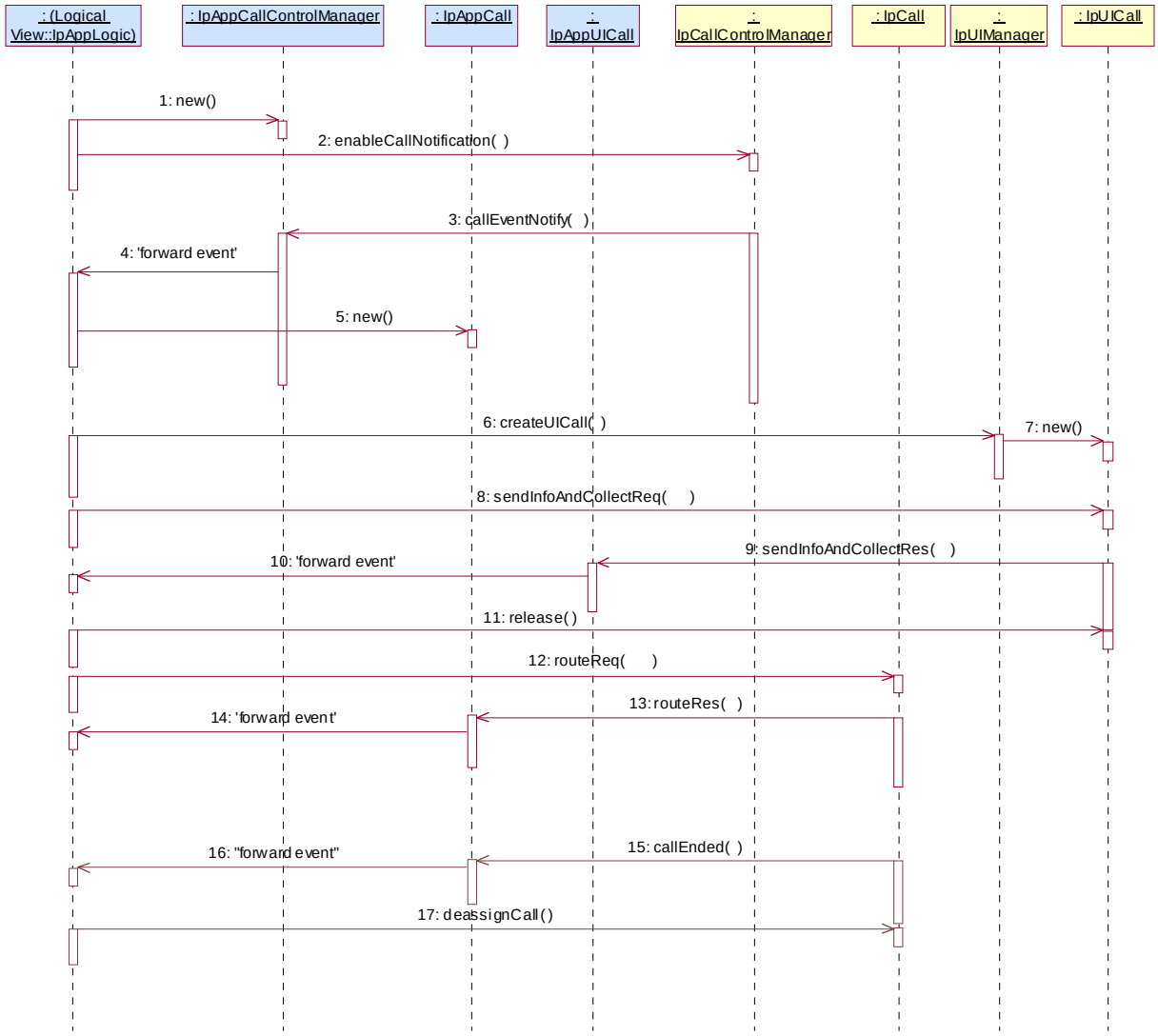
8: This message indicates that the B-party answered the call. The call now has two parties and a speech connection is automatically established between them.

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.

4.4 Call Barring 1

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the call control service. 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.

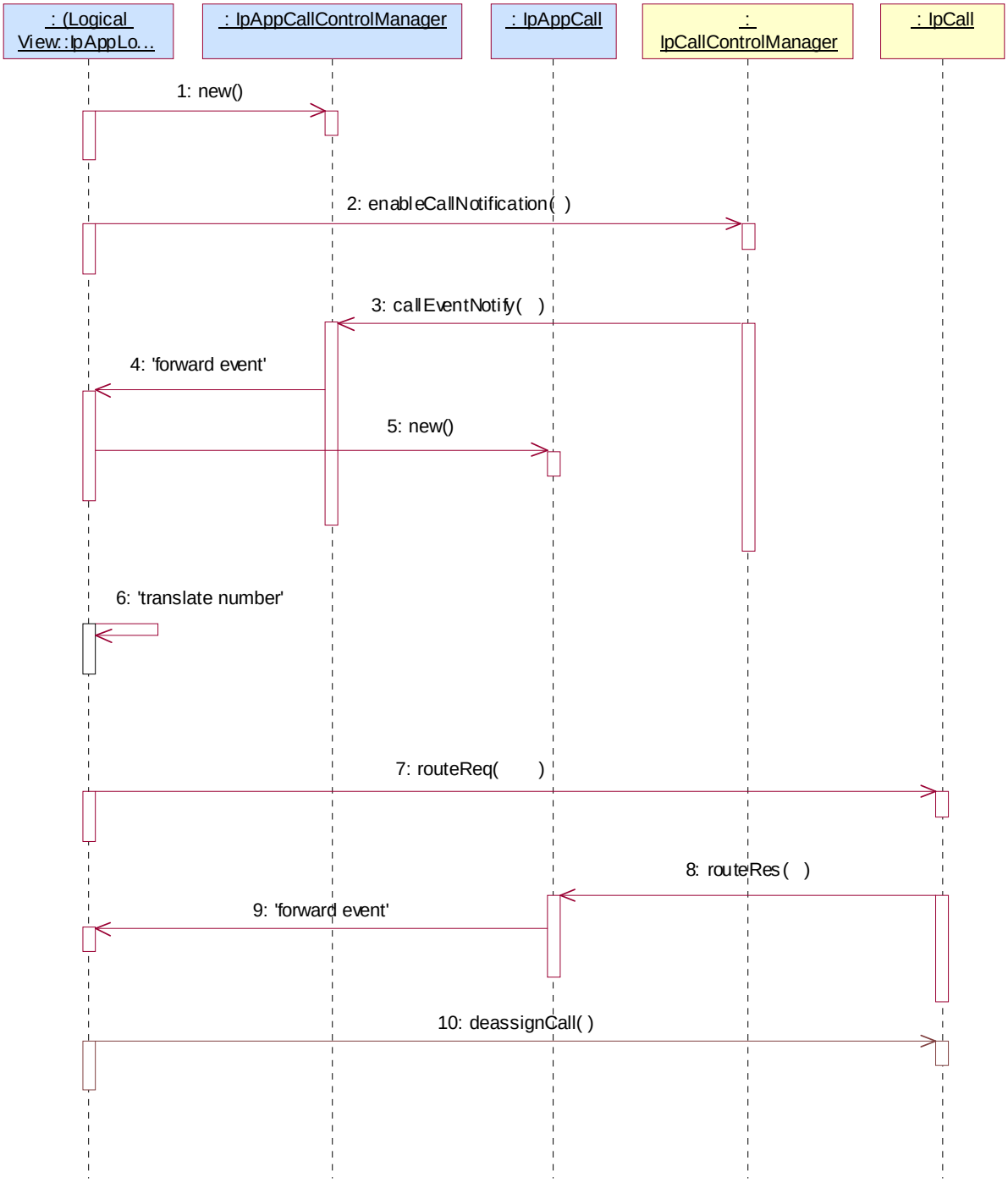
15: When the call is terminated in the network, the application will receive a notification. This notification will always be received when the call is terminated by the network in a normal way, the application does not have to request this event explicitly.

16: The event is forwarded to the application.

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

4.5 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 call control service.



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.

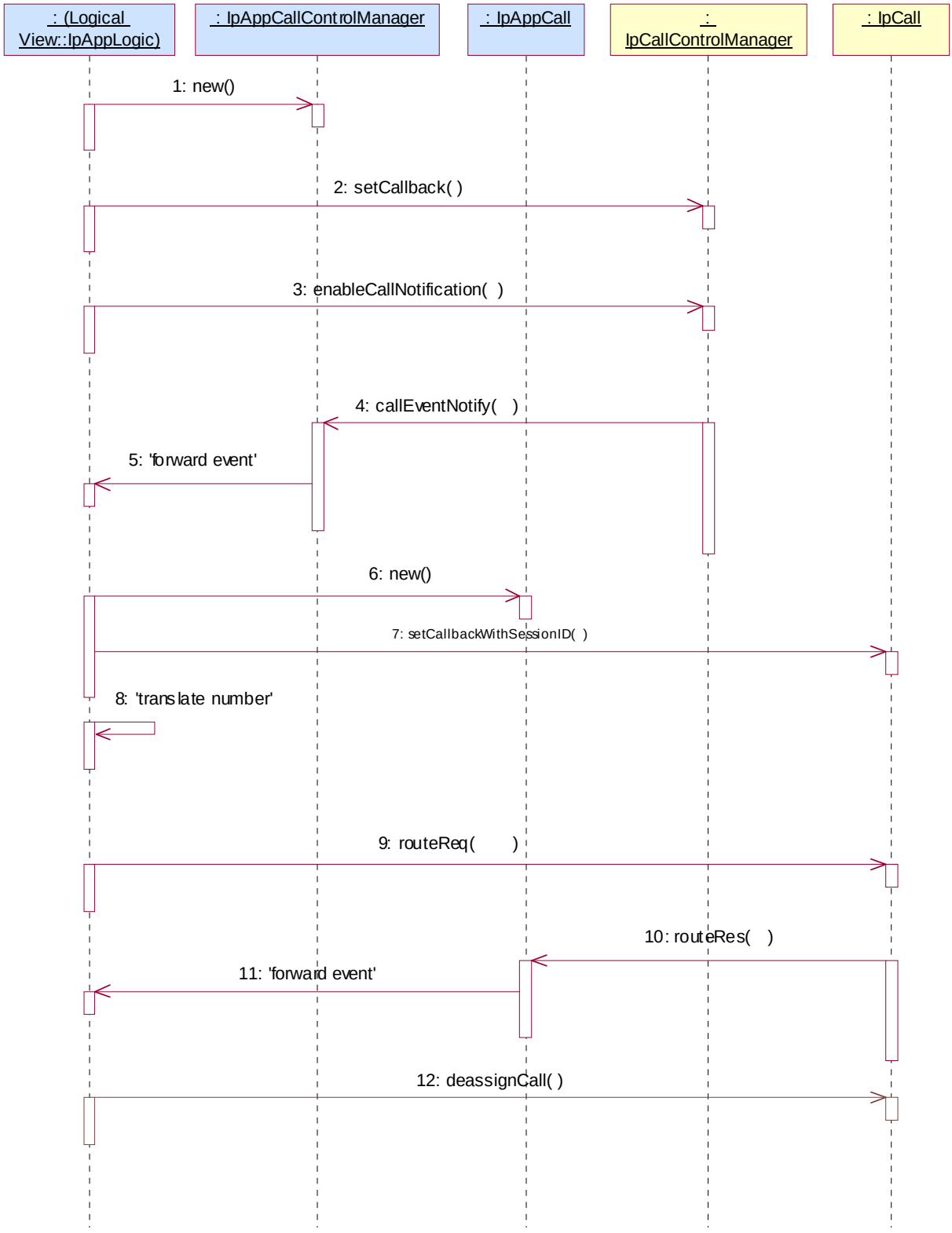
9: This message is used to forward the previous message to the IpAppLogic.

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.

4.6 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 call control service.

For illustration, in this sequence the callback references are set explicitly. 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 sets the reference of the IpAppCallControlManager object in the CallControlManager. The CallControlManager reports the callEventNotify to referenced object only for enableCallNotifications that do not have an explicit IpAppCallControlManager reference specified in the enableCallNotification.

3: 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 3, 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.

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.

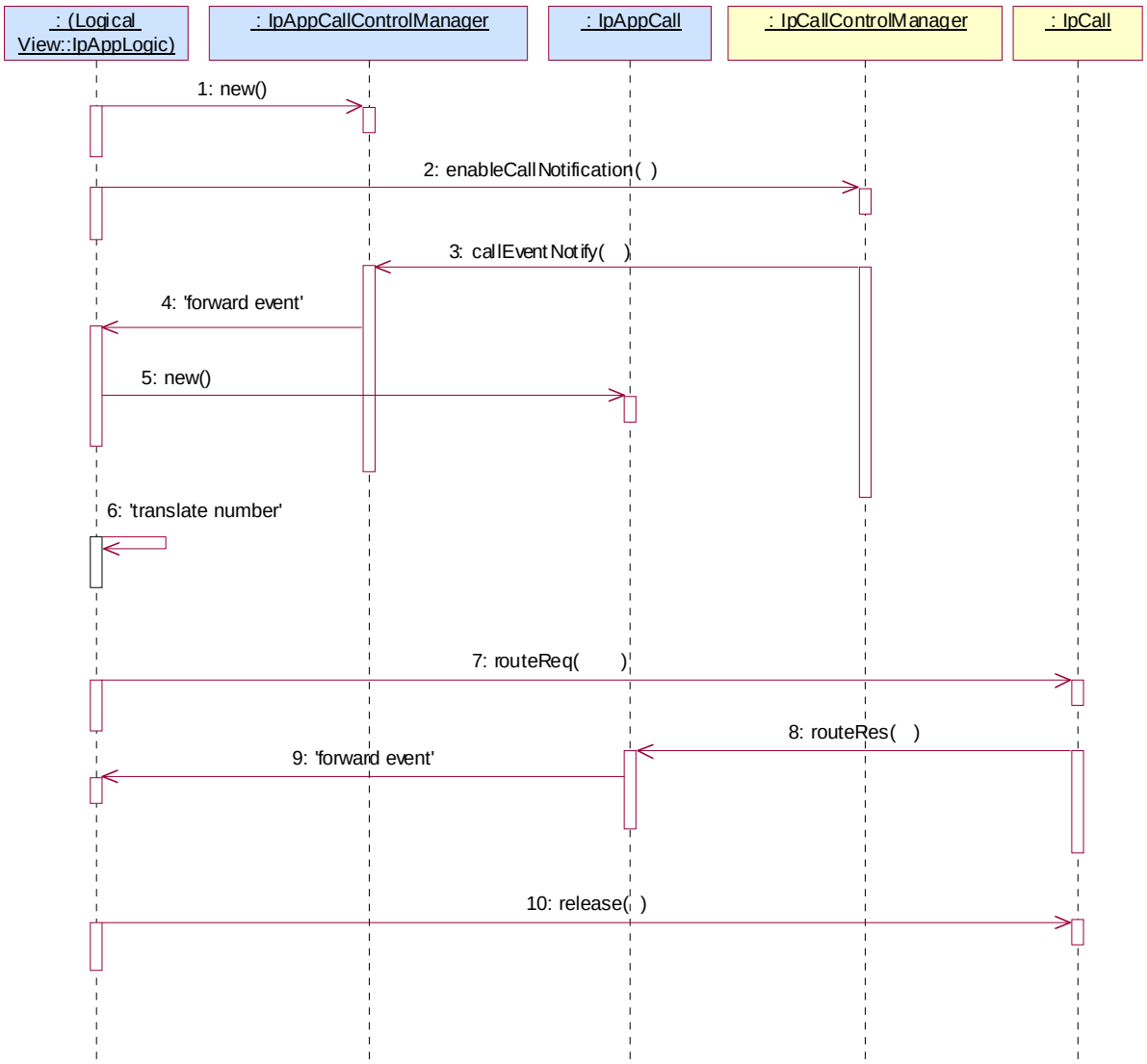
10: This message passes the result of the call being answered to its callback object.

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.

4.7 Number Translation 2

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

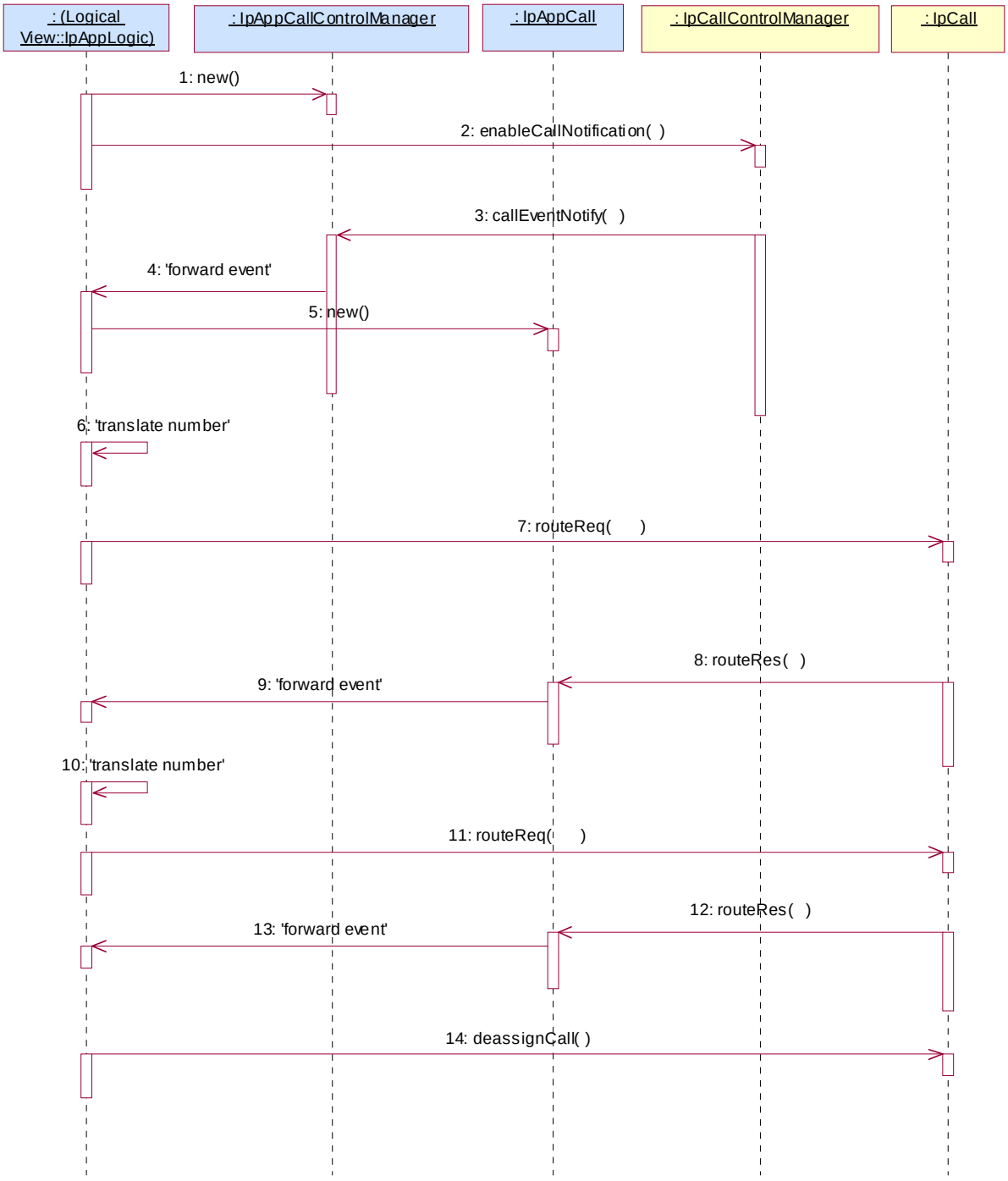
8: Assuming the called party is busy or does not answer, the object implementing the IpCall interface sends a callback in this message, indicating the unavailability of the called party.

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

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

4.8 Number Translation 3

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

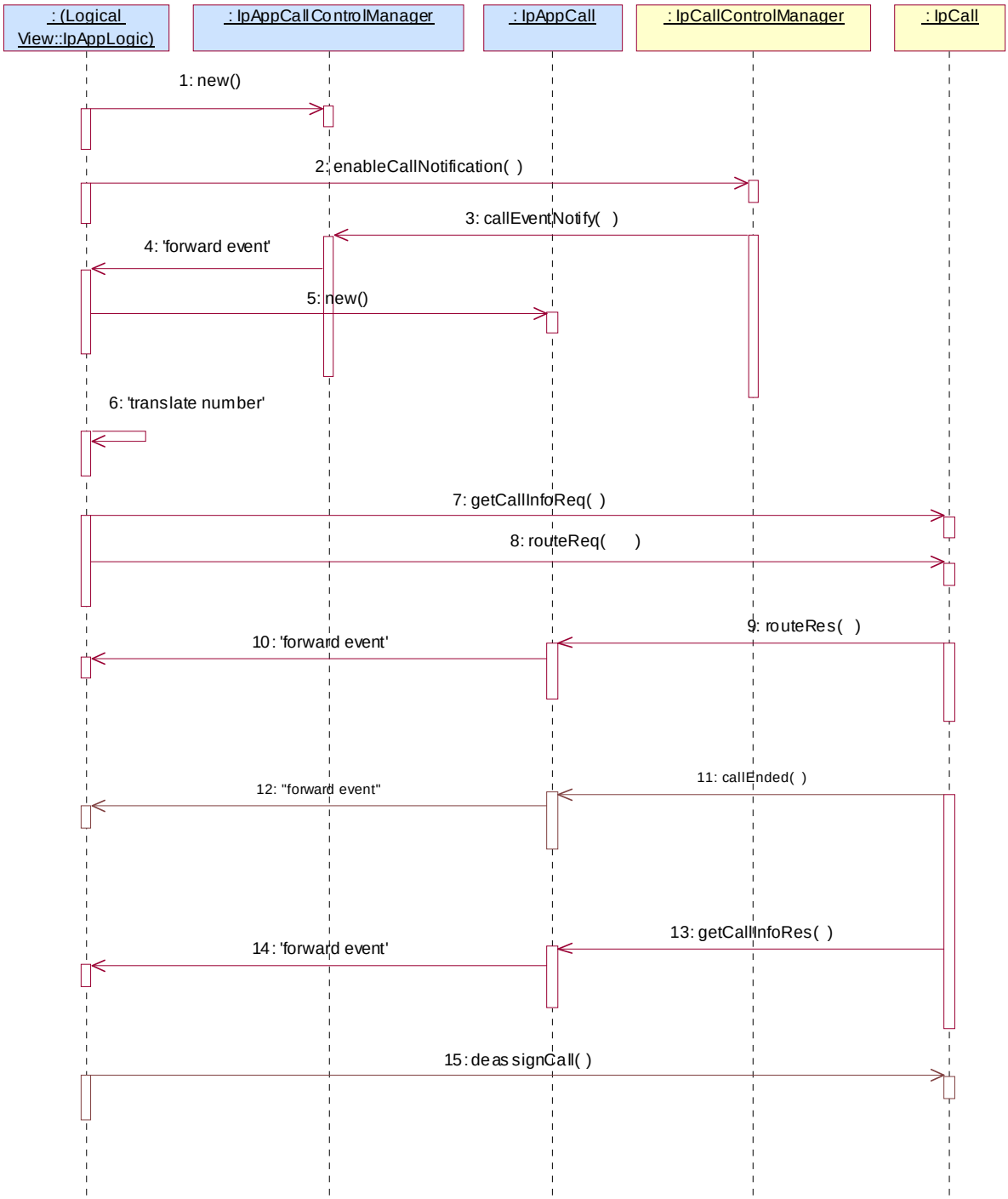
12: This message passes the result of the call being answered to its callback object.

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.

4.9 Number Translation 4

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

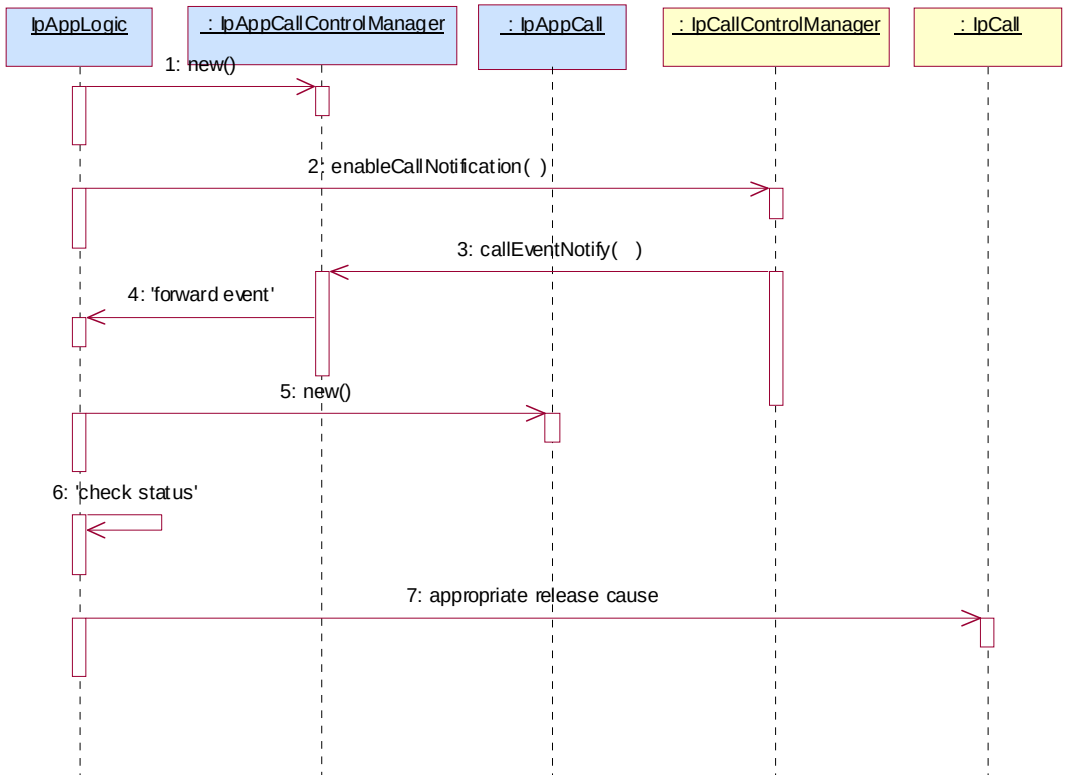
13: The application now waits for the call information to be sent. Now that the call has completed, the object implementing the IpCall interface passes the call information to its callback object.

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.

4.10 Number Translation 5

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.

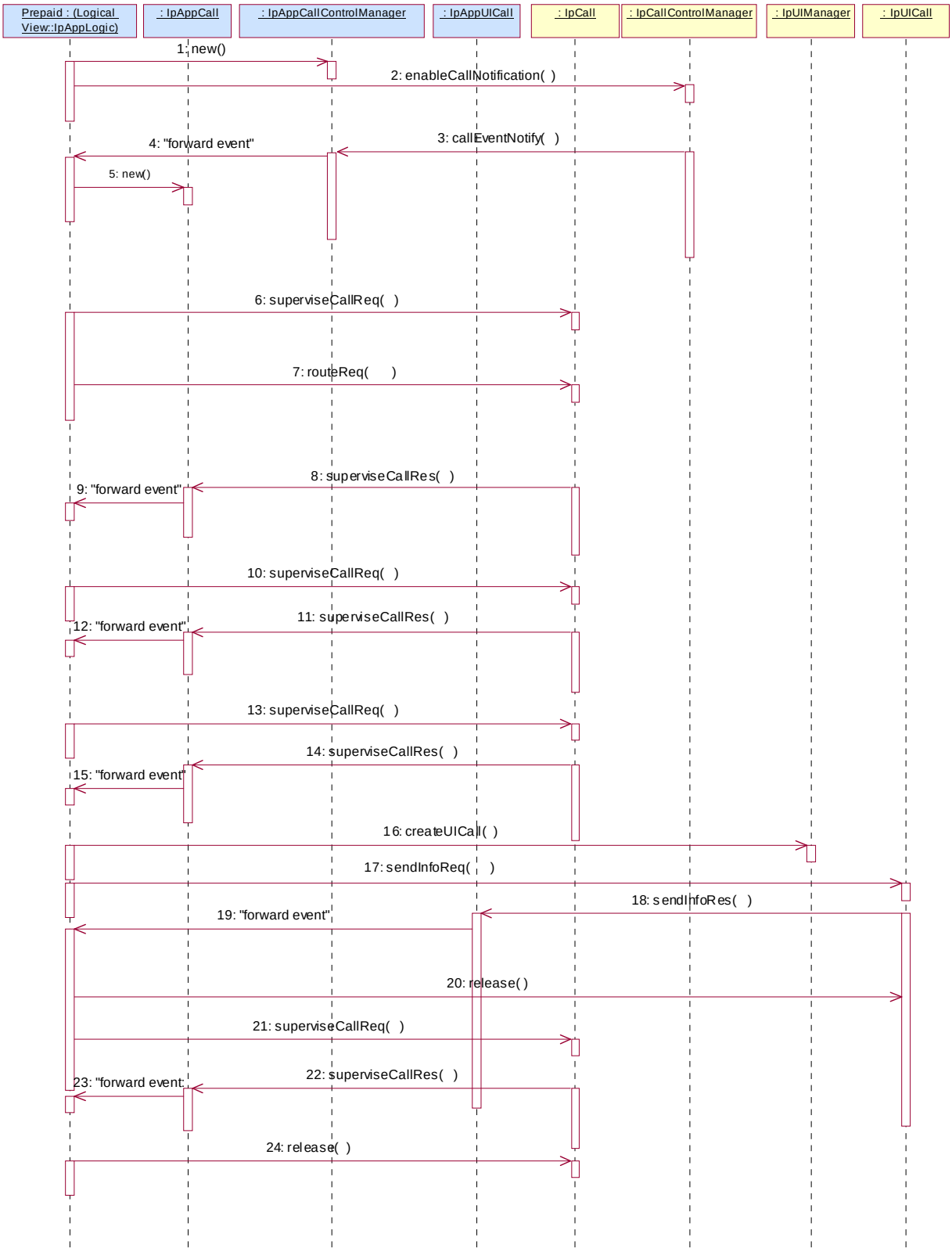
6: This message invokes the status checking function.

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

4.11 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 receive an announcement before his final timeslice.



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 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. When the timer expires it will indicate that the user is almost out of credit.

14: When the user is almost out of credit the application is informed.

15: The message is forwarded to the application.

16: The application decides to play an announcement to the parties in this call. A new UICall object is created and associated with the call.

17: An announcement is played informing the user about the near-expiration of his credit limit.

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.

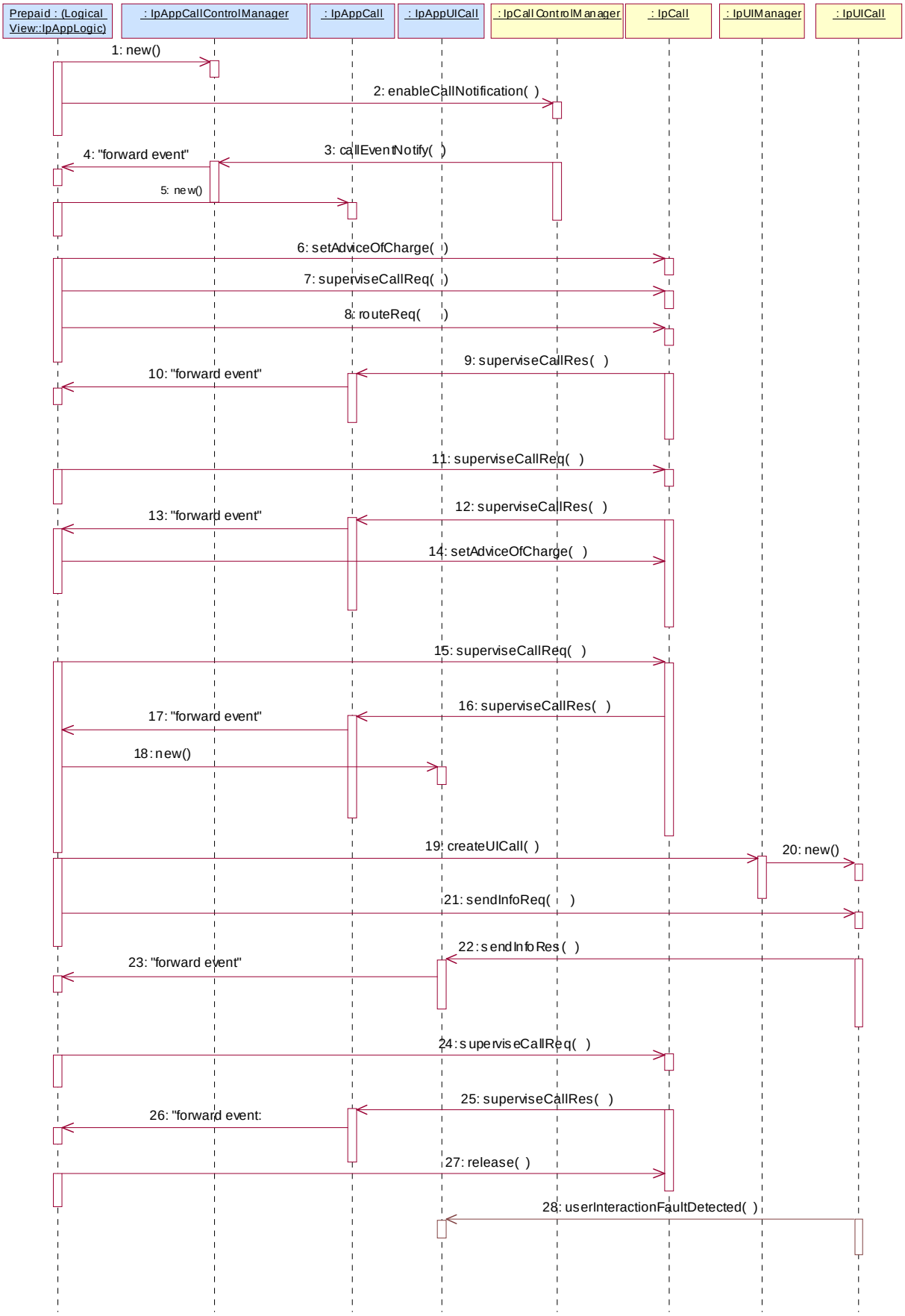
23: The event is forwarded to the logic.

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

4.12 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.



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 tariff 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. When the timer expires it will indicate that the user is almost out of credit.

16: When the user is almost out of credit the application is informed.

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 parties in the call.

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.

5 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 IpAppCallControlManager, by means of calling callback methods.

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

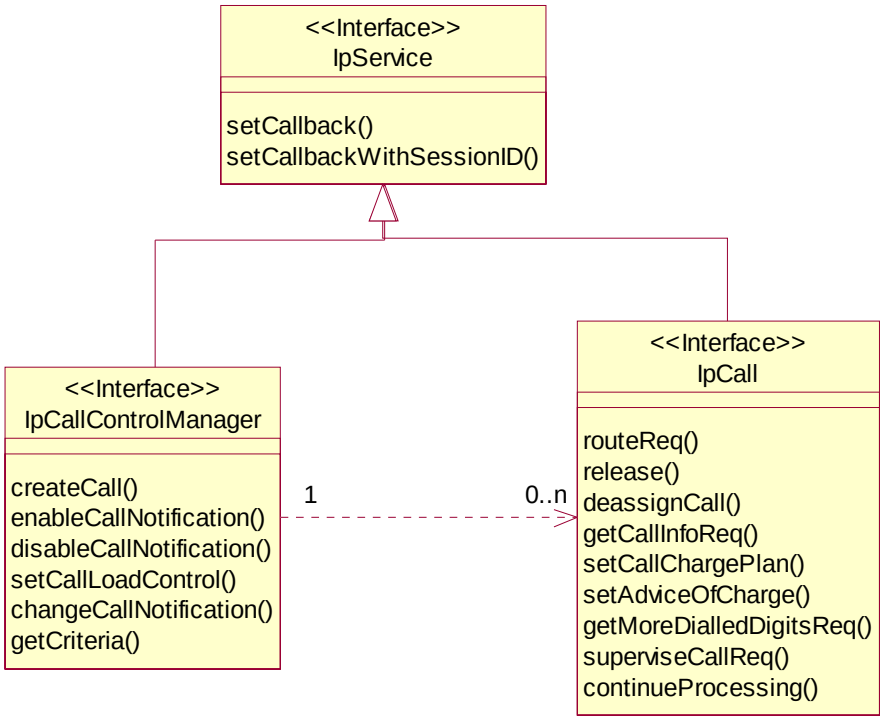


Figure 1: Service Interfaces

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.

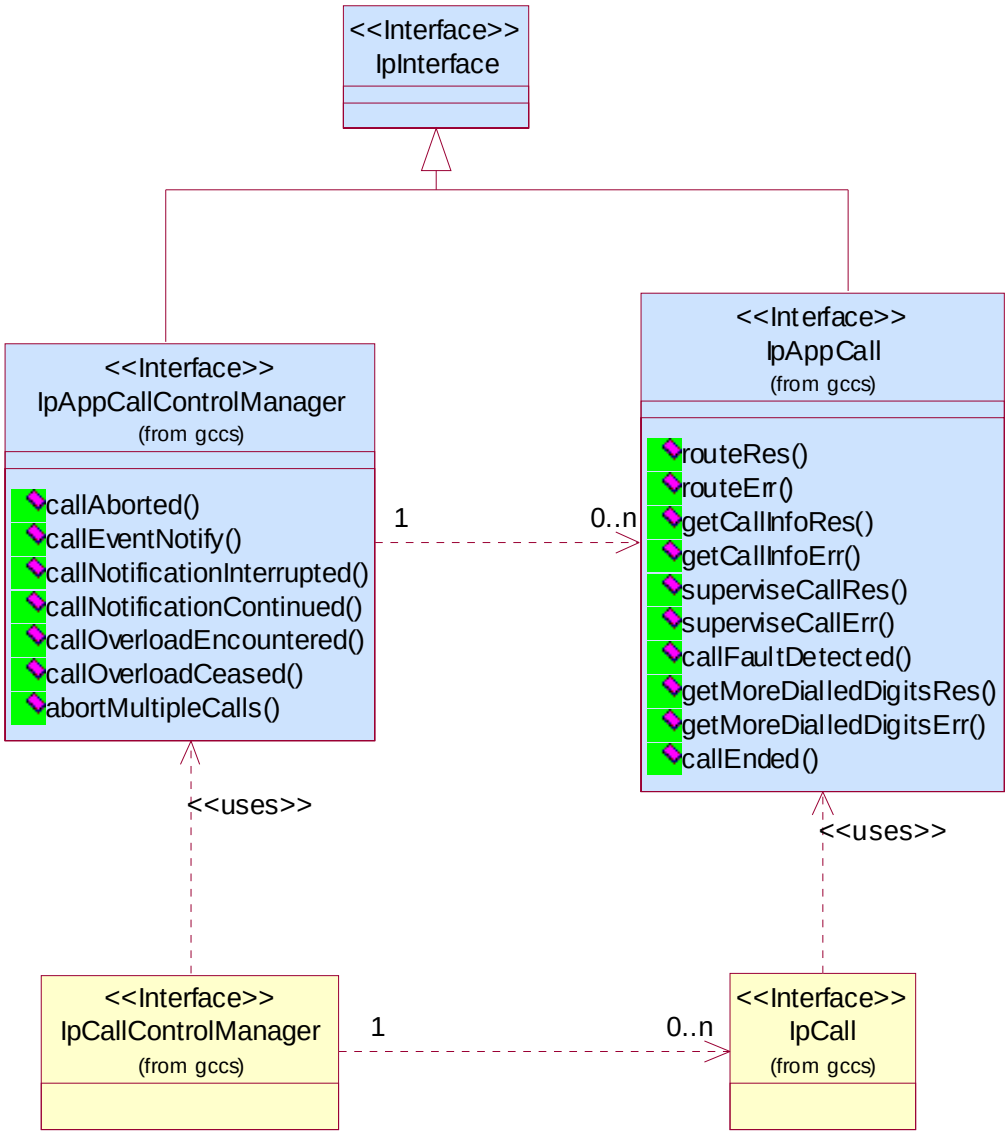


Figure 2: Application Interfaces

6 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, Q.763 ISUP, Q.931 and Q.2931, ATM Forum specification UNI3.1 and RFC 3261 Session Initiation Protocol, or any other call control technology.

For the generic call control service, only a subset of the call model defined in clause 4 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.

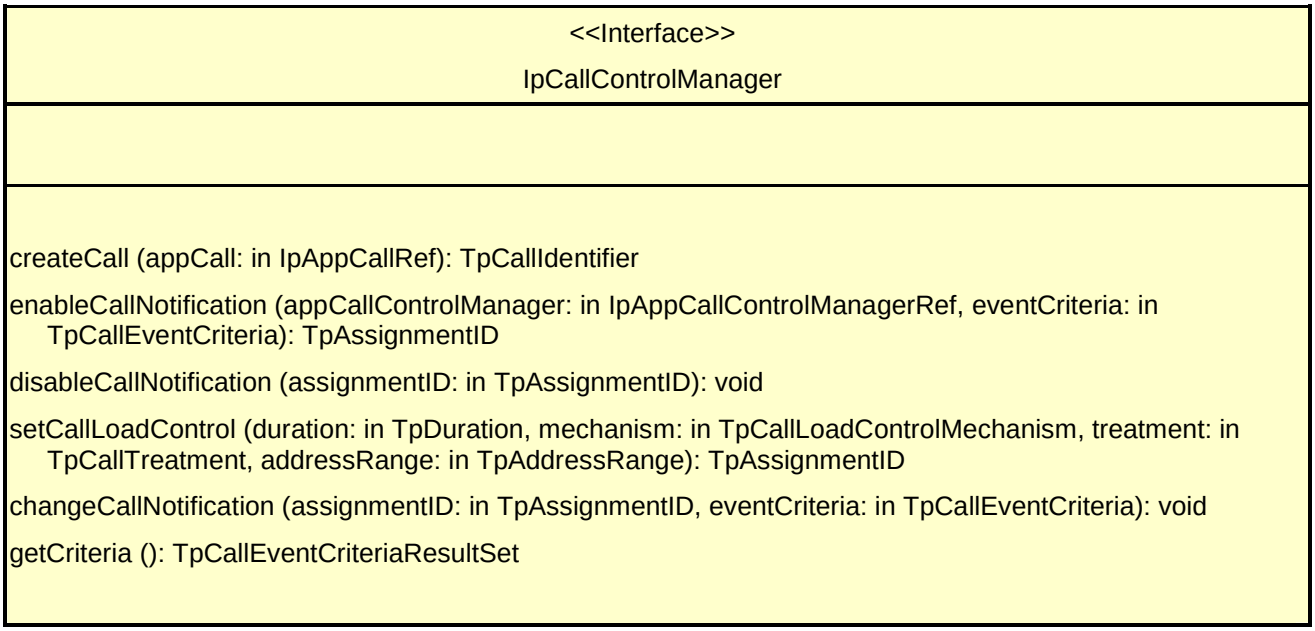
The GCCS is represented by the IpCallControlManager and IpCall interfaces that interface to services provided by the network. Some methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one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

6.1 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.

This interface shall be implemented by a Generic Call Control SCF. As a minimum requirement either the createCall() method shall be implemented, or the enableCallNotification() and disableCallNotification() methods shall be implemented.



6.1.1 Method createCall()

This method is used to create a new call object.

Callback reference:

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 shall invoke setCallback() prior to createCall() if it wishes to ensure this.

Returns callReference: Specifies the interface reference and sessionID of the call created.

Parameters

appCall: in IpAppCallRef

Specifies the application interface for callbacks from the call created.

Returns

TpCallIdentifier

Raises

TpCommonExceptions, P_INVALID_INTERFACE_TYPE

6.1.2 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 a notification is requested by an application with the monitor mode set to notify, then there is no need to check the rest of the criteria for overlapping with any existing request as the notify mode does not allow control on a call to be passed over. Only one application can place an interrupt request if the criteria overlap.

If a notification is requested by an application with an event type that is mutually exclusive compared to existing requested event types, then there is no need to check against the rest of the criteria for overlap. An example could be one application that trigger on "user busy" together with another application that trigger on "answer" - both requests should be allowed as only one can occur on the same call or session.

The overlap criteria have been defined to prevent multiple points of control, leading to possible interaction problems in networks that have no multi service support. Notice that dynamic aspects cannot be taken into account in the overlap criteria check. Therefore where dynamic event arming from an application causes a persistent control relationship it can prevent other applications to be invoked in the case single point of application control applies in the network.

However, the criteria check for overlap may as a network option be overruled by Multi Service networks allowing more services or applications to gain control of the same call or session at the same point in time. Refer to Call Control Common Definitions subpart of this specification (TS 129 198-4-1) for further details on application control over a call or session.

Setting the callback reference:

The callback reference can be registered either in a) enableCallNotification() or b) explicitly with a separate setCallback() method depending on how the application provides its callback reference.

Case a:

From an efficiency point of view the enableCallNotification() with explicit immediate registration (no "Null" value) of callback reference may be the preferred method.

Case b:

The enableCallNotfication() with no call back reference ("Null" value) is used where (e.g. due to distributed application logic) the callback reference is provided previously in a setCallback(). If no callback reference has been provided previously to the service, the exception P_NO_CALLBACK_ADDRESS_SET shall be raised.

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(). See example in clause 4.6.

Set additional callback:

If the same application invokes this method multiple times with exactly the same criteria but with different callback references, then these shall be treated as additional callback references. Each such notification request shall share the same assignmentID. The gateway shall use the most recent callback interface provided by the application using this method. In the event that a callback reference fails or is no longer available, the next most recent callback reference available shall be used. See example in clause 4.1.

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

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 previously 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.

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_INVALID_CRITERIA, P_INVALID_INTERFACE_TYPE, P_INVALID_EVENT_TYPE

6.1.3 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 enableCallNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the exception P_INVALID_ASSIGNMENT_ID will be raised.

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

6.1.4 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.

Returns assignmentID: Specifies the assignmentID assigned by the gateway to this request. This assignmentID can be used to correlate the callOverloadEncountered and callOverloadCeased methods with the request.

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.

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_INVALID_ADDRESS, P_UNSUPPORTED_ADDRESS_PLAN

6.1.5 Method changeCallNotification()

This method is used by the application to change the event criteria introduced with enableCallNotification. Any stored criteria associated with the specified assignmentID 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

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

6.1.6 Method getCriteria()

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

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

Parameters

No Parameters were identified for this method.

Returns

TpCallEventCriteriaResultSet

Raises

TpCommonExceptions

6.2 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): void callEventNotify (callReference: in TpCallIdentifier, eventInfo: in TpCallEventInfo, assignmentID: in TpAssignmentID): IpAppCallRef callNotificationInterrupted (): void callNotificationContinued (): void callOverloadEncountered (assignmentID: in TpAssignmentID): void callOverloadCeased (assignmentID: in TpAssignmentID): void abortMultipleCalls (callReferenceSet: in TpSessionIDSet): void

6.2.1 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.

6.2.2 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_CALL_MONITOR_MODE_INTERRUPT, 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).

Setting the callback reference:

A reference to the application interface has to be passed back to the call interface to which the notification relates. However, the setting of a call back reference is only applicable if the notification is in INTERRUPT mode. When the callEventNotify() method is invoked with a monitor mode of P_CALL_MONITOR_MODE_INTERRUPT, the application writer should ensure that no continue processing e.g. routeReq() is performed until an IpAppCall has been passed to the gateway, either through an explicit setCallbackWithSessionID() invocation on the supplied IpCall, or via the return of the callEventNotify() method.

The callback reference can be registered either in a) callEventNotify() or b) explicitly with a setCallbackWithSessionID() method e.g. depending on how the application provides its call reference.

Case a:

From an efficiency point of view the callEventNotify() with explicit pass of registration may be the preferred method.

Case b:

The callEventNotify() with no callback reference ("Null" value) is used where (e.g. due to distributed application logic) the callback reference is provided previously in a setCallbackWithSessionID(). If no callback reference has been provided previously to the service, the exception P_NO_CALLBACK_ADDRESS_SET shall be raised, and no further application invocations related to the call shall be permitted.

In case the callEventNotify() contains no callback, at the moment the application needs to be informed the gateway will use as callback the callback that has been registered previously by setCallbackWithSessionID(). See example in clause 4.6.

Returns appCall: Specifies a reference to the application interface which implements the callback interface for the new call. If the application has previously explicitly passed a reference to the IpAppCall interface using a setCallbackWithSessionID() invocation, this parameter may be null, or if supplied must be the same as that provided during the setCallbackWithSessionID().

This parameter will be null if the notification is in NOTIFY mode and in case b).

Parameters

callReference: in TpCallIdentifier

Specifies the reference to the call interface to which the notification relates. If the notification is in NOTIFY mode, this parameter shall be ignored by the application client implementation, and consequently the implementation of the SCS entity invoking callEventNotify may populate this parameter as it chooses.

eventInfo: in TpCallEventInfo

Specifies data associated with this event.

assignmentID: in TpAssignmentID

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

Returns

IpAppCallRef

6.2.3 Method callNotificationInterrupted()

This method indicates to the application that all event notifications have been temporarily 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.

6.2.4 Method callNotificationContinued()

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

Parameters

No Parameters were identified for this method.

6.2.5 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 address range for within which the overload has been encountered.

6.2.6 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 address range for within which the overload has been ceased.

6.2.7 Method abortMultipleCalls()

The service may invoke this method on the IpAppCallControlManager interface to indicate that a number of ongoing call sessions have aborted or terminated abnormally. No further communication will be possible between the application and the calls. This may be used for example in the event of service failure and recovery in order to instruct the application that a number of call sessions have failed. The service shall provide a set of call sessionIDs indicating to the application the call sessions that have aborted. In the case that the service invokes this method and provides an empty set of sessionIDs, this shall be used to indicate that all call sessions previously active on the IpCallControlManager interface have been aborted.

Parameters

callReferenceSet: in TpSessionIDSet

Specifies the set of sessionIDs of calls that have aborted or terminated abnormally. The empty set shall be used to indicate that all calls have aborted.

6.3 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.

This interface shall be implemented by a Generic Call Control SCF. As a minimum requirement, the routeReq (), release() and deassignCall() methods shall be implemented.

<<Interface>> IpCall
routeReq (callSessionID: in TpSessionID, responseRequested: in TpCallReportRequestSet, targetAddress: in TpAddress, originatingAddress: in TpAddress, originalDestinationAddress: in TpAddress, redirectingAddress: in TpAddress, appInfo: in TpCallAppInfoSet): TpSessionID release (callSessionID: in TpSessionID, cause: in TpCallReleaseCause): void deassignCall (callSessionID: in TpSessionID): void getCallInfoReq (callSessionID: in TpSessionID, callInfoRequested: in TpCallInfoType): void setCallChargePlan (callSessionID: in TpSessionID, callChargePlan: in TpCallChargePlan): void setAdviceOfCharge (callSessionID: in TpSessionID, aOCInfo: in TpAoCInfo, tariffSwitch: in TpDuration): void getMoreDialledDigitsReq (callSessionID: in TpSessionID, length: in TpInt32): void superviseCallReq (callSessionID: in TpSessionID, time: in TpDuration, treatment: in TpCallSuperviseTreatment): void continueProcessing (callSessionID: in TpSessionID): void

6.3.1 Method routeReq()

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

Note that in case of routeReq() it is recommended to request for 'successful' (e.g. 'answer' event) and 'failure' events at invocation, because those are needed for the application to keep track of the state of the call.

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.

This operation continues processing of the call implicitly.

Returns callLegSessionID: 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.

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).

Returns

TpSessionID

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_ADDRESS, P_UNSUPPORTED_ADDRESS_PLAN, P_INVALID_NETWORK_STATE, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

6.3.2 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.

This operation continues processing of the call implicitly.

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

6.3.3 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.

This operation continues processing of the call implicitly.

Parameters

callSessionID: in TpSessionID

Specifies the call session ID of the call.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.3.4 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.

A report is received when the destination leg or party terminates or when the call ends. 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. 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

TpCommonExceptions, P_INVALID_SESSION_ID

6.3.5 Method setCallChargePlan()

Set an operator specific charge plan for the call.

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

6.3.6 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

6.3.7 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

TpCommonExceptions, P_INVALID_SESSION_ID

6.3.8 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

TpCommonExceptions, P_INVALID_SESSION_ID

6.3.9 Method continueProcessing()

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

In case the operation is invoked and call processing is not interrupted the exception P_INVALID_NETWORK_STATE will be raised.

Parameters

callSessionID: in TpSessionID

Specifies the call session ID of the call.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

6.4 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): void routeErr (callSessionID: in TpSessionID, errorIndication: in TpCallError, callLegSessionID: in TpSessionID): void getCallInfoRes (callSessionID: in TpSessionID, callInfoReport: in TpCallInfoReport): void getCallInfoErr (callSessionID: in TpSessionID, errorIndication: in TpCallError): void superviseCallRes (callSessionID: in TpSessionID, report: in TpCallSuperviseReport, usedTime: in TpDuration): void superviseCallErr (callSessionID: in TpSessionID, errorIndication: in TpCallError): void callFaultDetected (callSessionID: in TpSessionID, fault: in TpCallFault): void getMoreDialledDigitsRes (callSessionID: in TpSessionID, digits: in TpString): void getMoreDialledDigitsErr (callSessionID: in TpSessionID, errorIndication: in TpCallError): void callEnded (callSessionID: in TpSessionID, report: in TpCallEndedReport): void

6.4.1 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_CALL_MONITOR_MODE_INTERRUPT, 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 sessionID returned at the routeReq() and can be used to correlate the response with the request.

6.4.2 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.

6.4.3 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.

6.4.4 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.

6.4.5 Method superviseCallRes()

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

It is also called when the connection is terminated before the supervision event occurs.

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).

6.4.6 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.

6.4.7 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.

6.4.8 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.

6.4.9 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.

6.4.10 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.

7 Generic Call Control Service State Transition Diagrams

7.1 State Transition Diagrams for IpCallControlManager

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

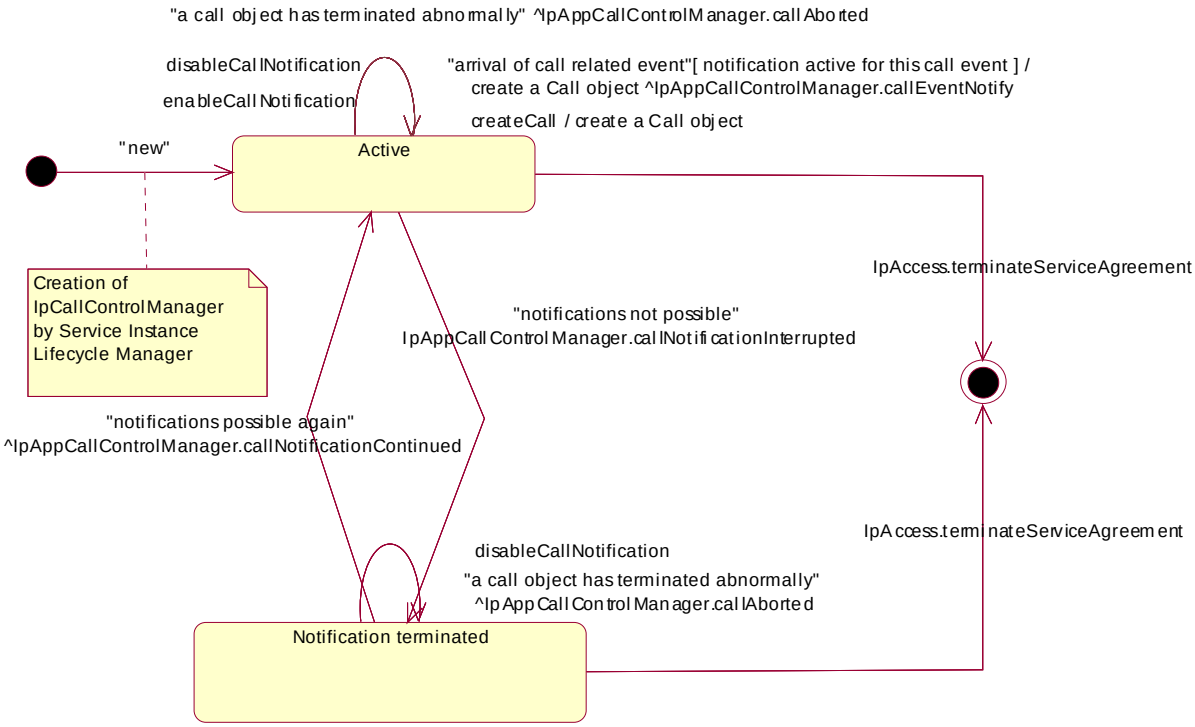


Figure 3: Application view on the Call Control Manager

7.1.1 Active State

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

7.1.2 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.

7.2 State Transition Diagrams for IpCall

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

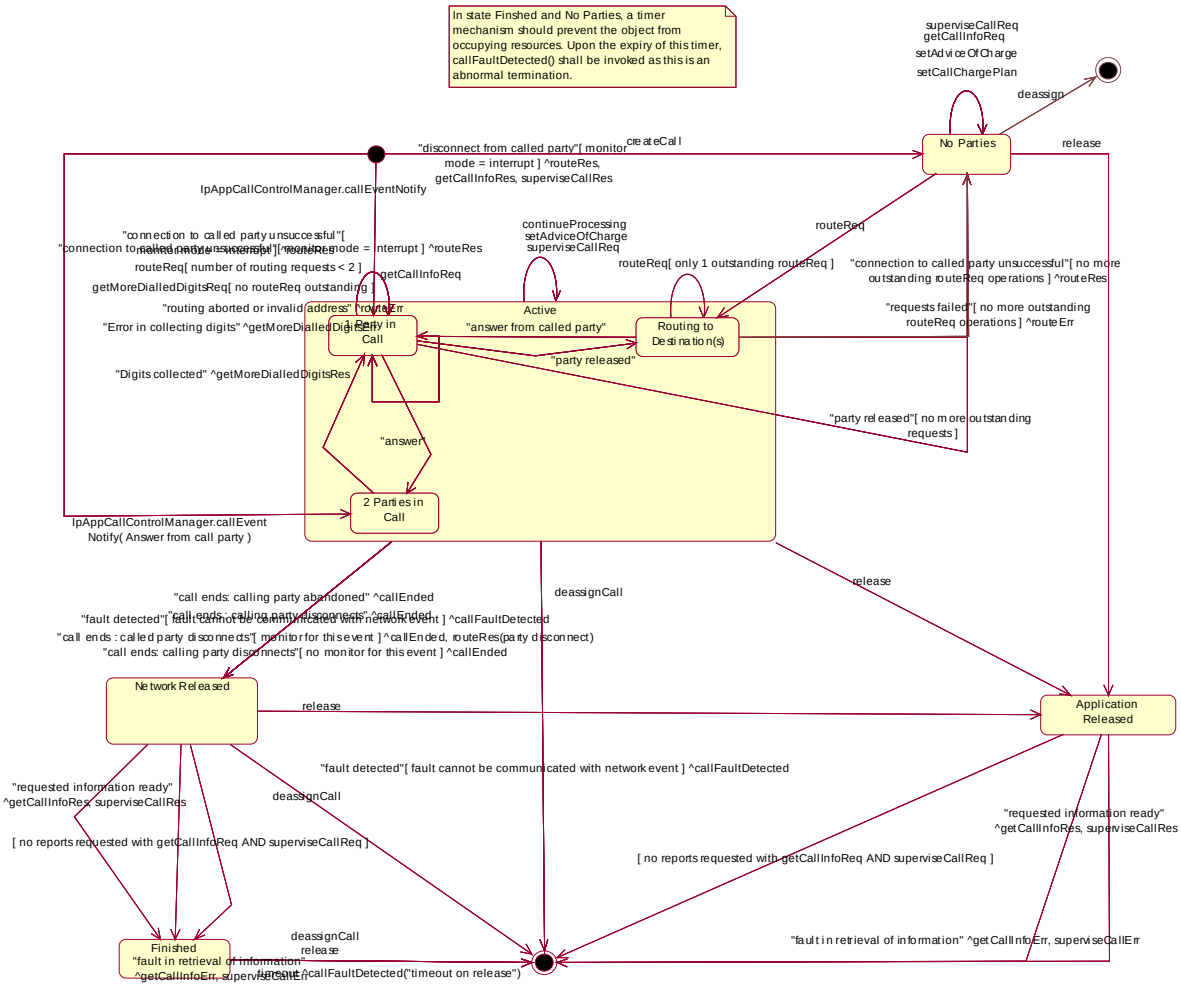


Figure 4: Application view on the IpCall object

7.2.1 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 Finished.

7.2.2 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.

7.2.3 Application Released State

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

7.2.4 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().

7.2.5 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() as well as to define the charging by invoking setCallChargePlan.

Call processing is suspended when a network event is met for the call, which was requested to be monitored in the P_CALL_MONITOR_MODE_INTERRUPT. In order to resume of the suspended call processing, the application invokes continueProcessing(), routeReq(), release() or deassignCall() method.

7.2.6 1 Party in Call State

In this state there is one party in the call.

In this state the application can 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 second party in the call by means of routeReq().

Two cases apply: network initiated calls and application initiated calls.

In case the call originated from the network the application can now request for more digits in case more digits are needed. When the calling party abandons the call before the application has invoked the routeReq() operation, the application is informed with callEnded(). 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().

In case the call was setup by the application and the called party was reached by issuing a routeReq() the application can request a connection to a second call party by calling the operation routeReq() again.

Otherwise, it depends on the actual number of invoked (and still outstanding or successful) routing requests whether the application can still call the routeReq() operation in order to setup a connection to a called party. Also in this case the called party can disconnect before another party is reached. In this case depending on the actual configuration, the call is ended or a transition is made back to the Routing to Destinations substate. When the second party answers the call, a transition will be made to the 2 Parties in Call state.

In this state user interaction is possible.

7.2.7 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.

7.2.8 Routing to Destination(s) State

In this state there is at least one outstanding routeReq.

8 Generic Call Control Service Properties

8.1 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 plans (defined in TpAddressPlan.) e.g. {P_ADDRESS_PLAN_E164, P_ADDRESS_PLAN_IP}). Note that more than one address plan may be supported.
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_NOTIFICATION_ADDRESS_RANGES	XML_ADDRESS_RANGE_SET	Indicates for which numbers notifications may be set. More than one range may be present. For terminating notifications they apply to the terminating number, for originating notifications they apply 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}.

Property	Type	Description
P_CHARGEPLAN_ALLOWED	INTEGER_SET	Indicates which charging is allowed in the setCallChargePlan indicator. Allowed values: {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.

8.2 Service Property values for the CAMEL Service Environment

Implementations of the Generic Call Control API relying on the CSE of CAMEL phase 4 shall have the Service Properties outlined above set to the indicated values:

```
P_OPERATION_SET = {
"IpCallControlManager.createCall",
"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_ALERTING,
P_CALL_REPORT_ANSWER,
P_CALL_REPORT_BUSY,
P_CALL_REPORT_NO_ANSWER,
P_CALL_REPORT_DISCONNECT,
P_CALL_REPORT_SERVICE_CODE,
P_CALL_REPORT_ROUTING_FAILURE,
P_CALL_REPORT_NOT_REACHABLE
}

P_ADDRESS_PLAN = {
P_ADDRESS_PLAN_E164
}

P_UI_CALL_BASED = {
TRUE
}

P_UI_AT_ALL_STAGES = {
FALSE
}

P_MEDIA_TYPE = {
P_AUDIO
}
```

9 Generic Call Control Data Definitions

This clause 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.

All data types referenced in the present document but not defined in this clause are defined either in the common call control data definitions in ES 204 915-4-1 or in the common data definitions which may be found in ES 204 915-2.

9.1 Generic Call Control Event Notification Data Definitions

9.1.1 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 getMoreDialledDigitsReq 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.

9.1.2 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	0	Indicates that the notification is related to the originating user in the call.
P_TERMINATING	1	Indicates that the notification is related to the terminating user in the call.

9.1.3 TpCallEventCriteria

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

9.1.4 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

9.2 Generic Call Control Data Definitions

9.2.1 IpCall

Defines the address of an `IpCall` Interface.

9.2.2 IpCallRef

Defines a `Reference` to type IpCall.

9.2.3 IpAppCall

Defines the address of an `IpAppCall` Interface.

9.2.4 IpAppCallRef

Defines a `Reference` to type IpAppCall.

9.2.5 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.

9.2.6 IpAppCallControlManager

Defines the address of an `IpAppCallControlManager` Interface.

9.2.7 IpAppCallControlManagerRef

Defines a `Reference` to type IpAppCallControlManager.

9.2.8 IpCallControlManager

Defines the address of an `IpCallControlManager` Interface.

9.2.9 IpCallControlManagerRef

Defines a `Reference` to type IpCallControlManager.

9.2.10 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

9.2.11 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

9.2.12 TpCallAppInfoSet

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

9.2.13 TpCallEndedReport

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

Sequence Element Name	Sequence Element Type	Description
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.

9.2.14 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.

9.2.15 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 as a result of a routeReq.
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.
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.

9.2.16 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 ITU-T Recommendation 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_NOT_REACHABLE	20	20
P_CALL_REPORT_ROUTING_FAILURE	3	Any other value

9.2.17 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

9.2.18 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
P_CALL_REPORT_QUEUED	TpString	QueueStatus
P_CALL_REPORT_NOT_REACHABLE	TpCallReleaseCause	NotReachable

9.2.19 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

9.2.20 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
P_CALL_REPORT_QUEUED	NULL	Undefined
P_CALL_REPORT_NOT_REACHABLE	NULL	Undefined

9.2.21 TpCallReportRequestSet

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

9.2.22 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.
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.
P_CALL_REPORT_NOT_REACHABLE	11	The called address is not reachable; e.g. the phone has been switched off or the phone is outside the coverage area of the network.

9.2.23 TpCallTreatment

Defines the `Sequence of Data Elements` that specify 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
CallTreatmentType	TpCallTreatmentType
ReleaseCause	TpCallReleaseCause
AdditionalTreatmentInfo	TpCallAdditionalTreatmentInfo

9.2.24 TpCallEventCriteriaResultSet

Defines a set of TpCallEventCriteriaResult.

9.2.25 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
CallEventCriteria	TpCallEventCriteria	The event criteria that were specified by the application.
AssignmentID	TpInt32	The associated assignmentID. This can be used to disable the notification.

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

The OMG IDL representation of this interface specification is contained in text files (gcc_data.idl and gcc_interfaces.idl) contained in archive es_2049150402IDL.zip which accompanies the present document.

This archive can be found in es_2049150402v010101m0.zip which accompanies the present document.

Annex B (informative): W3C WSDL Description of Generic Call Control SCF

The W3C WSDL representation of this interface specification is contained in zip file es_2049150402WSDL.zip which accompanies the present document.

This archive can be found in es_2049150402v010101m0.zip which accompanies the present document.

Annex C (informative): Java™ API Description of the Call Control SCFs

The Java™ API realisation of this interface specification is produced in accordance with the Java™ Realisation rules defined in ES 204 915-1. These rules aim to deliver for Java™, a developer API, provided as a realisation, supporting a Java™ API that represents the UML specifications. The rules support the production of both J2SE™ and J2EE™ versions of the API from the common UML specifications.

The J2SE™ representation of this interface specification is provided as Java™ Code, contained in archive 20491504-2J2SE.zip that accompanies the present document.

The J2EE™ representation of this interface specification is provided as Java™ Code, contained in archive 20491504-2J2EE.zip that accompanies the present document.

Both these archives can be found in es_2049150402v010101m0.zip which accompanies the present document.

Annex D (informative): Contents of 3GPP OSA Rel-7 Call Control

All items in Generic Call Control are relevant for TS 129 198-4-2 (Release 7).

Annex E (informative): Description of Call Control Sub-part 2: Generic call control SCF for 3GPP2 cdma2000 networks

This annex is intended to define the OSA API Stage 3 interface definitions and it provides the complete OSA specifications. It is an extension of OSA API specifications capabilities to enable operation in cdma2000 systems environment. They are in alignment with 3GPP2 Stage 1 requirements and Stage 2 architecture defined in [52], [53] and [54] of ES 204 915-1, clause 2. These requirements are expressed as additions to and/or exclusions from the 3GPP Release 7 specification. The information given here is to be used by developers in 3GPP2 cdma2000 network architecture to interpret the 3GPP OSA specifications.

E.1 General Exceptions

The terms 3GPP and UMTS are not applicable for the cdma2000 family of standards. Nevertheless these terms are used (TR 121 905) mostly in the broader sense of "3G Wireless System". If not stated otherwise there are no additions or exclusions required.

CAMEL and CAP mappings are not applicable for cdma2000 systems.

E.2 Specific Exceptions

E.2.1 Clause 1: Scope

There are no additions or exclusions.

E.2.2 Clause 2: References

Normative references on TS 123 078 and on TS 129 078 are not applicable for cdma2000 systems.

E.2.3 Clause 3: Definitions and abbreviations

There are no additions or exclusions.

E.2.4 Clause 4: Generic Call Control Service Sequence Diagrams

There are no additions or exclusions. Nevertheless, CAMEL and CAP mappings are not applicable for cdma2000 systems.

E.2.5 Clause 5: Class Diagrams

There are no additions or exclusions.

E.2.6 Clause 6: Generic Call Control Service Interface Classes

There are no additions or exclusions.

E.2.7 Clause 7: Generic Call Control Service State Transition Diagrams

There are no additions or exclusions.

E.2.8 Clause 8: Generic Call Control Service Properties

There are no additions or exclusions. Nevertheless, for cdma2000 systems the CAMEL data types and service properties are not applicable.

E.2.9 Clause 9: Generic Call Control Data Definitions

There are no additions or exclusions.

E.2.10 Annex A (normative): OMG IDL Description of Generic Call Control SCF

There are no additions or exclusions.

E.2.11 Annex B (informative): W3C WSDL Description of Generic Call Control SCF

There are no additions or exclusions.

E.2.12 Annex C (informative): Java™ API Description of the Call Control SCFs

There are no additions or exclusions.

Annex F (informative): Record of changes

The following is a list of the changes made to the present document for each release. The list contains the names of all changed, deprecated, added or removed items in the specifications and not the actual changes. Any type of change information that is important to the reader is put in the final clause of this annex.

Changes are specified as changes to the prior major release, but every minor release will have its own part of the table allowing the reader to know when the actual change was made.

F.1 Interfaces

F.1.1 New

Identifier	Comments
Interfaces added in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.1.2 Deprecated

Identifier	Comments
Interfaces deprecated in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.1.3 Removed

Identifier	Comments
Interfaces removed in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.2 Methods

F.2.1 New

Identifier	Comments
Methods added in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.2.2 Deprecated

Identifier	Comments
Methods deprecated in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.2.3 Modified

Identifier	Comments
Methods modified in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.2.4 Removed

Identifier	Comments
Methods removed in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.3 Data Definitions

F.3.1 New

Identifier	Comments
Data Definitions added in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.3.2 Modified

Identifier	Comments
Data Definitions modified in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.3.3 Removed

Identifier	Comments
Data Definitions removed in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.4 Service Properties

F.4.1 New

Identifier	Comments
Service Properties added in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.4.2 Deprecated

Identifier	Comments
Service Properties deprecated in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.4.3 Modified

Identifier	Comments
Service Properties modified in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.4.4 Removed

Identifier	Comments
Service Properties removed in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.5 Exceptions

F.5.1 New

Identifier	Comments
Exceptions added in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.5.2 Modified

Identifier	Comments
Exceptions modified in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.5.3 Removed

Identifier	Comments
Exceptions removed in ES 204 915-4-2 version 1.1.1 (Parlay 6.0)	

F.6 Others

None.

History

Document history		
V1.1.1	February 2008	Membership Approval Procedure MV 20080425: 2008-02-26 to 2008-04-25