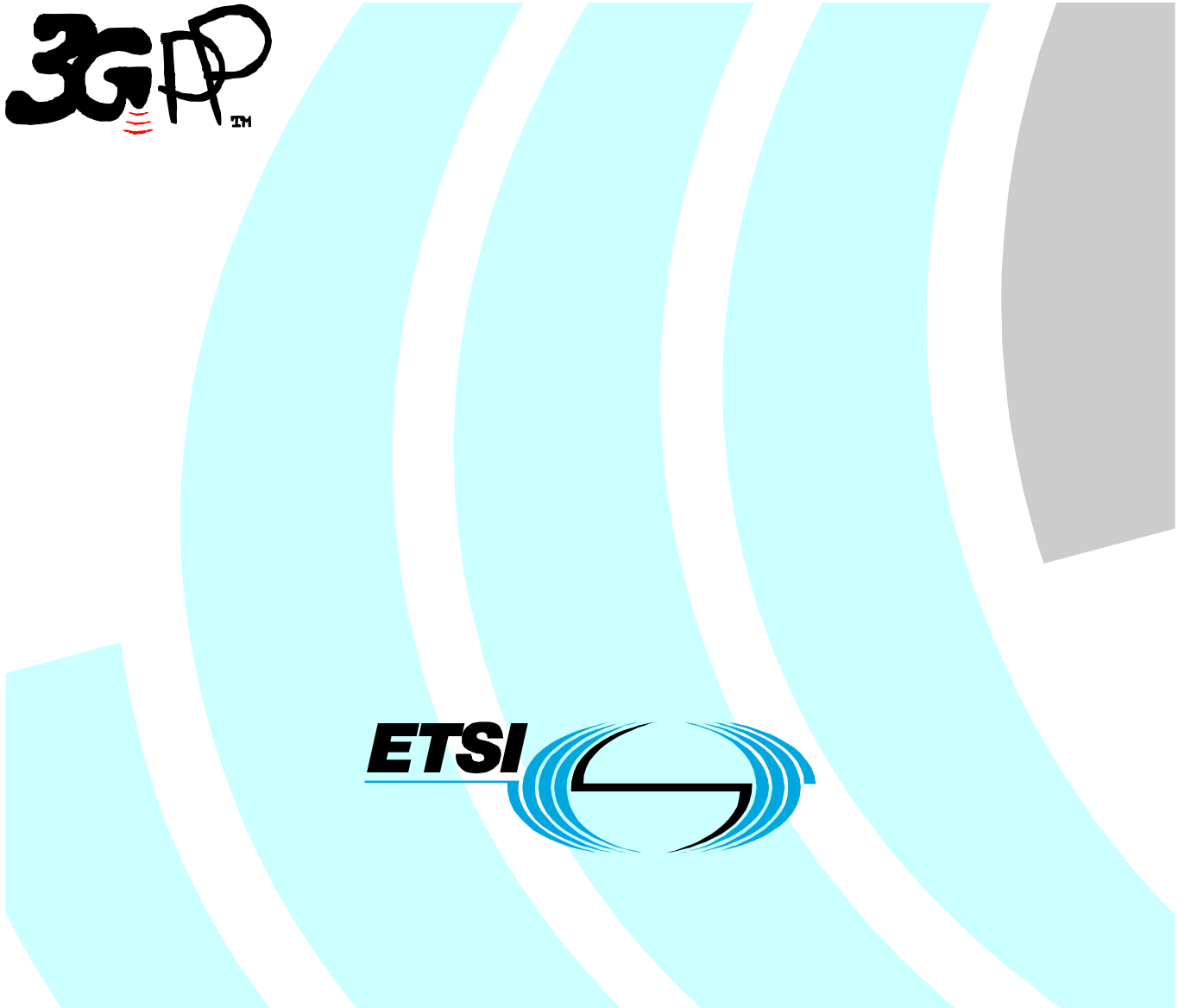
ETSI TS 129 198-4-3 V5.9.1 (2004-12)

Technical Specification

Universal Mobile Telecommunications System (UMTS);
Open Service Access (OSA)
Application Programming Interface (API);
Part 4: Call control;
Subpart 3: Multi-party call control
Service Capability Feature (SCF)
(3GPP TS 29.198-04-3 version 5.9.1 Release 5)



Reference
RTS/TSGN-0529198-04-3v591

Keywords
UMTS

ETSI

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

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

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

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://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 Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

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

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp .

Contents

Foreword

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

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

Version x.y.z

where:

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

Introduction

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

Part 1:	"Overview";	
Part 2:	"Common Data Definitions";	
Part 3:	"Framework";	
Part 4:	**"Call Control";**	
	Sub-part 1: "Call Control Common Definitions";	(new in 3GPP Release 5)
	Sub-part 2: "Generic Call Control SCF";	(new in 3GPP Release 5)
	Sub-part 3: "Multi-Party Call Control SCF";	**(new in 3GPP Release 5)**
	Sub-part 4: "Multi-Media Call Control SCF";	(new in 3GPP Release 5)
	Sub-part 5: "Conference Call Control SCF";	(not part of 3GPP Release 5)
Part 5:	"User Interaction SCF";	
Part 6:	"Mobility SCF";	
Part 7:	"Terminal Capabilities SCF";	
Part 8:	"Data Session Control SCF";	
Part 9:	"Generic Messaging SCF";	(not part of 3GPP Release 5)
Part 10:	"Connectivity Manager SCF";	(not part of 3GPP Release 5)
Part 11:	"Account Management SCF";	
Part 12:	"Charging SCF".	
Part 13:	"Policy Management SCF";	(new in 3GPP Release 5)
Part 14:	"Presence and Availability Management SCF";	(new in 3GPP Release 5)

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

Table: Overview of the OSA APIs & Protocol Mappings 29.198 & 29.998-family

OSA API specifications 29.198-family					**OSA API Mapping - 29.998-family**	
29.198-01	Overview				29.998-01	Overview
29.198-02	Common Data Definitions				*29.998-02*	*Not Applicable*
29.198-03	Framework				*29.998-03*	*Not Applicable*
Call Control (CC) SCF	29.198-04-1 Common CC data definitions	29.198-04-2 Generic CC SCF	**29.198-04-3 Multi-Party CC SCF**	29.198-04-4 Multi-media CC SCF	29.998-04-1	Generic Call Control – CAP mapping
					29.998-04-2	*Generic Call Control – INAP mapping*
					29.998-04-3	*Generic Call Control – Megaco mapping*
					29.998-04-4	Multiparty Call Control – SIP mapping
29.198-05	User Interaction SCF				29.998-05-1	User Interaction – CAP mapping
					29.998-05-2	*User Interaction – INAP mapping*
					29.998-05-3	*User Interaction – Megaco mapping*
					29.998-05-4	User Interaction – SMS mapping
29.198-06	Mobility SCF				29.998-06	User Status and User Location – MAP mapping
29.198-07	Terminal Capabilities SCF				*29.998-07*	*Not Applicable*
29.198-08	Data Session Control SCF				29.998-08	Data Session Control – CAP mapping
29.198-09	*Generic Messaging SCF*				*29.998-09*	*Not Applicable*
29.198-10	*Connectivity Manager SCF*				*29.998-10*	*Not Applicable*
29.198-11	Account Management SCF				*29.998-11*	*Not Applicable*
29.198-12	Charging SCF				*29.998-12*	*Not Applicable*
29.198-13	Policy Management SCF				*29.998-13*	*Not Applicable*
29.198-14	Presence & Availability Management SCF				*29.998-14*	*Not Applicable*

1 Scope

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

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

The present document specifies the Multi-Party Call Control Service Capability Feature (SCF) aspects of the interface. All aspects of the Multi-Party 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).

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI TISPAN and the Parlay Group, in co-operation with a number of JAIN™ Community member companies.

2 References

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

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

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

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

[3] 3GPP TS 23.127: "Virtual Home Environment (VHE) / Open Service Access (OSA)".

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

[5] ISO 4217 (1995): "Codes for the representation of currencies and funds ".

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

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

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

[9] ANSI T1.113: "Signalling System No. 7 (SS7) - Integrated Services Digital Network (ISDN) User Part".

3 Definitions and abbreviations

3.1 Definitions

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

3.2 Abbreviations

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

4 MultiParty Call Control Service Sequence Diagrams

The Multi-Party Call Control API of 3GPP Rel4 relies on the CAMEL Service Environment (CSE). It should be noted that a number of restrictions exist because CAMEL phase 3 supports only two-party calls and no leg based operations. Furthermore application initiated calls are not supported in CAMEL phase 3. The detailed description of the supported methods is given in the chapter 8.

4.1 Application initiated call setup

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

The service may as a variation be extended to include 3 parties (or more). After the two party call is established, the application can create a new leg and request to route it to a new destination address in order to establish a 3 party call.

The event that causes this to happen could for example be the report of answer event from B-party or controlled by the A-party by entering a service code (mid-call event).

The procedure for call setup to party C is exactly the same as for the set up of the connection to party B (sequence 13 to 17 in the sequence diagram).

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

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

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

4: Once the object implementing the IpMultiPartyCall interface is created it is used to pass the reference of the object implementing the IpAppMultiPartyCall interface as the callback reference to the object implementing the IpMultiPartyCall interface. Note that the reference to the callback interface could already have been passed in the createCall.

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

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

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

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

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

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

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

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

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

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

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

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

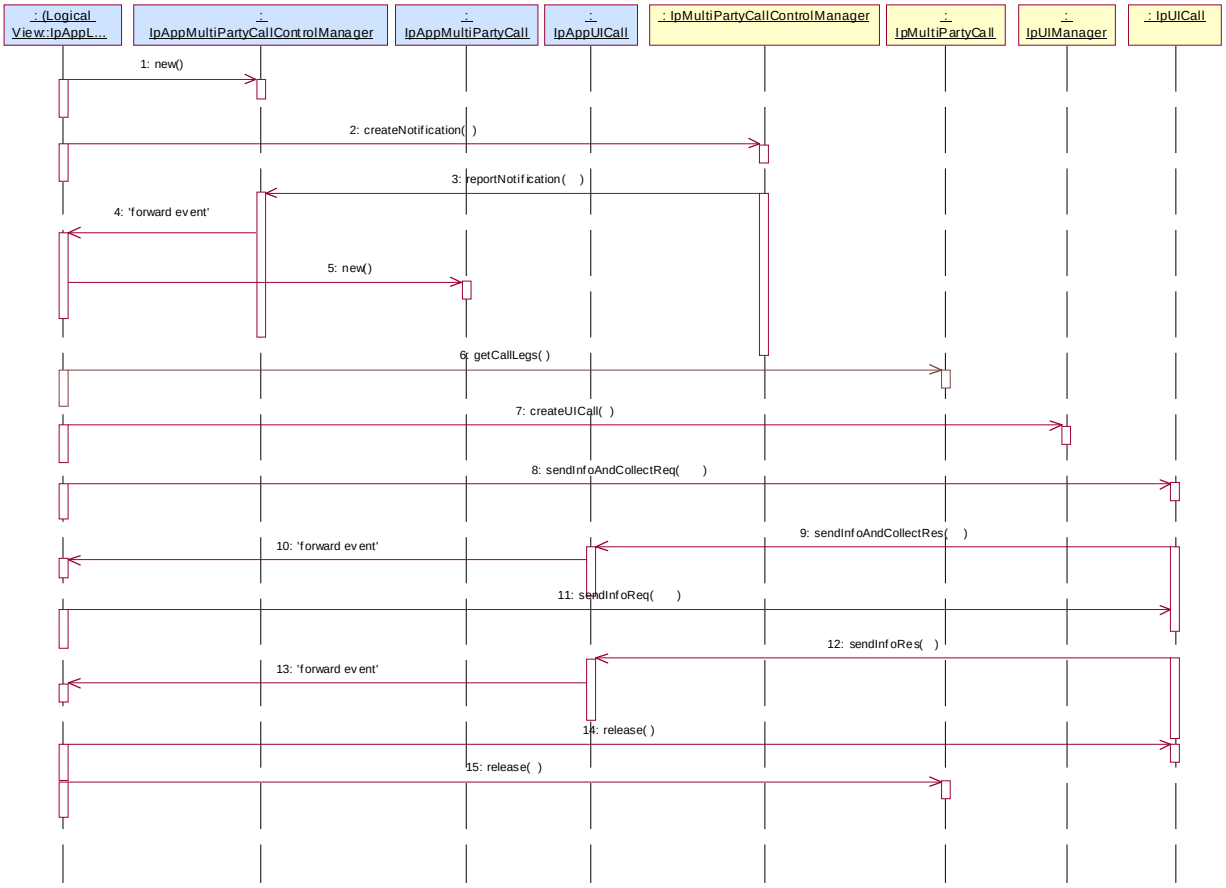
17: Assuming the call is answered, the object implementing party B's IpCallLeg interface passes the result of the call being answered back to its callback object. This message is then forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

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

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

4.2 Call Barring 2

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



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

2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for

a password before the call is allowed to progress. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

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

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

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

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

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

8: The call barring service dialogue is invoked.

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

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

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

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

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

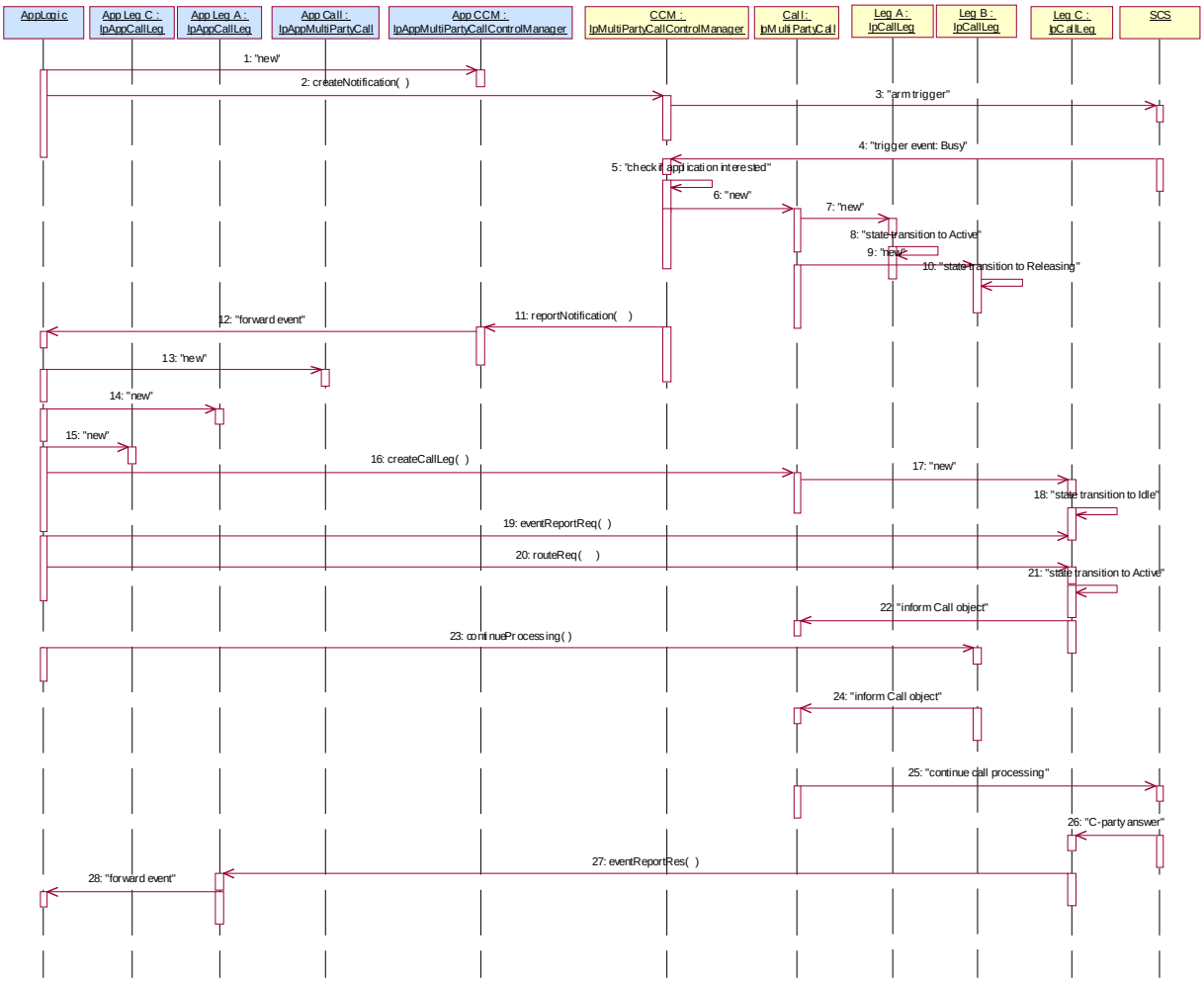
14: No more UI is required, so the UICall object is released.

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

4.3 Call forwarding on Busy Service

The following sequence diagram shows an application establishing a call forwarding on busy.

When a call is made from A to B but the B-party is detected to be busy, then the application is informed of this and sets up a connection towards a C party. The C party can for instance be a voicemail system.



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

2: This message is sent by the application to enable notifications on new call events.

4: When a new call, that matches the event criteria, arrives a message ("busy") is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface is met, other messages are used to create the call and associated call leg objects.

6: A new MultiPartyCall object is created to handle this particular call.

7: A new CallLeg object corresponding to Party A is created.

8: The new Call Leg instance transits to state Active.

11: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface. Applied monitor mode is "interrupt".

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

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

14: A new AppCallLeg is created to receive callbacks for the Leg corresponding to party A.

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

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

19: The application requests to be notified (monitor mode "INTERRUPT") when party C answers the call.

20: The application requests to route the terminating leg to reach the associated party C.

The application may request information about the original destination address be sent by setting up the field P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS of TpCallAppInfo in the request to route the call leg to the remote party C.

23: The application requests to resume call processing for the terminating call leg to party B to terminate the leg. Alternative the application could request to deassign the leg to party B for example if it is not interested in possible requested call leg information (getInfoRes, superviseRes).

When the terminating call leg is destroyed, the AppLeg B is notified (callLegEnded) and the event is forwarded to the application logic (not shown).

25: As a result call processing is resumed in the network that will try to reach the associated party C.

26: When the party C answers the call, the termination call leg is notified.

27: Assuming the call is answered, the object implementing party C's IpCallLeg interface passes the result of the call being answered back to its callback object.

28: This answer message is then forwarded to the object implementing the IpAppLogic interface.

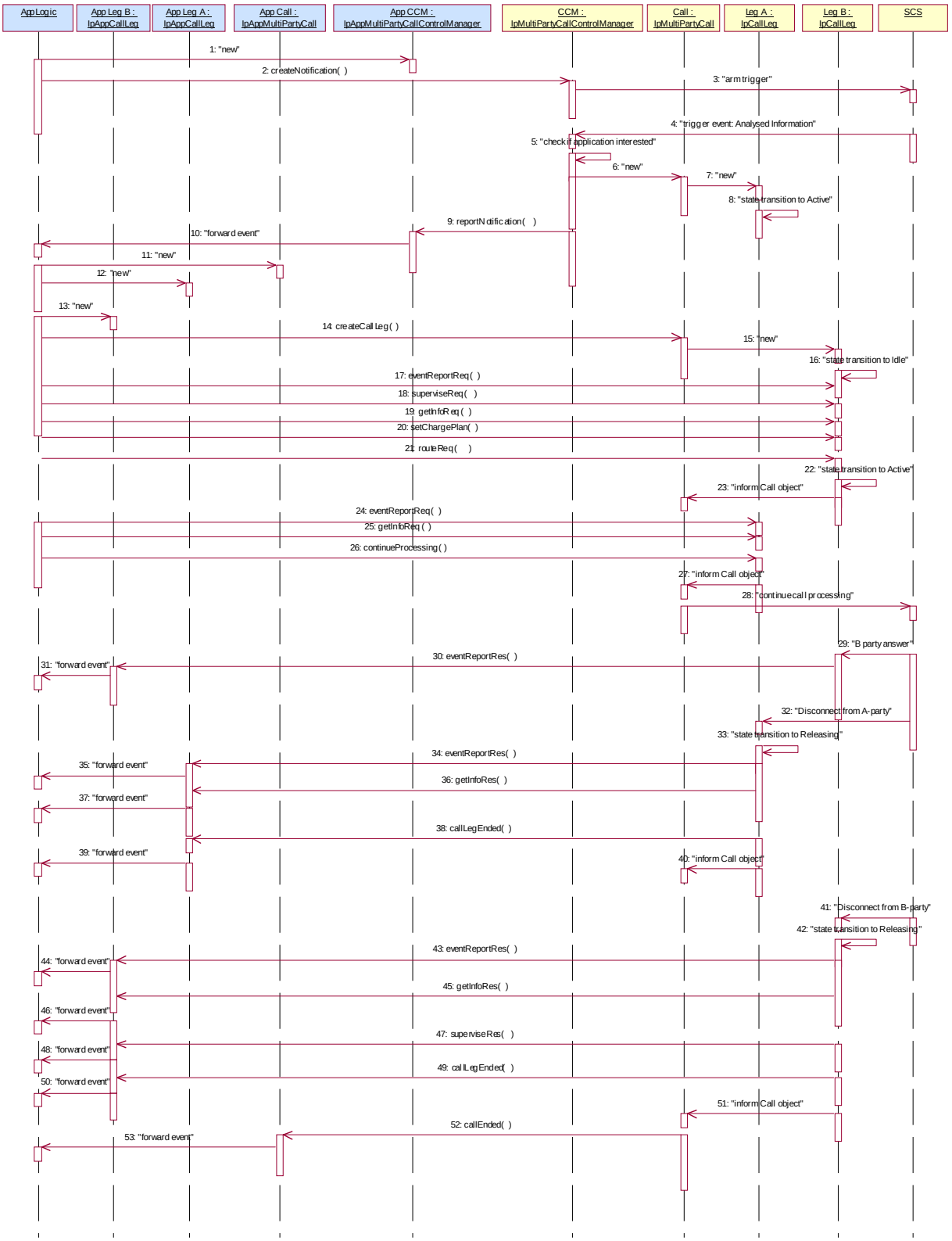
4.4 Call Information Collect Service

The following sequence diagram shows an application monitoring a call between party A and a party B in order to collect call information at the end of the call for e.g. charging and/or statistic information collection purposes. The service may apply to ordinary two-party calls, but could also include a number translation of the dialled number and special charging (e.g. a premium rate service).

Additional call leg related information is requested with the getInfoReq and superviseReq methods.

The answer and call release events are in this service example requested to be reported in notify mode and additional call leg related information is requested with the getInfoReq and superviseReq methods in order to illustrate the information that can be collected and sent to the application at the end of the call.

Furthermore the diagram shows the order in which information is sent to the application: network release event followed by possible requested call leg information, then the destroy of the call leg object (callLegEnded) and finally the destroy of the call object (callEnded).



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

2: This message is sent by the application to enable notifications on new call events.

4: When a new call, that matches the event criteria, arrives a message ("analysed information") is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface is met, other messages are used to create the call and associated call leg object.

6: A new MultiPartyCall object is created to handle this particular call.

7: A new CallLeg object corresponding to Party A is created.

8: The new Call Leg instance transits to state Active.

9: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface. Applied monitor mode is "interrupt".

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

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

12: A new AppCallLeg is created to receive callbacks for the Leg corresponding to party A.

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

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

15: A new CallLeg corresponding to party B is created.

16: A transition to state Idle is made after the Call leg has been created.

17: The application requests to be notified (monitor mode "NOTIFY") when party B answers the call and when the leg to B-party is released.

18: The application requests to supervise the call leg to party B.

19: The application requests information associated with the call leg to party b for example to calculate charging.

20: The application requests a specific charge plan to be set for the call leg to party B.

21: The application requests to route the terminating leg to reach the associated party B.

22: The Call Leg instance transits to state Active.

24: The application requests to be notified (monitor mode "Notify") when the leg to A-party is released.

25: The application requests information associated with the call leg to party A for example to calculate charging.

26: The application requests to resume call processing for the originating call leg. As a result call processing is resumed in the network that will try to reach the associated party B.

29: When the B-party answers the call, the termination call leg is notified.

30: Assuming the call is answered, the object implementing party B's IpCallLeg interface passes the result of the call being answered back to its callback object (monitor mode "NOTIFY").

31: This answer message is then forwarded.

32: When the A-party releases the call, the originating call leg is notified (monitor mode "NOTIFY") and makes a transition to "releasing state".

34: The application IpAppLeg A is notified, as the release event has been requested to be reported in Notify mode.

35: The event is forwarded to the application logic.

36: The call leg information is reported.

37: The event is forwarded to the application logic.

38: The origination call leg is destroyed, the AppLeg A is notified.

39: The event is forwarded to the application logic.

41: When the B-party releases the call or the call is released as a result of the release request from party A, i.e. a "originating release" indication, the terminating call leg is notified and makes a transition to "releasing state".

43: If a network release event is received being a "terminating release" indication from called party B, the application IpAppLeg B is notified, as the release event from party B has been requested to be reported in NOTIFY mode.

Note that no report is sent if the release is caused by propagation of network release event being a "originating release" indication coming from calling party A.

44: The event is forwarded to the application logic.

45: The call leg information is reported.

46: The event is forwarded to the application logic.

47: The supervised call leg information is reported.

48: The event is forwarded to the application logic.

49: The terminating call leg is destroyed, the AppLeg B is notified.

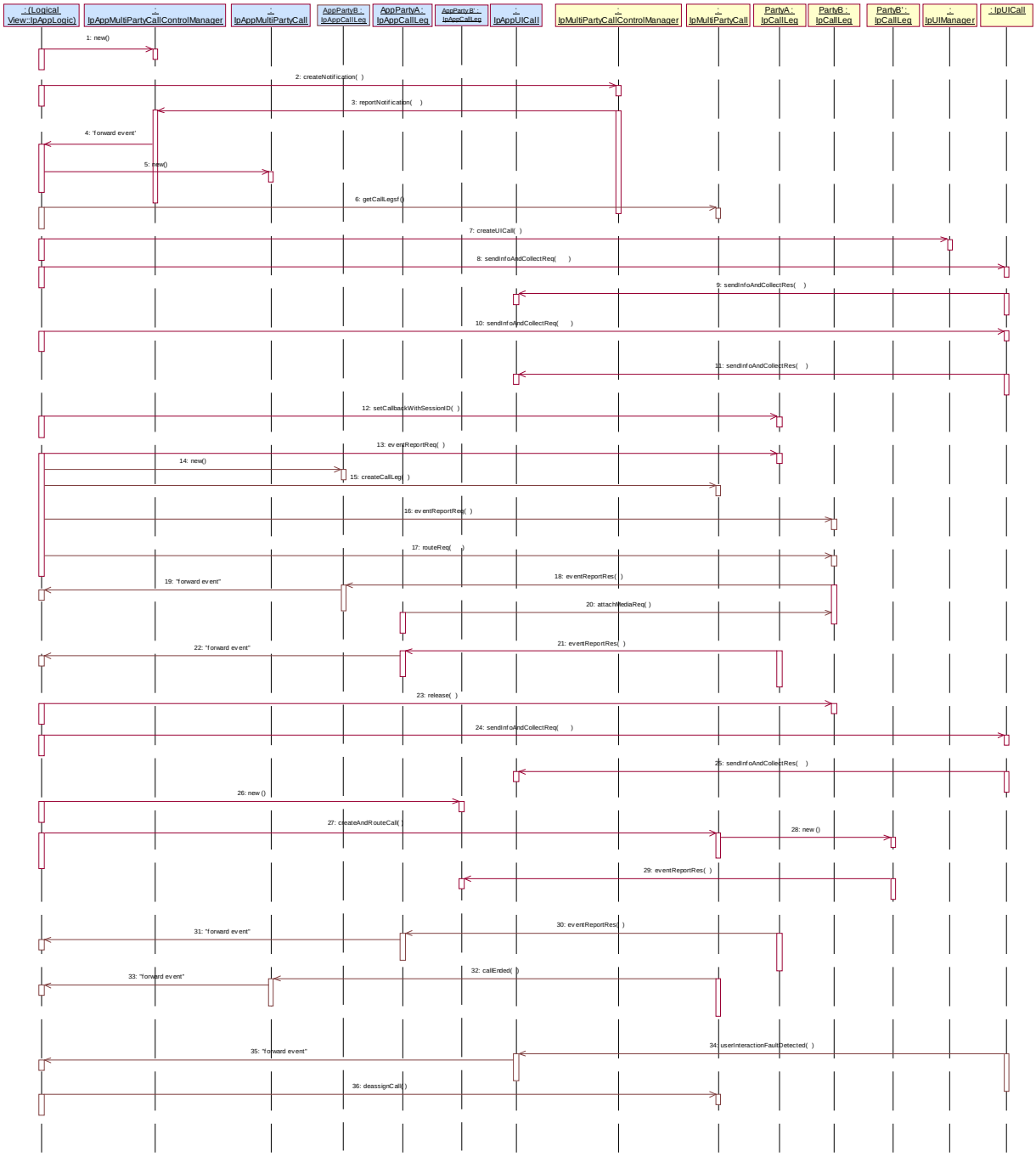
50: The event is forwarded to the application logic.

52: Assuming the IpCall object has been informed that the legs have been destroyed, the IpAppMultiPartyCall is notified that the call is ended .

53: The event is forwarded to the application logic.

4.5 Complex Card Service

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

19: The event is forwarded to the application logic.

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

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

22: The event is forwarded to the application.

23: This message releases the called party.

24: Another user interaction dialogue is invoked.

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

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

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

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

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

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

31: The event is forwarded to the application logic.

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

33: The event is forwarded to the application logic.

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

35: The event is forwarded to the application logic.

36: The application deassigns the call object.

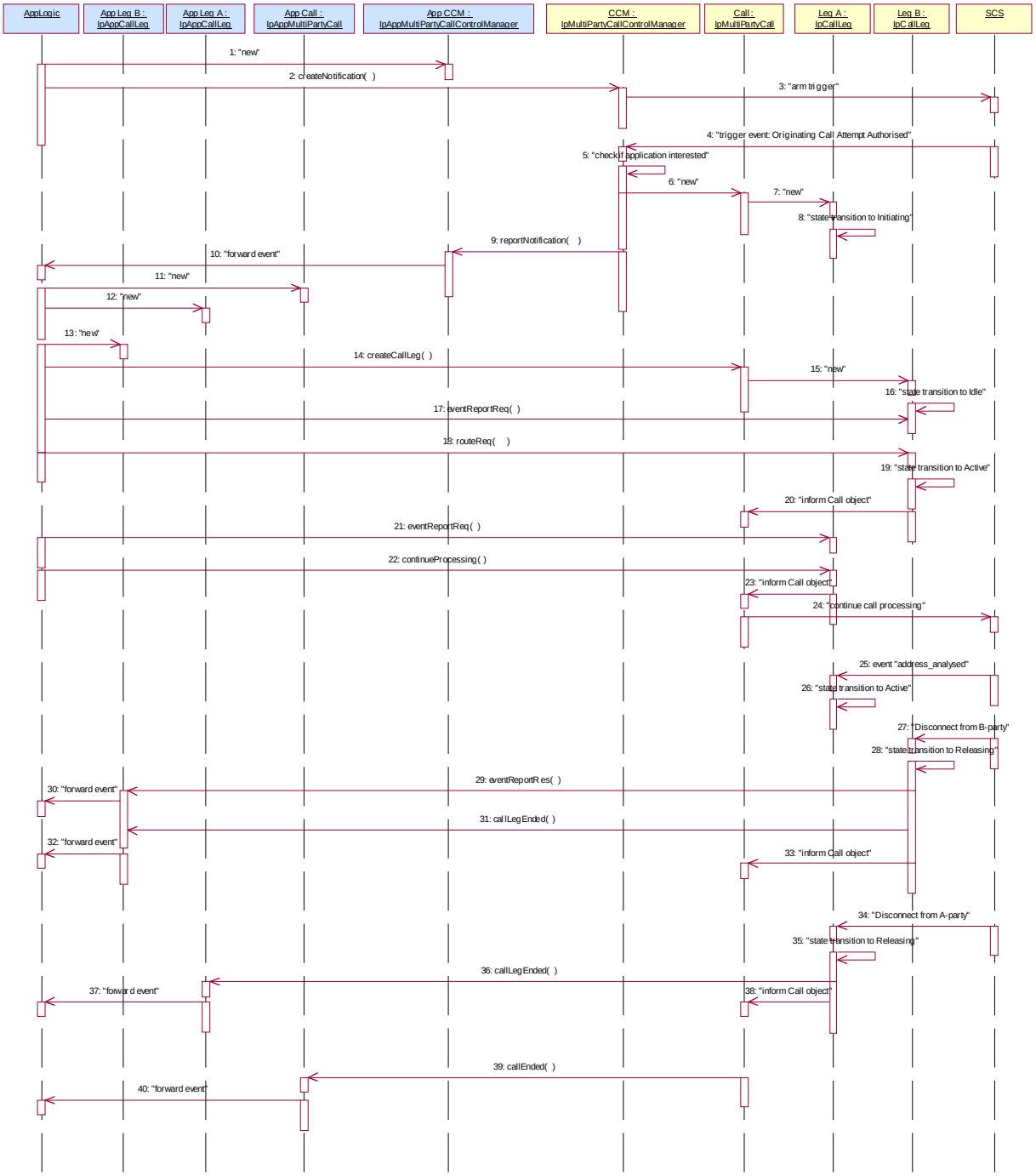
4.6 Hotline Service

The following sequence diagram shows an application establishing a call between party A and pre-arranged party B defined to constitute a hot-line address. The address of the destination party is provided by the application as the calling party makes a call attempt (goes off-hook) and do not dial any number within a predefined time. In this case a pre-defined number (hot-line number) is provided by the application. The call is then routed to the pre-defined destination party.

The call release is monitored to enable the sending of information to the application at call release, e.g. for charging purposes.

Note that this service could be extended as follows:

Sometime during the call the calling party enters '#5' which causes the called leg to be released. The calling party is now prompted to enter the address of a new destination party, to which it is then routed.



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

2: This message is sent by the application to enable notifications on new call events.

4: When a new call, that matches the event criteria, arrives a message ("originating call attempt authorised") is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface is met, other messages are used to create the call and associated call leg object.

6: A new MultiPartyCall object is created to handle this particular call.

7: A new CallLeg object corresponding to Party A is created.

8: The new Call Leg instance transits to state Initiating.

9: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface. Applied monitor mode is "interrupt".

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

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

12: A new AppCallLeg is created to receive callbacks for the Leg corresponding to party A.

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

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

15: A new CallLeg corresponding to party B is created.

16: A transition to state Idle is made after the Call leg has been created.

17: The application requests to be notified (monitor mode "NOTIFY") when the leg to party B is released.

18: The application requests to route the terminating leg to reach the associated party as specified by the application ("hot-line number").

19: The Call Leg instance transits to state Active.

21: The application requests to be notified (monitor mode "Notify") when the leg to A-party is released.

22: The application requests to resume call processing for the originating call leg. As a result call processing is resumed in the network that will try to reach the associated party as specified by the application (E.164 number provided by application).

25: The originating call leg is notified that the number (provided by application) has been analysed by the network and the originating call leg STD makes a transition to "active" state. The application is not notified as it has not requested this event to be reported.

27: When the B-party releases the call, the terminating call leg is notified (monitor mode "NOTIFY") and makes a transition to "Releasing state".

29: The application is notified, as the release event has been requested to be reported in Notify mode.

30: The event is forwarded to the application logic.

31: The terminating call leg is destroyed, the AppLeg B is notified.

32: This answer message is then forwarded.

34: When the call release ("terminating release" indication) is propagated in the network toward the party A, the originating call leg is notified and makes a transition to "releasing state". This release event (being propagated from party B) is not reported to the application.

36: When the originating call leg is destroyed, the AppLeg A is notified.

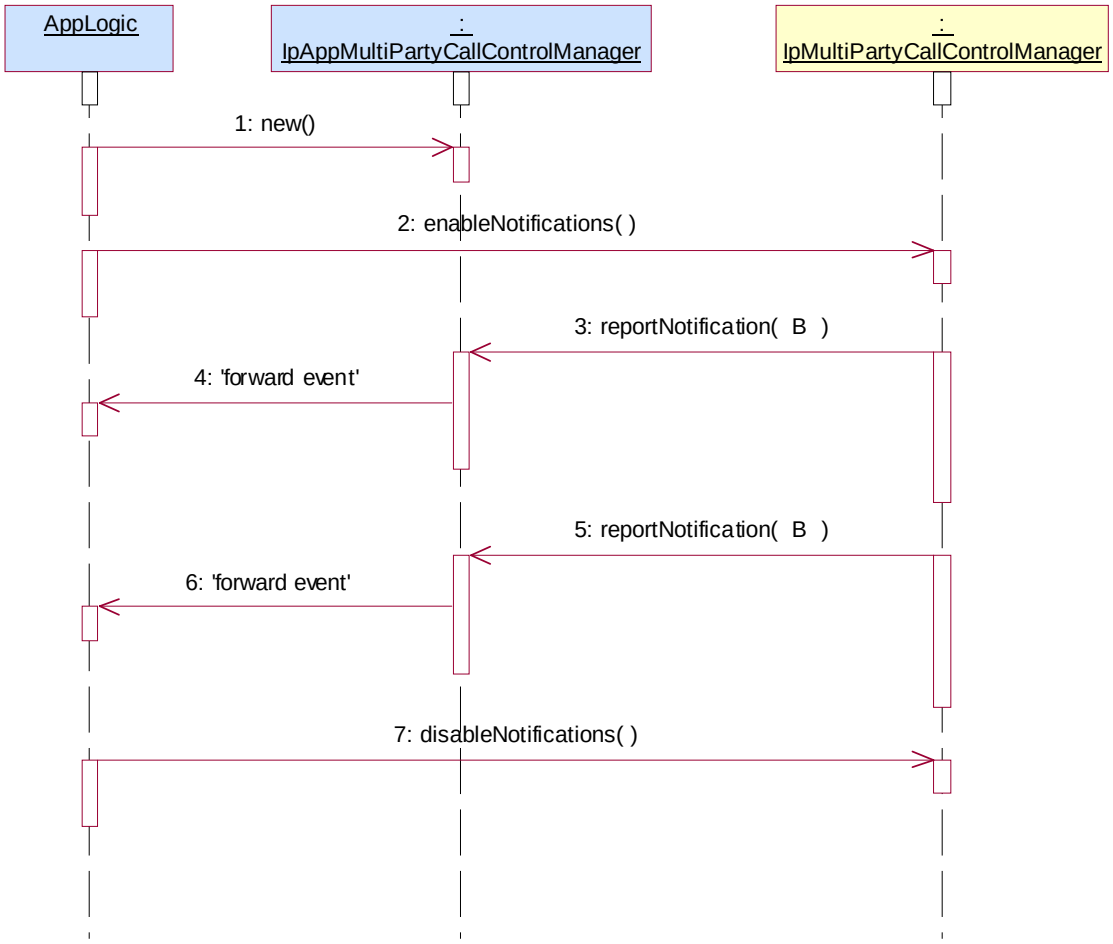
37: The event is forwarded to the application logic.

39: When all legs have been destroyed, the IpAppMultiPartyCall is notified that the call is ended.

40: The event is forwarded to the application logic.

4.7 Network Controlled Notifications

The following sequence diagram shows how an application can receive notifications that have not been created by the application, but are provisioned from within the network.



1: The application is started. The application creates a new IpAppMultiPartyCallControlManager to handle callbacks.

2: The enableNotifications method is invoked on the IpMultiPartyCallControlManager interface to indicate that the application is ready to receive notifications that are created in the network. For illustrative purposes we assume notifications of type "B" are created in the network.

3: When a network created trigger occurs the application is notified on the callback interface.

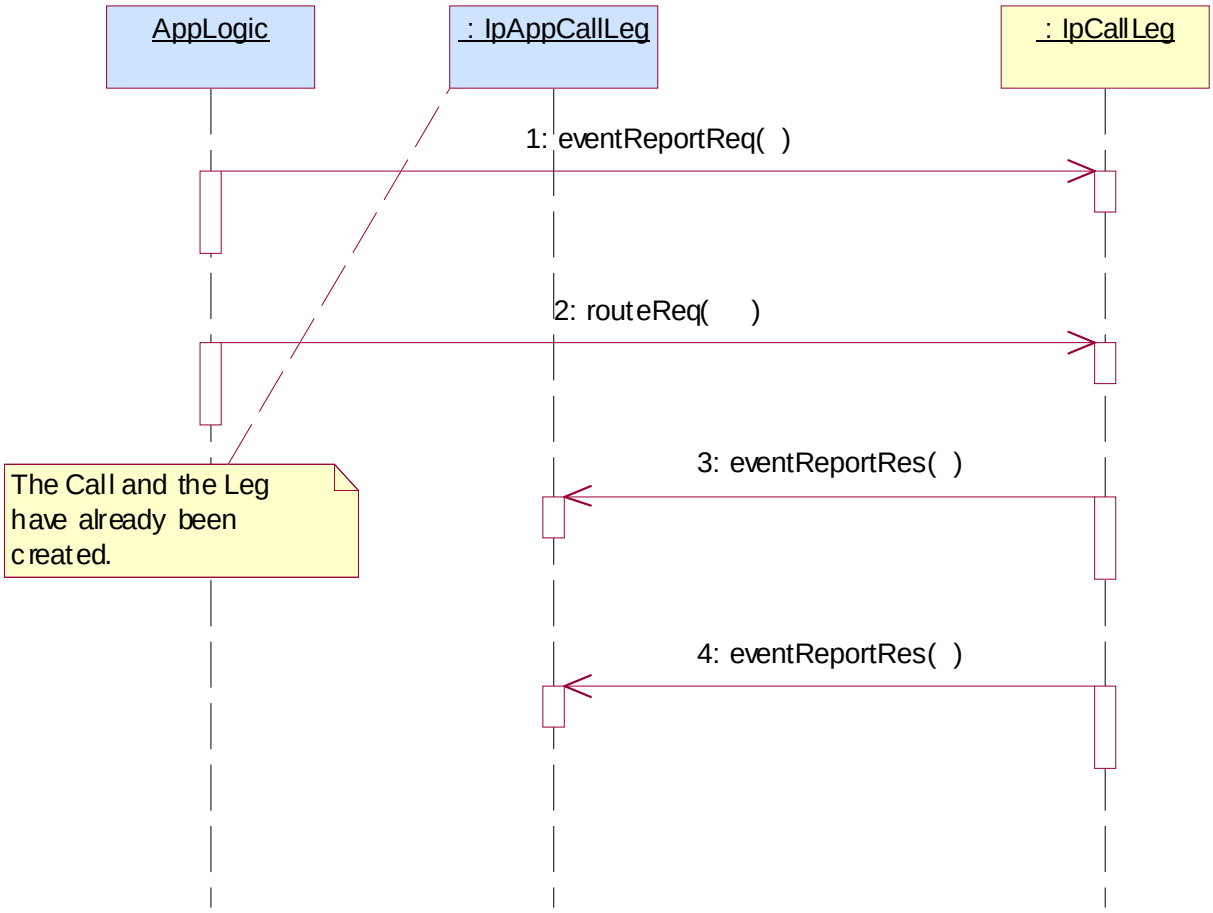
4: The event is forwarded to the application.

5: When a network created trigger occurs the application is notified on the callback interface.

6: The event is forwarded to the application.

7: When the application does not want to receive notifications created in the network anymore, it invokes disableNotifications on the IpMultiPartyCallConrolManager interface. From now on the gateway will not send any notifications to the application that are created in the network.

4.8 Use of the Redirected event



1: The application has already created the call and a call leg. It places an event report request for the ANSWER and REDIRECTED events in NOTIFY mode.

2: The application routes the call leg.

3: The call is redirected within the network and the application is informed. The new destination address is passed within the event. The event is not disarmed, so subsequent redirections will also be reported. Also, the same call leg is used so the application does not have to create a new one.

4: The call is answered at its new destination.

5 Class Diagrams

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

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

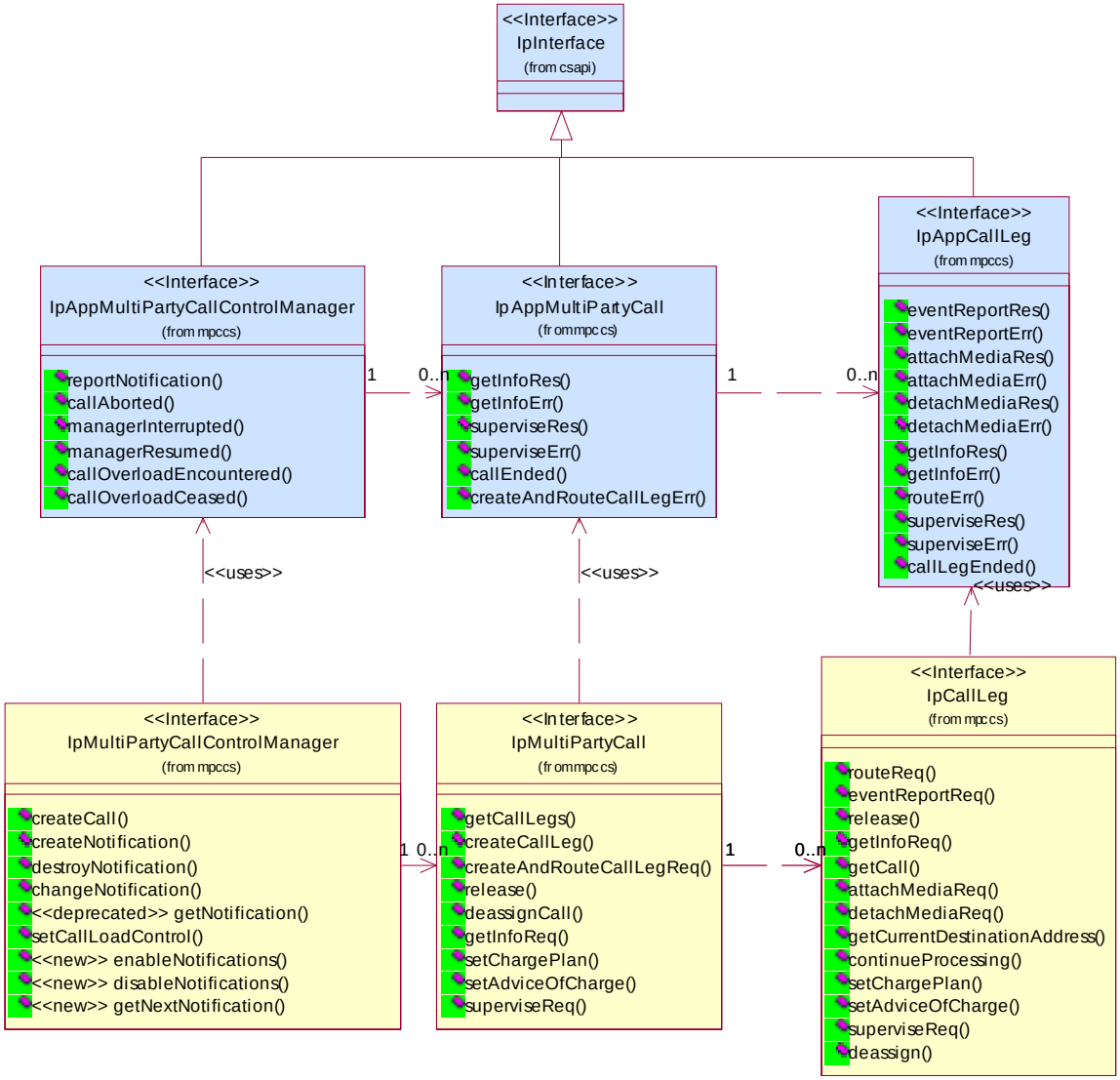


Figure: Application Interfaces

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

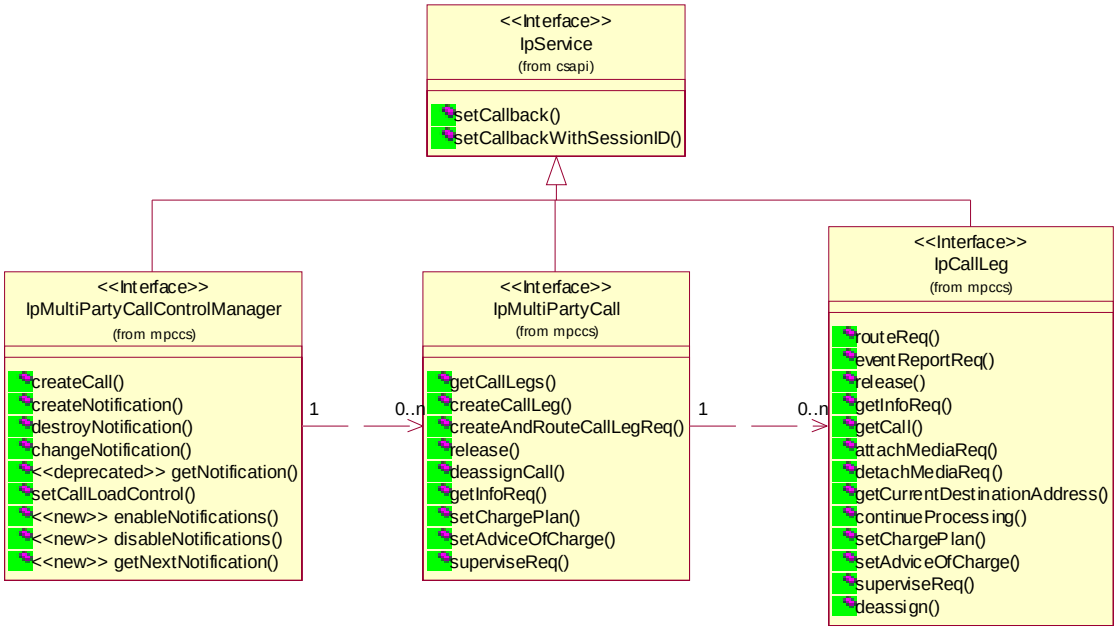


Figure: Service Interfaces

6 MultiParty Call Control Service Interface Classes

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

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

6.1 Interface Class IpMultiPartyCallControlManager

Inherits from: IpService

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

This interface shall be implemented by a Multi Party Call Control SCF. As a minimum requirement either the createCall() method shall be implemented, or the createNotification() and destroyNotification() methods shall be implemented, or the enableNotifications() and disableNotifications() methods shall be implemented.

<<Interface>>

IpMultiPartyCallControlManager

createCall (appCall : in IpAppMultiPartyCallRef) : TpMultiPartyCallIdentifier

createNotification (appCallControlManager : in IpAppMultiPartyCallControlManagerRef, notificationRequest : in TpCallNotificationRequest) : TpAssignmentID

destroyNotification (assignmentID : in TpAssignmentID) : void

changeNotification (assignmentID : in TpAssignmentID, notificationRequest : in TpCallNotificationRequest) : void

<<deprecated>> getNotification () : TpNotificationRequestedSet

setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange) : TpAssignmentID

<<new>> enableNotifications (appCallControlManager : in IpAppMultiPartyCallControlManagerRef) : TpAssignmentID

<<new>> disableNotifications () : void

<<new>> getNextNotification (reset : in TpBoolean) : TpNotificationRequestedSetEntry

6.1.1 Method createCall()

This method is used to create a new call object. An IpAppMultiPartyCallControlManager should already have been passed to the IpMultiPartyCallControlManager, otherwise the call control will not be able to report a callAborted() to the application. The application 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 IpAppMultiPartyCallRef`

Specifies the application interface for callbacks from the call created.

Returns

`TpMultiPartyCallIdentifier`

Raises

`TpCommonExceptions, P_INVALID_INTERFACE_TYPE`

6.1.2 Method createNotification()

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

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

If some application already requested notifications with criteria that overlap the specified criteria or the specified criteria overlap with criteria already present in the network (when provisioned from within the network), the request is refused with P_INVALID_CRITERIA. The criteria are said to overlap when it leads to more than one application controlling the call or session at the same point in time during call or session processing.

If a notification is requested by an application with 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 overlaps.

Setting the callback reference:

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

Case a:

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

Case b:

The createNotification() with no callback 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 createNotification() 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().

Set additional callback:

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. Both notifications will share the same assignmentID. The gateway will always use the most recent callback. In case this most recent callback fails the second most recent is used.

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

Parameters

`appCallControlManager : in IpAppMultiPartyCallControlManagerRef`

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

`notificationRequest : in TpCallNotificationRequest`

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

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_INVALID_CRITERIA, P_INVALID_INTERFACE_TYPE, P_INVALID_EVENT_TYPE

6.1.3 Method destroyNotification()

This method is used by the application to disable call notifications. This method only applies to notifications created with createNotification().

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID given by the multi party call control manager interface when the previous createNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the exception P_INVALID_ASSIGNMENTID will be raised. If two callbacks have been registered under this assignment ID both of them will be disabled.

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

6.1.4 Method changeNotification()

This method is used by the application to change the event criteria introduced with createNotification. 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 multi party call control manager interface for the event notification. If two callbacks have been registered under this assignment ID both of them will be changed.

notificationRequest : in TpCallNotificationRequest

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

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

6.1.5 Method <<deprecated>> getNotification()

This method is deprecated and replaced by getNextNotification(). It will be removed in a later release.

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

Returns notificationsRequested: Specifies the notifications that have been requested by the application. An empty set is returned when no notifications exist.

Parameters
No Parameters were identified for this method

Returns

TpNotificationRequestedSet

Raises

TpCommonExceptions

6.1.6 Method setCallLoadControl()

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

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

6.1.7 Method <<new>> enableNotifications()

This method is used to indicate that the application is able to receive notifications which are provisioned from within the network (i.e. these notifications are NOT set using createNotification() but via, for instance, a network management system). If notifications provisioned for this application are created or changed, the application is unaware of this until the notification is reported.

Setting the callback reference:

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

Case a:

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

Case b:

The enableNotifications() with no callback 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 enableNotifications() 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().

Set additional Call back:

If the same application requests to enable notifications for a second time with a different IpAppMultiPartyCallControlManager reference (i.e. without first disabling them), the second callback will be treated as an additional callback. The gateway will always use the most recent callback. In case this most recent callback fails the second most recent is used.

When this method is used, it is still possible to use createNotification() for service provider provisioned notifications on the same interface as long as the criteria in the network and provided by createNotification() do not overlap. However, it is NOT recommended to use both mechanisms on the same service manager.

The methods changeNotification(), getNotification(), and destroyNotification() do not apply to notifications provisioned in the network and enabled using enableNotifications(). These only apply to notifications created using createNotification().

Returns assignmentID: Specifies the ID assigned by the manager interface for this operation. This ID is contained in any reportNotification() that relates to notifications provisioned from within the network. Repeated calls to enableNotifications() return the same assignment ID.

Parameters

`appCallControlManager : in IpAppMultiPartyCallControlManagerRef`

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

Returns

`TpAssignmentID`

Raises

`TpCommonExceptions`

6.1.8 Method <<new>> disableNotifications()

This method is used to indicate that the application is not able to receive notifications for which the provisioning has been done from within the network. (i.e. these notifications that are NOT set using createNotification() but via, for instance, a network management system). After this method is called, no such notifications are reported anymore.

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

6.1.9 Method <<new>> getNextNotification()

This method is used by the application to query the event criteria set with createNotification or changeNotification. Since a lot of data can potentially be returned (which might cause problem in the middleware), this method must be used in an iterative way. Each method invocation may return part of the total set of notifications if the set is too large to return it at once. The reset parameter permits the application to indicate whether an invocation to getNextNotification is requesting more notifications from the total set of notifications or is requesting that the total set of notifications shall be returned from the beginning.

Returns notificationRequestedSetEntry: The set of notifications and an indication whether all off the notifications have been obtained or if more notifications are available that have not yet been obtained by the application. If no notifications exist, an empty set is returned and the final indication shall be set to TRUE.

Note that the (maximum) number of items provided to the application is determined by the gateway.

Parameters

reset : in TpBoolean

TRUE: indicates that the application is intended to obtain the set of notifications starting at the beginning.

FALSE: indicates that the application requests the next set of notifications that have not (yet) been obtained since the last call to this method with this parameter set to TRUE.

The first time this method is invoked, reset shall be set to TRUE. Following the receipt of a final indication in TpNotificationRequestedSetEntry, for the next call to this method reset shall be set to TRUE. P_TASK_REFUSED may be thrown if these conditions are not met.

Returns

TpNotificationRequestedSetEntry

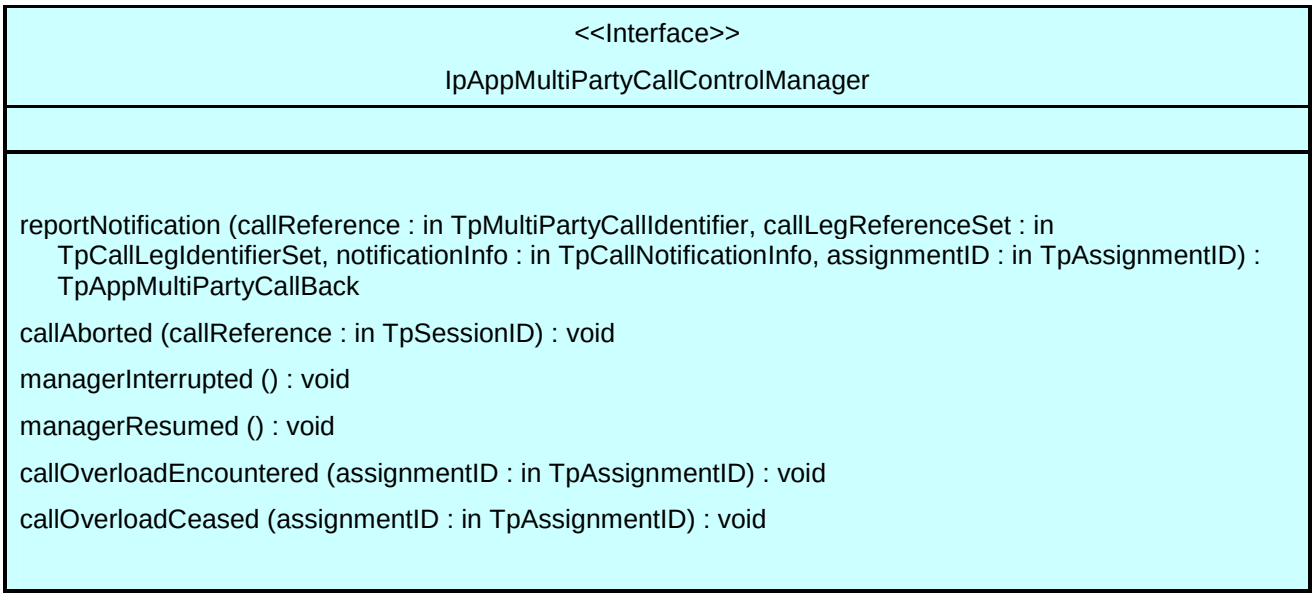
Raises

TpCommonExceptions

6.2 Interface Class IpAppMultiPartyCallControlManager

Inherits from: IpInterface

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



6.2.1 Method reportNotification()

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

If this method is invoked with a monitor mode of P_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 P_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 reportNotification() is invoked with a monitor mode of P_CALL_MONITOR_MODE_INTERRUPT, the application writer should ensure that no continue processing e.g. createAndRouteCallLegReq() is performed until the callback interface for the new call and/or new call leg has been passed to the gateway, either through an explicit setCallbackWithSessionID() invocation, or via the return of the reportNotification() method.

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

Case a:

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

Case b:

The reportNotification() with no callback reference ("Null" value) is used where (e.g. due to distributed application logic) the call back 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 reportNotification() 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().

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

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

Parameters

callReference : in TpMultiPartyCallIdentifier

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

callLegReferenceSet : in TpCallLegIdentifierSet

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

However, if the notification is being given in NOTIFY mode, this parameter shall be ignored by the application client implementation, and consequently the implementation of the SCS entity invoking reportNotification may populate this parameter as it chooses.

notificationInfo : in TpCallNotificationInfo

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

assignmentID : in TpAssignmentID

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

Returns

TpAppMultiPartyCallBack

6.2.2 Method callAborted()

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

Parameters

callReference : in TpSessionID

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

6.2.3 Method managerInterrupted()

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

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

Parameters

No Parameters were identified for this method

6.2.4 Method managerResumed()

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

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 addressrange 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 addressrange for within which the overload has been ceased.

6.3 Interface Class IpMultiPartyCall

Inherits from: IpService

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

This interface shall be implemented by a Multi Party Call Control SCF. The release() and deassignCall() methods, and either the createCallLeg() or the createAndRouteCallLegReq(), shall be implemented as a minimum requirement.

<<Interface>>

IpMultiPartyCall

getCallLegs (callSessionID : in TpSessionID) : TpCallLegIdentifierSet

createCallLeg (callSessionID : in TpSessionID, appCallLeg : in IpAppCallLegRef) : TpCallLegIdentifier

createAndRouteCallLegReq (callSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet, targetAddress : in TpAddress, originatingAddress : in TpAddress, appInfo : in TpCallAppInfoSet, appLegInterface : in IpAppCallLegRef) : TpCallLegIdentifier

release (callSessionID : in TpSessionID, cause : in TpReleaseCause) : void

deassignCall (callSessionID : in TpSessionID) : void

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

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

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

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

6.3.1 Method getCallLegs()

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

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

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

Returns

TpCallLegIdentifierSet

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.3.2 Method createCallLeg()

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

Returns callLeg: Specifies the interface and sessionID of the call leg created.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

appCallLeg : in IpAppCallLegRef

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

Returns

TpCallLegIdentifier

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE

6.3.3 Method createAndRouteCallLegReq()

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

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

If the application wishes that the call leg should be represented in the network as being a redirection it should include a value for the field P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS of TpCallAppInfo.

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

Note that for application initiated calls in some networks the result of the first createAndRouteCallLegReq() has to be received before the next createAndRouteCallLegReq() can be invoked. The Service Property P_PARALLEL_INITIAL_ROUTING_REQUESTS (see clause 8.1) indicates how a specific implementation handles the initial createAndRouteCallLegReq(). This method shall throw P_TASK_REFUSED if an application is not allowed to use parallel routing requests.

Returns callLegReference: Specifies the reference to the CallLeg interface that was created.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

eventsRequested : in TpCallEventRequestSet

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

targetAddress : in TpAddress

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

originatingAddress : in TpAddress

Specifies the address of the originating (calling) party.

appInfo : in TpCallAppInfoSet

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

appLegInterface : in IpAppCallLegRef

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

Returns

TpCallLegIdentifier

Raises

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

6.3.4 Method release()

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

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

cause : in TpReleaseCause

Specifies the cause of the release.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

6.3.5 Method deassignCall()

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

When this method is invoked, all outstanding supervision requests will be cancelled.

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.3.6 Method getInfoReq()

This asynchronous method requests information associated with the call to be provided at the appropriate time (for example, to calculate charging). This method must be invoked before the call is routed to a target address.

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

6.3.7 Method setChargePlan()

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

6.3.8 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, P_INVALID_CURRENCY, P_INVALID_AMOUNT

6.3.9 Method superviseReq()

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

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

time : in TpDuration

Specifies the granted time in milliseconds for the connection. Measurement will start as soon as the call is connected in the network, e.g. answered by the B-party or the user-interaction system.

treatment : in TpCallSuperviseTreatment

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

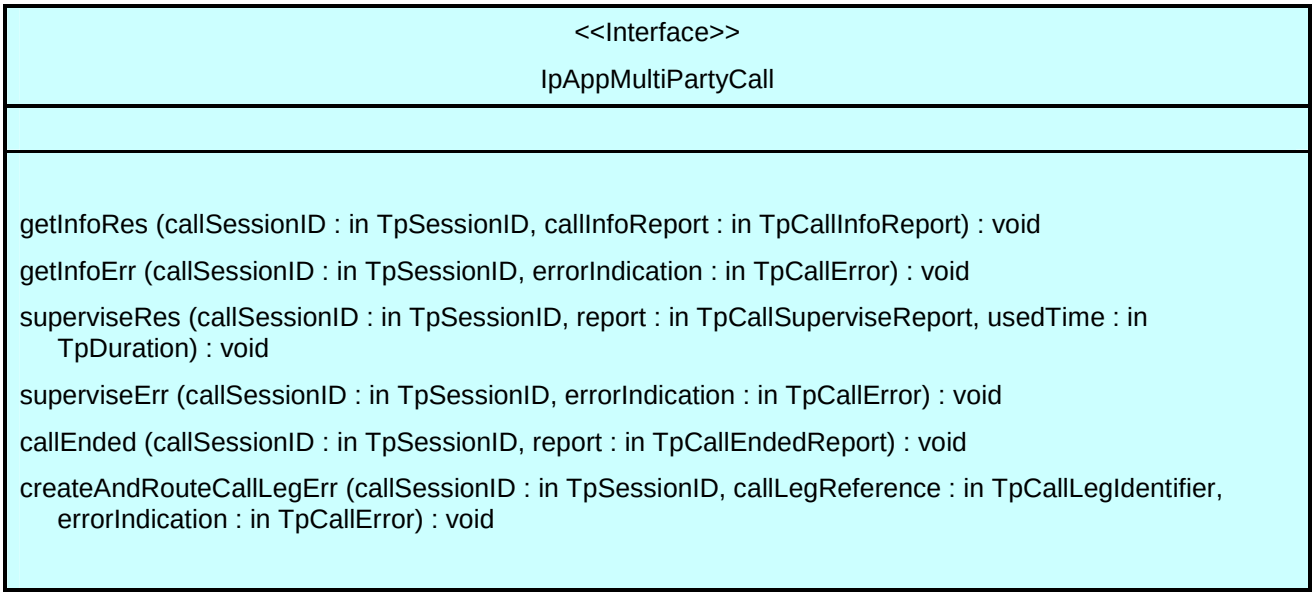
Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.4 Interface Class IpAppMultiPartyCall

Inherits from: IpInterface

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



6.4.1 Method getInfoRes()

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

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`callInfoReport : in TpCallInfoReport`

Specifies the call information requested.

6.4.2 Method getInfoErr()

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

Parameters

`callSessionID : in TpSessionID`

Specifies the call session ID of the call.

`errorIndication : in TpCallError`

Specifies the error which led to the original request failing.

6.4.3 Method superviseRes()

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).

6.4.4 Method superviseErr()

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

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.4.5 Method callEnded()

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

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

Parameters

callSessionID : in TpSessionID

Specifies the call sessionID.

report : in TpCallEndedReport

Specifies the reason the call is terminated.

6.4.6 Method createAndRouteCallLegErr()

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

Parameters

callSessionID : in TpSessionID

Specifies the call session ID of the call.

callLegReference : in TpCallLegIdentifier

Specifies the reference to the CallLeg interface that was created.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.5 Interface Class IpCallLeg

Inherits from: IpService

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

This interface shall be implemented by a Multi Party Call Control SCF. The routeReq(), eventReportReq(), release(), continueProcessing() and deassign() methods shall be implemented as a minimum requirement.

<<Interface>> IpCallLeg
routeReq (callLegSessionID : in TpSessionID, targetAddress : in TpAddress, originatingAddress : in TpAddress, appInfo : in TpCallAppInfoSet, connectionProperties : in TpCallLegConnectionProperties) : void eventReportReq (callLegSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet) : void release (callLegSessionID : in TpSessionID, cause : in TpReleaseCause) : void getInfoReq (callLegSessionID : in TpSessionID, callLegInfoRequested : in TpCallLegInfoType) : void getCall (callLegSessionID : in TpSessionID) : TpMultiPartyCallIdentifier attachMediaReq (callLegSessionID : in TpSessionID) : void detachMediaReq (callLegSessionID : in TpSessionID) : void getCurrentDestinationAddress (callLegSessionID : in TpSessionID) : TpAddress continueProcessing (callLegSessionID : in TpSessionID) : void setChargePlan (callLegSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : void setAdviceOfCharge (callLegSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tariffSwitch : in TpDuration) : void superviseReq (callLegSessionID : in TpSessionID, time : in TpDuration, treatment : in TpCallLegSuperviseTreatment) : void deassign (callLegSessionID : in TpSessionID) : void

6.5.1 Method routeReq()

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

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

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

If the application wishes that the call leg should be represented in the network as being a redirection it should include a value for the field P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS of TpCallAppInfo.

This operation continues processing of the call leg.

Note that for application initiated calls in some networks the result of the first routeReq() has to be received before the next routeReq() can be invoked. The Service Property P_PARALLEL_INITIAL_ROUTING_REQUESTS (see clause 8.1) indicates how a specific implementation handles the initial routeReq().This method shall throw P_TASK_REFUSED if an application is not allowed to use parallel routing requests.

Parameters

`callLegSessionID : in TpSessionID`

Specifies the call leg session ID of the call leg.

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.

appInfo : in TpCallAppInfoSet

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

connectionProperties : in TpCallLegConnectionProperties

Specifies the properties of the connection.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE, P_INVALID_ADDRESS, P_UNSUPPORTED_ADDRESS_PLAN

6.5.2 Method eventReportReq()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

eventsRequested : in TpCallEventRequestSet

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

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_EVENT_TYPE, P_INVALID_CRITERIA

6.5.3 Method release()

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

This operation continues processing of the call leg.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

cause : in TpReleaseCause

Specifies the cause of the release.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

6.5.4 Method getInfoReq()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

callLegInfoRequested : in TpCallLegInfoType

Specifies the call leg information that is requested.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.5.5 Method getCall()

This method requests the call associated with this call leg.

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

Returns

TpMultiPartyCallIdentifier

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.5.6 Method attachMediaReq()

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

In case this method is invoked while there is still a request to detach the Media pending, the exception "P_TASK_REFUSED" will be raised.

Parameters

callLegSessionID : in TpSessionID

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

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

6.5.7 Method detachMediaReq()

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

In case this method is invoked while there is still a request to attach the Media pending, the exception "P_TASK_REFUSED" will be raised.

Parameters

callLegSessionID : in TpSessionID

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

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

6.5.8 Method getCurrentDestinationAddress()

Queries the current address of the destination the leg has been directed to.

Returns the address of the destination point towards which the call leg has been routed.

If this method is invoked on the Originating Call Leg, exception P_INVALID_STATE will be thrown.

Parameters

callLegSessionID : in TpSessionID

Specifies the call session ID of the call leg.

Returns

TpAddress

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.5.9 Method continueProcessing()

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

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

6.5.10 Method setChargePlan()

Set an operator specific charge plan for the call leg.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

callChargePlan : in TpCallChargePlan

Specifies the charge plan to use.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.5.11 Method setAdviceOfCharge()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

aOCInfo : in TpAoCInfo

Specifies two sets of Advice of Charge parameter.

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, P_INVALID_CURRENCY, P_INVALID_AMOUNT

6.5.12 Method superviseReq()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call party.

time : in TpDuration

Specifies the granted time in milliseconds for the connection. Measurement will start as soon as the callLeg is connected in the network.

treatment : in TpCallLegSuperviseTreatment

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

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.5.13 Method deassign()

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

The application should not release or deassign the call leg when received a callLegEnded() or callEnded(). This operation continues processing of the call leg.

When this method is invoked, all outstanding supervision requests will be cancelled.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

6.6 Interface Class IpAppCallLeg

Inherits from: IpInterface

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

<<Interface>>

IpAppCallLeg

eventReportRes (callLegSessionID : in TpSessionID, eventInfo : in TpCallEventInfo) : void

eventReportErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void

attachMediaRes (callLegSessionID : in TpSessionID) : void

attachMediaErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void

detachMediaRes (callLegSessionID : in TpSessionID) : void

detachMediaErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void

getInfoRes (callLegSessionID : in TpSessionID, callLegInfoReport : in TpCallLegInfoReport) : void

getInfoErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void

routeErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void

superviseRes (callLegSessionID : in TpSessionID, report : in TpCallSuperviseReport, usedTime : in TpDuration) : void

superviseErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void

callLegEnded (callLegSessionID : in TpSessionID, cause : in TpReleaseCause) : void

6.6.1 Method eventReportRes()

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

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

If this method is invoked for a report with a monitor mode of P_CALL_MONITOR_MODE_INTERRUPT, then the application has control of the call leg. If the application does nothing with the call leg within a specified time period (the duration which forms a part of the service level agreement), then the connection in the network shall be released and callLegEnded() shall be invoked, giving a release cause of P_TIMER_EXPIRY.

Parameters

`callLegSessionID : in TpSessionID`

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

`eventInfo : in TpCallEventInfo`

Specifies data associated with this event.

6.6.2 Method eventReportErr()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.6.3 Method attachMediaRes()

This asynchronous method reports the attachment of a call leg to a call has succeeded. The media channels or bearer connections to this leg is now available.

Parameters

callLegSessionID : in TpSessionID

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

6.6.4 Method attachMediaErr()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.6.5 Method detachMediaRes()

This asynchronous method reports the detachment of a call leg from a call has succeeded. The media channels or bearer connections to this leg is no longer available.

Parameters

callLegSessionID : in TpSessionID

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

6.6.6 Method detachMediaErr()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.6.7 Method getInfoRes()

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

Parameters

callLegSessionID : in TpSessionID

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

callLegInfoReport : in TpCallLegInfoReport

Specifies the call leg information requested.

6.6.8 Method getInfoErr()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.6.9 Method routeErr()

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.6.10 Method superviseRes()

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

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

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

report : in TpCallSuperviseReport

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

usedTime : in TpDuration

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

6.6.11 Method superviseErr()

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

errorIndication : in TpCallError

Specifies the error which led to the original request failing.

6.6.12 Method callLegEnded()

This method indicates to the application that the leg has terminated in the network. The application has received all requested results (e.g. getInfoRes) related to the call leg. The call leg will be destroyed after returning from this method.

Parameters

callLegSessionID : in TpSessionID

Specifies the call leg session ID of the call leg.

cause : in TpReleaseCause

Specifies the reason the connection is terminated.

7 MultiParty Call Control Service State Transition Diagrams

7.1 State Transition Diagrams for IpMultiPartyCallControlManager

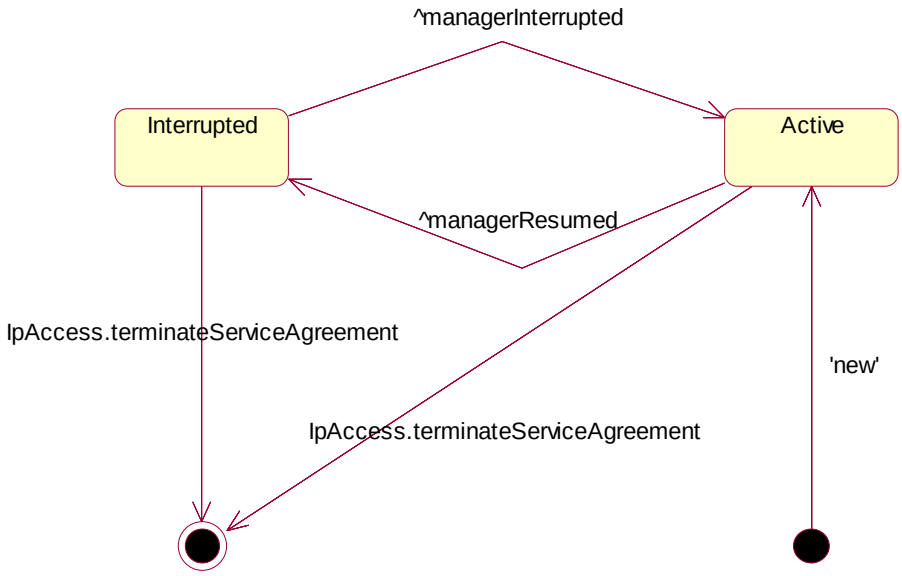


Figure : Application view and the Multi-Party Call Control Manager

7.1.1 Active State

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

7.1.2 Interrupted State

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

7.1.3 Overview of allowed methods

Call Control Manager State	Methods applicable
Active	createCall, createNotification, destroyNotification, changeNotification, getNotification, getNextNotification, setCallLoadControl, enableNotifications, disableNotifications
Interrupted	getNotification, getNextNotification, enableNotifications, disableNotifications

7.2 State Transition Diagrams for IpMultiPartyCall

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

When an IpMultiPartyCall is created using createCall, or when an IpMultiPartyCall is given to the application for a notification with a monitor mode of P_CALL_MONITOR_MODE_INTERRUPT, an activity timer is started. The activity timer is stopped when the application invokes a method on the IpMultiPartyCall. The action upon expiry of this activity timer is to invoke callEnded() on the IpAppMultiPartyCall with a release cause of P_TIMER_EXPIRY. In the case when no IpAppMultiPartyCall is available on which to invoke callEnded(), callAborted() shall be invoked on the IpAppMultiPartyCallControlManager as this is an abnormal termination.

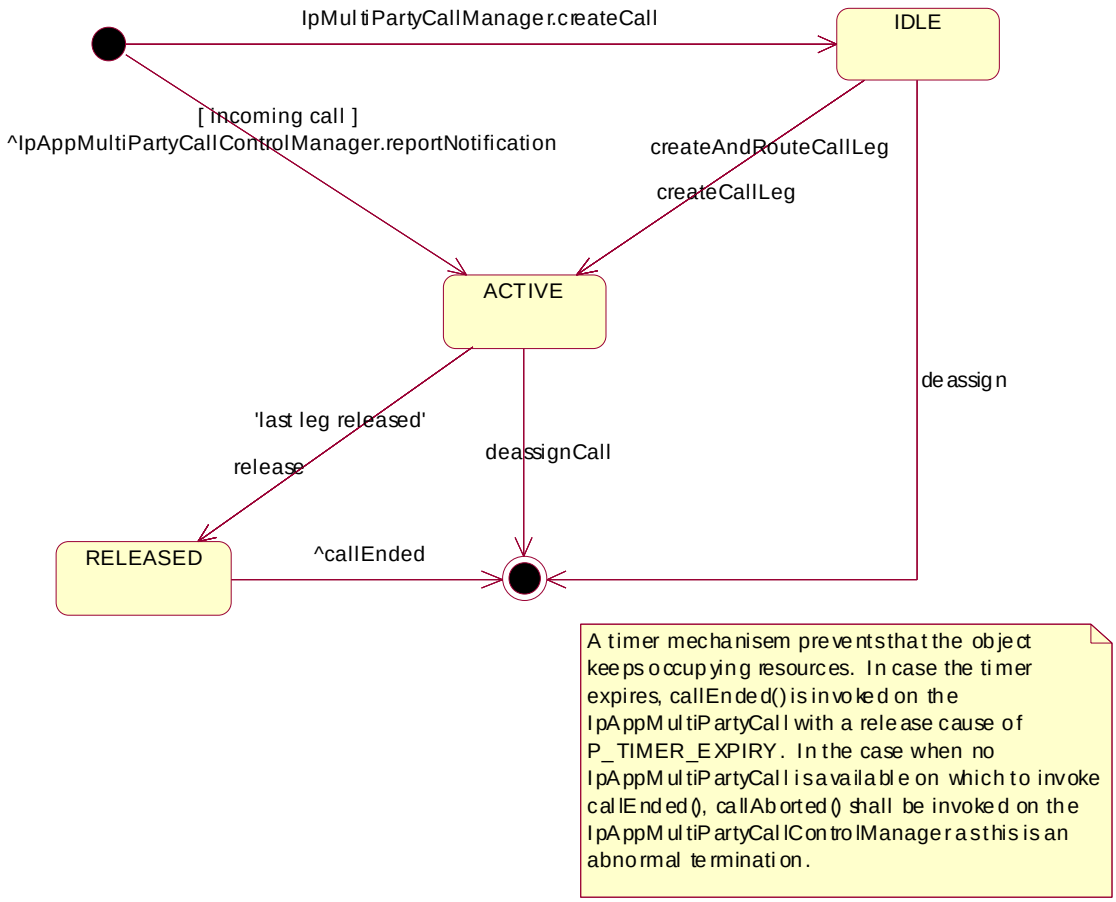


Figure : Application view on the MultiParty Call object

7.2.1 IDLE State

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

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

7.2.2 ACTIVE State

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

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

7.2.3 RELEASED State

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

7.2.4 Overview of allowed methods

Methods applicable	Call Control Call State	Call Control Manager State	
getCallLegs	Idle, Active, Released	-	
createCallLeg, createAndRouteCallLegReq, setAdviceOfCharge, superviseReq,	Idle, Active	Active	
release	Active	Active	
deassignCall	Idle, Active	-	
setChargePlan, getInfoReq	Idle, Active	Active	

7.3 State Transition Diagrams for IpCallLeg

The IpCallLeg State Transition Diagram is divided in two State Transition Diagrams, one for the originating call leg and one for the terminating call leg.

Call Leg State Model General Objectives:

1) Events in backwards direction (upstream), coming from terminating leg, are not directly visible in originating leg model. NOTE1

2) Events in forwards direction (downstream), coming from originating leg, are not directly visible in terminating leg model. NOTE1

3) States are as seen from the application: if there is no change in the method an application is permitted to apply on the IpCallLeg object, then there is no state change. Therefore receipt of e.g. answer or alerting events on terminating leg do not change state. NOTE 2

4) Call processing is suspended if for a leg a network event is met, which was requested to be monitored in the P_CALL_MONITOR_MODE_INTERRUPT. The application shallsend a request to continue processing (using an appropriate method like continueProcessing, deassign, release or routeReq) for each leg and event reported in monitor mode 'interrupt'.
   If the event leads to a state transition, the call processing is suspended when entering the state.

5) In case on a leg more than one network event (for example a mid-call event 'service_code' and a disconnection event) is to be reported to the application at quasi the same time, then the events are to be reported one by one to the application in the order received from the network. When for a leg an event is reported in interrupt mode, a next pending event is not to be reported to the application until a request to resume call processing for the current reported event has been received on the leg.

NOTE 1: Although events coming from a specific party will always be tied to the callLeg related to that party, these events might lead to state transitions of other callLegs. Examples of such events are terminating release, where also the originating leg might transit to the releasing state and originating_release where the terminating leg might transit to the releasing state.

NOTE 2: Even though there in the Originating Call Leg STD is no change in the methods the application is permitted to apply to the IpCallLeg object for the states Analysing and Active, separate states are maintained. The states may therefore from an application viewpoint appear as just one state that may be have substates like Analysing and Active. The digit collection task in state Analysing state may be viewed as a specialised task that may not at all be applicable in some networks and therefore here described as being a state on its own.

7.3.1 Originating Call Leg

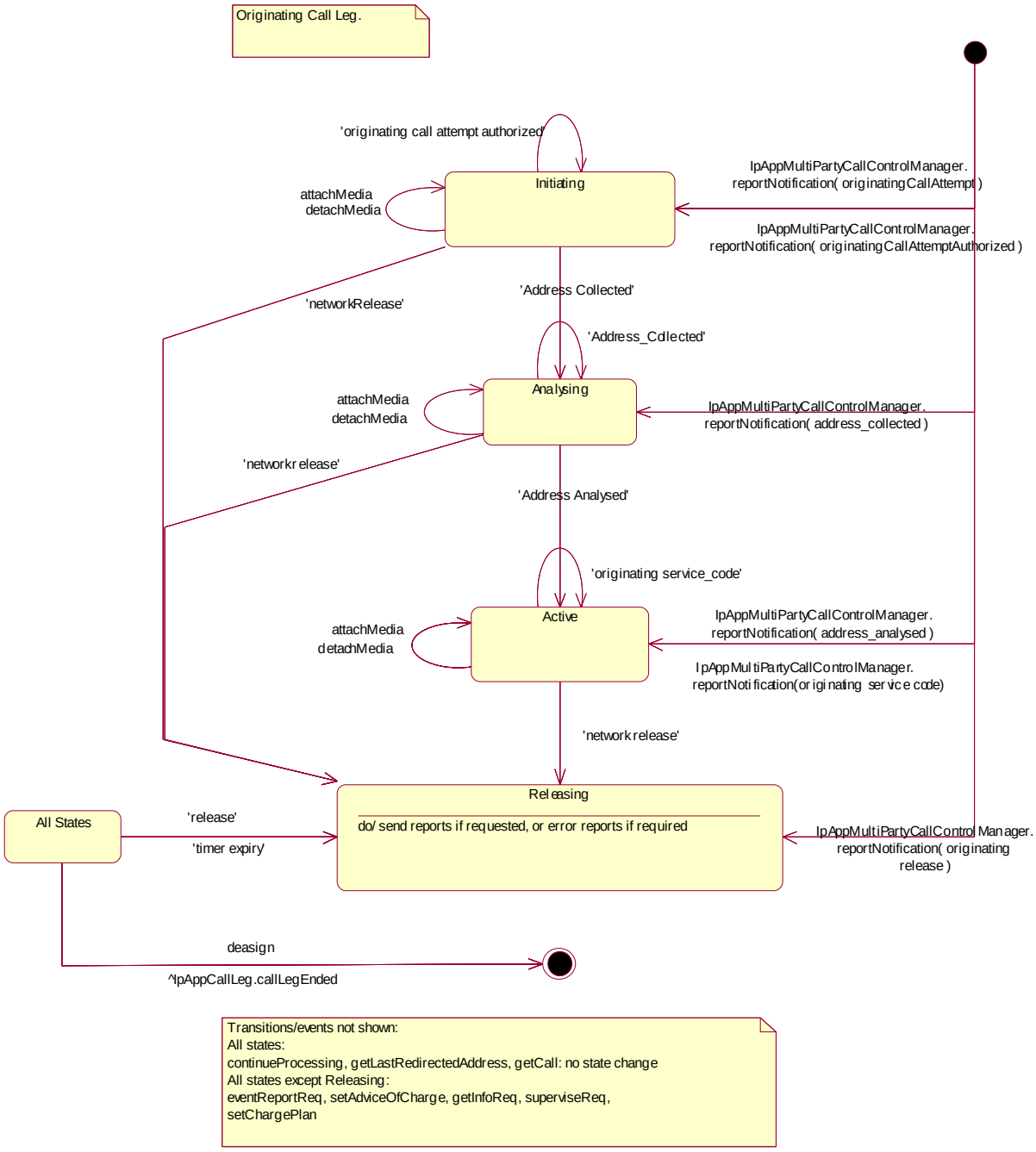


Figure : Originating Leg

7.3.1.1 Initiating State

Entry events:

- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Originating_Call_Attempt" initial notification criterion.

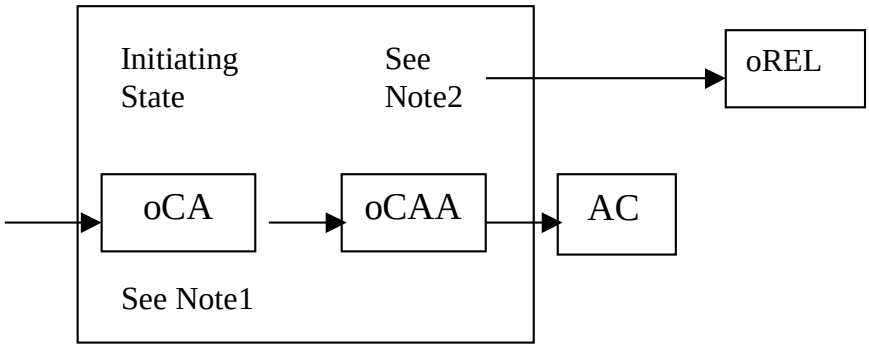
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Originating_Call_Attempt_Authorised" initial notification criterion.

Functions:

In this state the network checks the authority/ability of the party to place the connection to the remote (destination) party with the given properties, e.g. based on the originating party's identity and service profile.

The setup of the connection for the party has been initiated and the application activity timer is being provided.

The figure below shows the order in which network events may be detected in the Initiating state and depending on the monitor mode be reported to the application.



Note 1: Event oCA only applicable as an initial notification .
Note 2: The release event (oREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:
oCA: originating Call Attempt;
oCAA originating Call Attempt Authorized;
AC: Address Collected;
oREL originating RELease.

Figure : Application view on event reporting order in Initiating State

In this state the following functions are applicable:

- The detection of an "Originating_Call_Attempt" initial notification criterion.

- The detection of an "Originating_Call_Attempt_Authorised" initial notification criterion as a result that the call attempt authorisation is successful.

- The report of the "Originating_Call_Attempt_Authorised" event indication whereby the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_CALL_ATTEMPT_AUTHORISED then the event is reported and call leg processing is suspended.

  ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_CALL_ATTEMPT_AUTHORISED then the event is notified and call leg processing continues.

  iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_CALL_ATTEMPT_AUTHORISED then no monitoring is performed.

- The receipt of destination address information, i.e. initial information package/dialling string as received from calling party.

- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.

Exit events:

- Availability of destination address information, i.e. the initial information package/dialling string received from the calling party.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period while processing is suspended for the leg.
- Receipt of a deassign() method.
- Receipt of a release() method.
- Detection of a "originating release" indication as a result of a premature disconnect from the calling party.

7.3.1.2 Analysing State

Entry events:

- Availability of an "Address_Collected" event indication as a result of the receipt of the (complete) initial information package/dialling string from the calling party.
- Availability of an "Address_Collected" event indication as a result of additional digits received from the calling party as requested by the application (with eventReportReq).
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Address_Collected" initial notification criterion.

Functions:

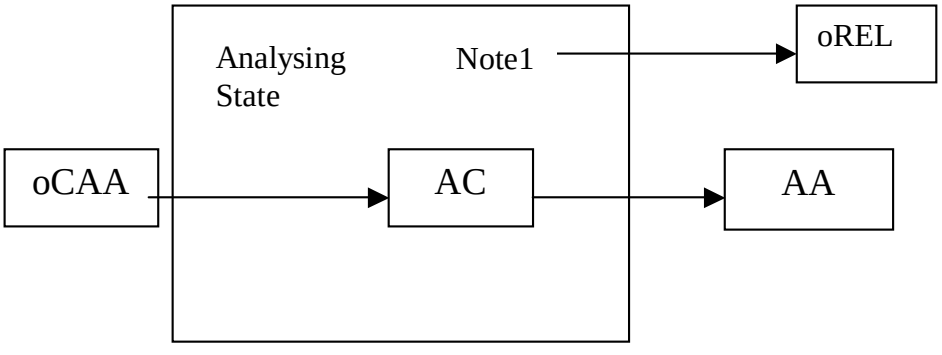
In this state the destination address provided by the calling party is collected and analysed.

The received information (dialled address string from the calling party) is being collected and examined in accordance to the dialling plan in order to determine end of address information (digit) collection. Additional address digits can be collected. Upon completion of address collection the address is analysed.

The address analysis is being made according to the dialling plan in force to determine the routing address of the call leg connection and the connection type (e.g. local, transit, gateway).

The request (with eventReportReq method) to collect a variable number of more address digits and report them to the application (within eventReportRes method) is handled within this state. The collection of more digits as requested and the reporting of received digits to the application (when the digit collect criteria is met) is done in this state. This action can be repeated, e.g. the application may request first for 3 digits to be collected and when reported request further digits.

The figure below shows the order in which network events may be detected in the Analysing state and depending on the monitor mode be reported to the application.



Note 1: The release event (oREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:
oCAA: originating Call Attempt Authorized;
AC: Address Collected;
AA: Address Analysed;
oREL: originating RELease.

Figure : Application view on event reporting order in Analysing State

In this state the following functions are applicable:

- The detection of an "Address_Collected" initial notification criterion.
- On receipt of the "Address_Collected" indication the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_ADDRESS_COLLECTED then the event is reported and call leg processing is suspended.

  ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_ADDRESS_COLLECTED then the event is notified and call leg processing continues.

  iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_ADDRESS_COLLECTED then no monitoring is performed.
- Receipt of an eventReportReq() method defining the criteria for the events the call leg object is to observe.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() or a routeReq() method.

Exit events:

- Detection of an "Address_Analysed" indication as a result of the availability of the routing address and nature of address.
- Receipt of a deassign() method.
- Receipt of a release() method.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period while processing is suspended for the leg.
- Detection of a "originating release" indication as a result of a premature disconnect from the calling party.

7.3.1.3 Active State

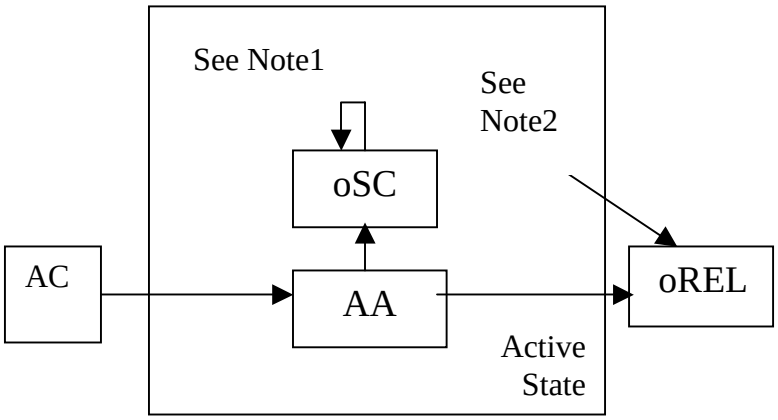
Entry events:

- Receipt of an "Address_Analysed" indication as a result of the availability of the routing address and nature of address.
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Address_Analysed initial indication criterion.

Functions:

In this state the call leg connection to the calling party exists and originating mid call events can be received.

The figure below shows the order in which network events may be detected in the Active state and depending on the monitor mode be reported to the application.



Note 1: Only the detected service code or the range to which the service code belongs is disarmed as the service code is reported to the application.
Note 2: The release event (oREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:
AC: Address Collected;
AA: Address Analysed;
oSC: originating Service Code;
oREL: originating RELease.

Figure : Application view on event reporting order Active State

In this state the following functions are applicable:

- The detection of an Address_Analysed initial indication criterion.
- On receipt of the "Address_Analysed" indication the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_ADDRESS_ANALYSED then the event is reported and call leg processing is suspended.

  ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_ADDRESS_ANALYSED then the event is notified and call leg processing continues.

  iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_ADDRESS_ANALYSED then no monitoring is performed.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.
- When entering this state the routing information is interpreted, the authority of the calling party to establish this connection is verified. Note that no call leg connection is set up to the remote party at this point when the application is still in control. The application explicitly has to create and route the terminating leg, optionally using the address information from the Address_Analysed event. Only in case the call is deassigned (the application relinquishes control) in this state, the network will setup the connection to terminating leg automatically based on the received information.
- In this state a connection to the calling party is established.
- On receipt of the "originating_service code" indication the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_ORIGINATING_SERVICE_CODE then the event is reported and call leg processing is suspended.

ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_ORIGINATING_SERVICE_CODED then the event is notified and call leg processing continues.

iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_ORIGINATING_SERVICE_CODE then no monitoring is performed.

- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.

Exit events:

- Detection of an "originating release" indication as a result of a disconnect from the calling party.
- Detection of a propagated disconnect from the called party
- Receipt of a deassign() method.
- Receipt of a release() method from the application.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period while call processing is suspended.

7.3.1.4 Releasing State

Entry events:

- Detection of an "Originating_Release" indication as a result of the network release initiated by calling party.
- Propagated release from called party.
- Release of the entire call (e.g., after invoking IpCall.release()).
- Reception of the release() method from the application.
- A transition due to fault detection to this state is made when the Call leg object is in a state and no requests from the application have been received during a certain time period (timer expiry).

Functions:

In this state the connection to the call party is released as requested by the network or by the application and the reports are processed and sent to the application if requested.

When the Releasing state is entered the order of actions to be performed is as follows:

i) the network release event handling is performed,

ii) the possible call leg information requested with getInfoReq() and/ or superviseReq() is collected and send to the application,

iii) the callLegEnded() method is sent to the application to inform that the call leg object is destroyed.

In this state the following functions are applicable:

- The detection of an "originating_release" initial indication criterion..
- On receipt of the "originating_release" indication the following functions are performed:
  - The network release event handling is performed as follows:

    i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_RELEASE then the event is reported and call leg processing is suspended.

    ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_RELEASE then the event is notified and call leg processing continues.

iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_RELEASE then no monitoring is performed.

Note that this handling is not performed for propagated releases from the called party.

- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.
- The possible call leg information requested with the getInfoReq() and/or superviseReq() is collected and sent to the application with respectively the getInfoRes() and/or superviseRes() methods.
- The callLegEnded() method is sent to the application after all information has been sent. In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application will also be informed that the connection has ended
- In case of abnormal termination due to a fault and the application requested for call leg related information previously, the application will be informed that this information is not available and additionally the application is informed that the call leg object is destroyed (callLegEnded) and the leg is released in the network.

Note: the call in the network may continue or be released, depending e.g. on the call state.

- In case the release() method is received in Releasing state it will be discarded. The request from the application to release the leg is ignored in this case because release of the leg is already ongoing.

Exit events:

- In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.
- After the sending of the last call leg information to the application the Call Leg object is destroyed and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period while processing is suspended for the leg (re-enter releasing state).
- Receipt of a deassign() method. The leg will be released and call leg object destroyed, but no reports will be sent to the application anymore. Also no CallLegEnded will be invoked.

7.3.1.5 Overview of allowed methods, Originating Call Leg STD

State	Methods allowed
Initiating	attachMediaReq (as a request), detachMediaReq, (as a request) getCall, continueProcessing, release (call leg), deassign eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Analysing	attachMediaReq (as a request), detachMediaReq, (as a request) getCall , continueProcessing, release (call leg), deassign, eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Active	attachMediaReq, detachMediaReq, getCall, continueProcessing, release, deassign, eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Releasing	getCall, continueProcessing, release, deassign

7.3.2 Terminating Call Leg

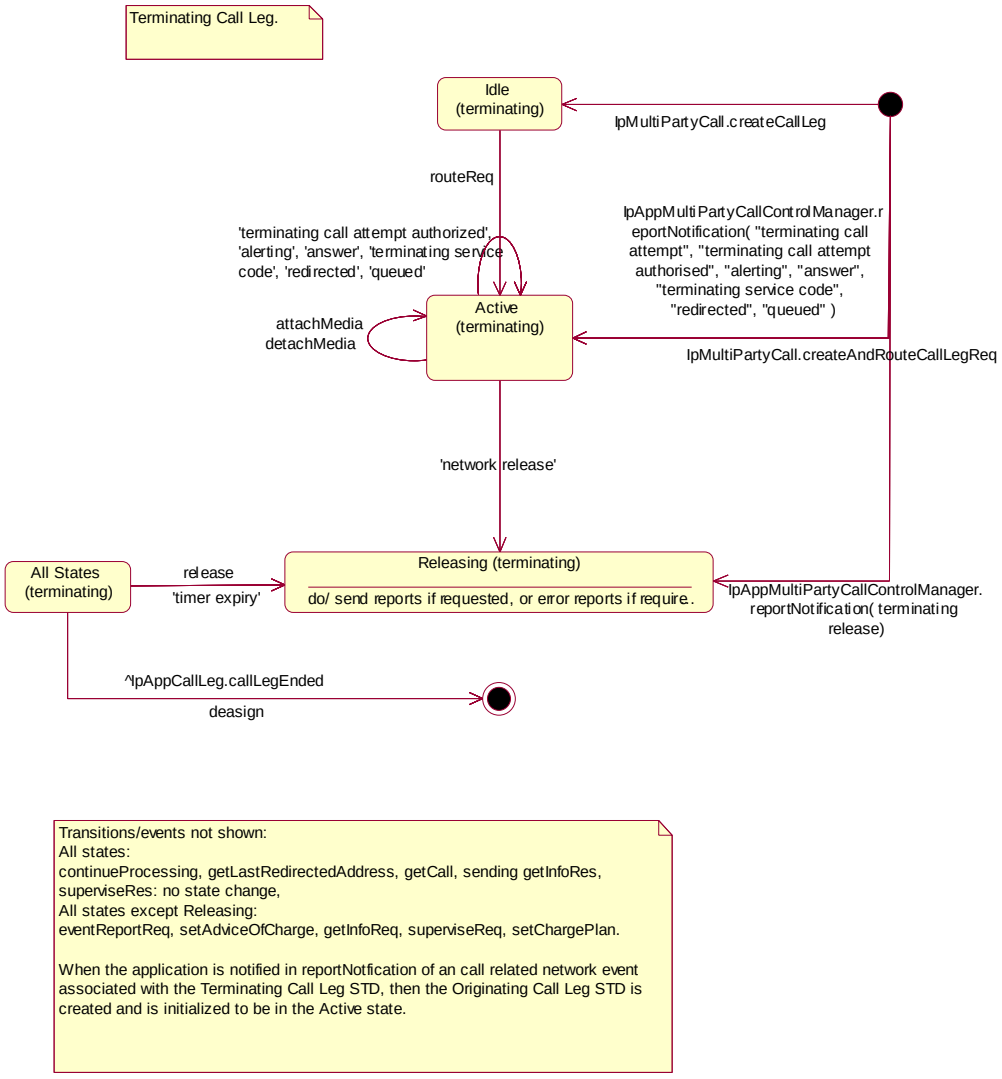


Figure : Terminating Leg

7.3.2.1 Idle (terminating) State

Entry events:

- Receipt of a createCallLeg() method to start an application initiated call leg connection.

Functions:

In this state the call leg object is created and the interface connection is idled.

The application activity timer is being provided.

In this state the following functions are applicable:

- Invoking routeReq will result in a request to actually route the call leg object and resumption of call processing.

Exit events:

- Receipt of a routeReq() method from the application.

- Application activity timer expiry indicating that no requests from the application have been received during a certain period to continue processing.
- Receipt of a deassign() method.
- Receipt of a release() method.
- Propagation ofa network release event as a result of a disconnect from the calling party.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period while processing is suspended for the leg.

7.3.2.2 Active (terminating) State

Entry events:

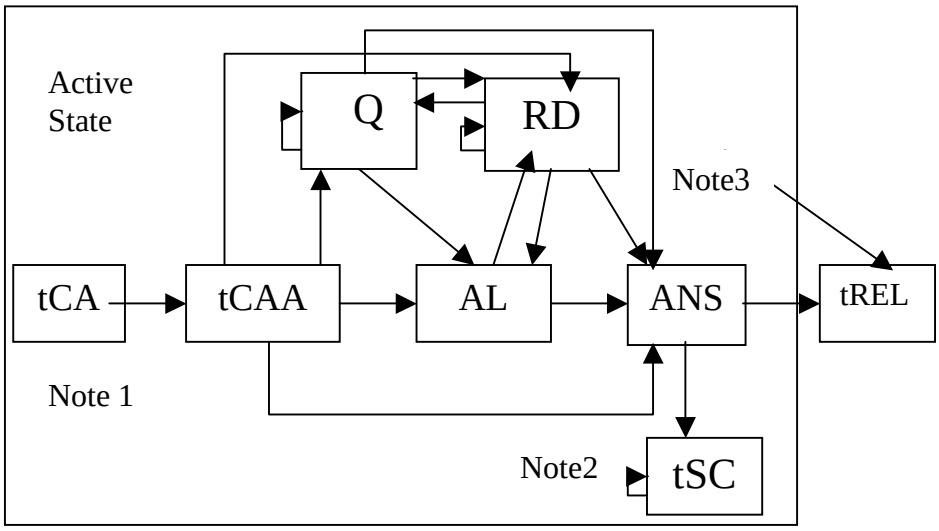
- Receipt of a routeReq will result in actually routing the call leg object.
- Receipt of a createAndRouteCallLegReq() method to start an application initiated call leg connection.
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for the following trigger criteria: "Terminating_Call_Attempt", "Terminating_Call_Attempt_Authorised", "Alerting", "Answer", "Terminating service code", "Redirected" and "Queued".

Functions:

In this state the routing information is interpreted, the authority of the called party to establish this connection is verified for the call leg connection. In this state a connection to the call party is established whereby events from the network may indicate to the application when the party is alerted (acknowledge connection setup) and when the party answer (confirmation of connection setup).

Furthermore, in this state terminating service code events can be received.

The figure below shows the order in which network events may be detected in the Active state and depending on the monitor mode be reported to the application.



Note 1: Event tCA applicable as initial notification.
Note 2: Only the detected service code or the range to which the service code belongs is disarmed as the service code is reported to the application.
Note 3: The release event (tREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:
tCA: Terminating Call Attempt;
tCAA: terminating Call Attempt Authorized;
AL: Alerting;
ANS: Answer;
tREL: terminating RELease;
Q: Queued;
RD: ReDirected;
tSC: terminating Service Code.

Figure : Application view on event reporting order in Active State

In this state the following functions are applicable:

- The detection and report of the "Terminating_Call_Attempt_Authorised" event indication whereby the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED then the event is reported and call leg processing is suspended.

  ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED then the event is notified and call leg processing continues.

  iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_CALL_TERMINATING_ATTEMPT_AUTHORISED then no monitoring is performed.

- Detection of an "Queued" indication as a result of the terminating call being queued.

- On receipt of the "Queued" indication the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_QUEUED then the event is reported and call leg processing is suspended.

  ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_QUEUED then the event is notified and call leg processing continues.

  iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_QUEUED then no monitoring is performed.

- On receipt of the "Alerting" indication the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_ALERTING then the event is reported and call leg processing is suspended.

  ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_ALERTING then the event is notified and call leg processing continues.

  iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_ALERTING then no monitoring is performed.

- Detection of an "Answer" indication as a result of the remote party being connected (answered).

- On receipt of the "Answer" indication the following functions are performed:

  i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_ANSWER then the event is reported and call leg processing is suspended.

ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_ANSWER then the event is notified and call leg processing continues.

iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_ANSWER then no monitoring is performed.

- The detection of a "service_code" trigger criterion suspends call leg processing.
- On receipt of the "service code" indication the following functions are performed:

i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_TERMINATING_SERVICE_CODE then the event is reported and call leg processing is suspended.

ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_TERMINATING_SERVICE_CODE then this is not a valid event (that event is not notified) and call leg processing continues.

iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_TERMINATING_SERVICE_CODE then no monitoring is performed.

- On receipt of the "redirected" indication the following functions are performed:

i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_REDIRECTED then the event is reported and call leg processing is suspended.

ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_REDIRECTED then the event is notified and call leg processing continues.

iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_REDIRECTED then no monitoring is performed.

- Resumption of call leg processing occurs on receipt of a continueProcessing() method.

Exit events:

- Detection of a network release event being a "terminating release" indication as a result of the following events:

i) Unable to select a route or indication from the remote party of the call leg connection cannot be presented (this is the network determined busy condition).

ii) Occurrence of an authorisation failure when the authority to place the call leg connection was denied (e.g. business group restriction mismatch).

iii) Detection of a route busy condition received from the remote call leg connection portion.

iv) Detection of a no-answer condition received from the remote call leg connection portion.

v) Detection that the remote party was not reachable.

- Propagation of a network release event as a result of the following events:
  - Detection of a premature disconnect from the calling party.
- Receipt of a deassign() method.
- Receipt of a release() method from the application.
- Propagation of network release event as a result of a disconnect from the calling party .
- Detection of a network release event being a "terminating release" indication as a result of a disconnect from the called party.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period while processing is suspended for the leg.

7.3.2.3 Releasing (terminating) State

Entry events:

- Propagation ofa network release disconnect from the calling party.
- Detection of a network release event being a "terminating release" indication as a result of the network release initiated by called party.
- Release of the entire call (e.g., after invoking IpCall.release()).
- Sending of the release() method by the application.
- A transition due to fault detection to this state is made when the Call leg object awaits a request from the application and this is not received within a certain time period.
- Detection of a network event being a "terminating release" indication as a result of the following events:

  i) Unable to select a route or indication from the remote party of the call leg connection cannot be presented (this is the network determined busy condition).

  ii) Occurrence of an authorisation failure when the authority to place the call leg connection was denied (e.g. business group restriction mismatch).

  iii) Detection of a route busy condition received from the remote call leg connection portion.

  iv) Detection of a no-answer condition received from the remote call leg connection portion.

  v) Detection that the remote party was not reachable.

- Propagation of a network release event as a result of the following events:
  - Detection of a premature disconnect from the calling party.

Functions:

In this state the connection to the call party is released as requested by the network or by the application and the reports are processed and sent to the application if requested .

When the Releasing state is entered the order of actions to be performed is as follows:

i) the release event handling is performed.

ii) the possible call leg information requested with getInfoReq() and/ or superviseReq() is collected and send to the application.

iii) the callLegEnded() method is sent to the application to inform that the call leg object is destroyed.

Where the entry to this state is caused by the application, for example because the application has requested the leg to be released or deassigned or a fault (e.g. timer expiry, no response from application) has been detected, then i) is not applicable. In the fault case for action ii) error report methods are sent to the application for any possible requested reports.

In this state the following functions are applicable:

- The detection of a "Terminating Release" trigger criterion.
- On receipt of the network release event being a "Terminating Release" indication the following functions are performed:
  - The network release event handling is performed as follows:

    i) When the P_CALL_MONITOR_MODE_INTERRUPT is requested for the call leg event P_CALL_EVENT_TERMINATING_RELEASE then the event is reported and call leg processing is suspended.

ii) When the P_CALL_MONITOR_MODE_NOTIFY is requested for the call leg event P_CALL_EVENT_TERMINATING_RELEASE then the event is notified and call leg processing continues.

iii) When the P_CALL_MONITOR_MODE_DO_NOT_MONITOR is requested for the call leg event P_CALL_EVENT_TERMINATING_RELEASE then no monitoring is performed.

Note that this handling is not performed for propagated releases from the calling party.

- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.
- The possible call leg information requested with the getInfoReq() and/or superviseReq() is collected and sent to the application with respectively the getInfoRes() and/or superviseRes() methods.
- The callLegEnded() method is sent to the application after all information has been sent. In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application will also be informed that the connection has ended
- In case of abnormal termination due to a fault and the application requested for call leg related information previously, the application will be informed that this information is not available and additionally the application is informed that the call leg object is destroyed (callLegEnded) and the leg is released in the network.

NOTE: The call in the network may continue or be released, depending e.g. on the call state.

- In case the release() method is received in Releasing state it will be discarded. The request from the application to release the leg is ignored in this case because release of the leg is already ongoing.

Exit events:

- In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.
- After the sending of the last call leg information to the application the Call Leg object is destroyed and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period while processing is suspended for the leg (re-enter releasing state).
- Receipt of a deassign() method. The leg will be released and call leg object destroyed, but no reports will be sent to the application anymore. Also no CallLegEnded will be invoked.

7.3.2.4 Overview of allowed methods and trigger events, Terminating Call Leg STD

State	Methods allowed
Idle	routeReq, getCall, getCurrentDestinationAddress, release, deassign, eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Active	attachMediaReq, detachMediaReq, getCall, getCurrentDestinationAddress, continueProcessing, release, deassign, eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Releasing	getCall, getCurrentDestinationAddress, continueProcessing, release, deassign

8 Multi-Party Call Control Service Properties

8.1 List of Service Properties

The following table lists properties relevant for the MPCC 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
P_MAX_CALLLEGS_PER_CALL	INTEGER_SET	Indicates the maximum number of legs in a call for which a connection to a call party exists in the network. The enforcement of this property is done only when a leg is created or routed by the application.
P_UI_CALLLEG_BASED	BOOLEAN_SET	Value = TRUE : User interaction can be performed on leg level and a reference to a CallLeg object can be used in the IpUIManager.createUICall() operation. Value = FALSE : No user interaction on leg level is supported.
P_PARALLEL_INITIAL_ROUTING_REQUESTS	BOOLEAN_SET	Indicates whether for application initiated calls it is possible to issue multiple routing request methods in parallel or that the application has to wait for the result of the first request before another one can be invoked. Value = TRUE: Multiple routing requests can be invoked in parallel. Value = FALSE: Result of first request has to be received before another request can be issued.

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

Property	Type	Description
P_TRIGGERING_ADDRESSES (Deprecated)	ADDRESSRANGE_SET	Indicates for which numbers the notification may be set. For terminating notifications it applies to the terminating number, for originating notifications it applies only to the originating number. See further explanation on which events are originating and which are terminating, below.
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_MONITOR_MODE	INTEGER_SET	Indicates whether the application is allowed to monitor in interrupt and/or notify mode. Set is: P_INTERRUPT P_NOTIFY
P_NUMBERS_TO_BE_CHANGED	INTEGER_SET	Indicates which numbers the application is allowed to change or fill for legs in an incoming call. Allowed value set: {P_ORIGINAL_CALLED_PARTY_NUMBER, P_REDIRECTING_NUMBER, P_TARGET_NUMBER, P_CALLING_PARTY_NUMBER}.
P_CHARGEPLAN_ALLOWED	INTEGER_SET	Indicates which charging is allowed in the setCallChargePlan indicator. Allowed values: {P_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.
P_HIGH_PROBABILITY_OF_COMPLETION	BOOLEAN_SET	Value = TRUE : high probability of call completion field can be set. Value = FALSE : high probability of call completion field can not be set. FALSE is the default value.

The following table explains how the P_TRIGGERING_ADDRESSES property that is inherited via the Generic Call Control properties should be interpreted with respect to which of the notifications apply to originating numbers and which of the notifications apply to terminating numbers.

P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	Originating
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	Originating
P_CALL_EVENT_ADDRESS_COLLECTED	Originating
P_CALL_EVENT_ADDRESS_ANALYSED	Originating
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	Originating
P_CALL_EVENT_ORIGINATING_RELEASE	Originating
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT	Terminating
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED	Terminating
P_CALL_EVENT_ALERTING	Terminating
P_CALL_EVENT_ANSWER	Terminating
P_CALL_EVENT_TERMINATING_RELEASE	Terminating
P_CALL_EVENT_REDIRECTED	Terminating
P_CALL_EVENT_TERMINATING_SERVICE_CODE	Terminating
P_CALL_EVENT_QUEUED	N/A

8.2 Service Property values for the CAMEL Service Environment.

Implementations of the MultiParty 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 = {
"IpMultiPartyCallControlManager.createCall",
"IpMultiPartyCallControlManager.createNotification",
"IpMultiPartyCallControlManager.destroyNotification",
"IpMultiPartyCallControlManager.changeNotification",
"IpMultiPartyCallControlManager.getNotification",
"IpMultiPartyCallControlManager.getNextNotification",
"IpMultiPartyCallControlManager.enableNotifications",
"IpMultiPartyCallControlManager.disableNotifications",
"IpMultiPartyCallControlManager.setCallLoadControl"
"IpMultiPartyCall.getCallLegs",
"IpMultiPartyCall.createCallLeg",
"IpMultiPartyCall.createAndRouteCallLegReq",
"IpMultiPartyCall.release",
"IpMultiPartyCall.deassignCall",
"IpMultiPartyCall.getInfoReq",
"IpMultiPartyCall.setChargePlan",
"IpMultiPartyCall.setAdviceOfCharge",
"IpMultiPartyCall.superviseReq",
"IpCallLeg.routeReq",
"IpCallLeg.eventReportReq",
"IpCallLeg.release",
"IpCallLeg.getInfoReq",
"IpCallLeg.getCall",
"IpCallLeg.continueProcessing"
}

P_TRIGGERING_EVENT_TYPES = {
P_CALL_EVENT_ADDRESS_COLLECTED,
P_CALL_EVENT_ADDRESS_ANALYSED,
P_CALL_EVENT_ORIGINATING_RELEASE,
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED,
P_CALL_EVENT_TERMINATING_RELEASE
}
```

NOTE: P_CALL_EVENT_ORIGINATING_RELEASE only for the routing failure case, TpReleaseCause = P_ROUTING_FAILURE.

```
P_DYNAMIC_EVENT_TYPES = {
P_CALL_EVENT_ALERTING,
P_CALL_EVENT_ANSWER,
P_CALL_EVENT_ORIGINATING_RELEASE,
P_CALL_EVENT_ORIGINATING_SERVICE_CODE,
P_CALL_EVENT_TERMINATING_RELEASE,
P_CALL_EVENT_TERMINATING_SERVICE_CODE
}

P_ADDRESS_PLAN = {
P_ADDRESS_PLAN_E164
}

P_UI_CALL_BASED = {
TRUE
}

P_UI_AT_ALL_STAGES = {
FALSE
}

P_MEDIA_TYPE = {
P_AUDIO
}

P_MAX_CALLLEGS_PER_CALL = {
1,
2,
3,
4,
5,
6
}

P_UI_CALLLEG_BASED = {
TRUE
}
```

```
P_MEDIA_ATTACH_EXPLICIT = {
FALSE
}
```

9 Multi-Party Call Control Data Definitions

This clause provides the MPCC 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 3GPP TS 29.198-4-1 or in the common data definitions which may be found in 3GPP TS 29.198-2.

9.1 Event Notification Data Definitions

No specific event notification data defined.

9.2 Multi-Party Call Control Data Definitions

9.2.1 IpCallLeg

Defines the address of an `IpCallLeg` Interface.

9.2.2 IpCallLegRef

Defines a `Reference` to type IpCallLeg.

9.2.3 IpAppCallLeg

Defines the address of an `IpAppCallLeg` Interface.

9.2.4 IpAppCallLegRef

Defines a `Reference` to type IpAppCallLeg.

9.2.5 IpMultiPartyCall

Defines the address of an `IpMultiPartyCall` Interface.

9.2.6 IpMultiPartyCallRef

Defines a `Reference` to type IpMultiPartyCall.

9.2.7 IpAppMultiPartyCall

Defines the address of an `IpAppMultiPartyCall` Interface.

9.2.8 IpAppMultiPartyCallRef

Defines a `Reference` to type IpAppMultiPartyCall.

9.2.9 IpMultiPartyCallControlManager

Defines the address of an `IpMultiPartyCallControlManager` Interface.

9.2.10 IpMultiPartyCallControlManagerRef

Defines a `Reference` to type IpMultiPartyCallControlManager.

9.2.11 IpAppMultiPartyCallControlManager

Defines the address of an `IpAppMultiPartyCallControlManager` Interface.

9.2.12 IpAppMultiPartyCallControlManagerRef

Defines a `Reference` to type IpAppMultiPartyCallControlManager..

9.2.13 TpAppCallLegRefSet

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

9.2.14 TpMultiPartyCallIdentifier

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

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

9.2.15 TpAppMultiPartyCallBack

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

	Tag Element Type	
	TpAppMultiPartyCallBackRefType	

Tag Element Value	Choice Element Type	Choice Element Name
P_APP_CALLBACK_UNDEFINED	NULL	Undefined
P_APP_MULTIPARTY_CALL_CALLBACK	IpAppMultiPartyCallRef	AppMultiPartyCall
P_APP_CALL_LEG_CALLBACK	IpAppCallLegRef	AppCallLeg
P_APP_CALL_AND_CALL_LEG_CALLBACK	TpAppCallLegCallBack	AppMultiPartyCallAndCallLeg

9.2.16 TpAppMultiPartyCallBackRefType

Defines the type application call back interface.

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

9.2.17 TpAppCallLegCallBack

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

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

9.2.18 TpMultiPartyCallIdentifierSet

Defines a Numbered Set of Data Elements of TpMultiPartyCallIdentifier.

9.2.19 TpCallAppInfo

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

	Tag Element Type	
	TpCallAppInfoType	

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

9.2.20 TpCallAppInfoType

Defines the type of call application-related specific information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)
P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. telephony)
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64 kbit/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified service-service information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address
P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS	9	Contains the original address specified by the originating user when launching the call.
P_CALL_APP_REDIRECTING_ADDRESS	10	Contains the address of the user from which the call is diverting.
P_CALL_APP_HIGH_PROBABILITY_COMPLETION	11	Indicates high probability of completion and its priority
P_CALL_APP_CARRIER	12	Indicates the set of Carrier identifications to be used to route the call.

9.2.21 TpCallAppInfoSet

Defines a Numbered Set of Data Elements of TpCallAppInfo.

9.2.22 TpCallEventRequest

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

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

9.2.23 TpCallEventRequestSet

Defines a Numbered Set of Data Elements of TpCallEventRequest.

9.2.24 TpCallEventType

Defines a specific call event report type.

Name	Value	Description
P_CALL_EVENT_UNDEFINED	0	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	1	An originating call attempt takes place (e.g. Off-hook event).
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	2	An originating call attempt is authorised
P_CALL_EVENT_ADDRESS_COLLECTED	3	The destination address has been collected.
P_CALL_EVENT_ADDRESS_ANALYSED	4	The destination address has been analysed.
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	5	Mid-call originating service code received.
P_CALL_EVENT_ORIGINATING_RELEASE	6	A originating call/call leg is released
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT	7	A terminating call attempt takes place
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED	8	A terminating call is authorized
P_CALL_EVENT_ALERTING	9	Call is alerting at the call party.
P_CALL_EVENT_ANSWER	10	Call answered at address.
P_CALL_EVENT_TERMINATING_RELEASE	11	A terminating call leg has been released or the call could not be routed.
P_CALL_EVENT_REDIRECTED	12	Call redirected to new address: an indication from the network that the call has been redirected to a new address (no events disarmed as a result of this).
P_CALL_EVENT_TERMINATING_SERVICE_CODE	13	Mid call terminating service code received.
P_CALL_EVENT_QUEUED	14	The Call Event has been queued. (no events are disarmed as a result of this)

EVENT HANDLING RULES:

The following general event handling rules apply to dynamically armed events:

When requesting events for one leg;

- When the monitor mode is set to P_CALL_MONITOR_MODE_DO_NOT_MONITOR all events armed for that eventtype are disarmed. The additionalEventCriteria are not taken into account.
- When requesting two events for the same event type with different criteria and/or different monitor mode the last used criteria and monitor mode apply.
- Events that are not applicable to a leg are refused with exception P_INVALID_EVENT_TYPE. The same exception is used when criteria are used that are not applicable to the leg,
  E.g., requesting P_CALL_EVENT_TERMINATING_SERVICE_CODE on an originating leg is refused with exception P_INVALID_CRITERIA.
  When P_CALL_EVENT_ORIGINATING_RELEASE is requested with P_BUSY in the criteria the request is refused with the same exception.

When receiving events:

- If an armed event is met, then it is disarmed, unless explicit stated that it will not to be disarmed.
- If an event is met that causes the release of the related leg, then all events related to that leg are disarmed .
- When an event is met on a call leg irrespective of the event monitor mode, then only events belonging to that call leg may become disarmed (see table below) .
- If a call is released, then all events related to that call are disarmed.

NOTE 1: Event disarmed means monitor mode is set to DO_NOT_MONITOR. and event armed means monitor mode is set to INTERRUPT or NOTIFY..

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

Event Occurred	Events Disarmed
P_CALL_EVENT_UNDEFINED	Not Applicable
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	Not applicable, can only be armed as trigger
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED
P_CALL_EVENT_ADDRESS_COLLECTED	P_CALL_EVENT_ADDRESS_COLLECTED
P_CALL_EVENT_ADDRESS_ANALYSED	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED
P_CALL_EVENT_ALERTING	P_CALL_EVENT_ALERTING P_CALL_EVENT_TERMINATING_RELEASE with criteria: P_USER_NOT_AVAILABLE P_BUSY P_NOT_REACHABLE P_ROUTING_FAILURE P_CALL_RESTRICTED P_UNAVAILABLE_RESOURCES
P_CALL_EVENT_ANSWER	P_CALL_EVENT_ALERTING P_CALL_EVENT_ANSWER P_CALL_EVENT_TERMINATING_RELEASE with criteria: P_USER_NOT_AVAILABLE P_BUSY P_NOT_REACHABLE P_ROUTING_FAILURE P_CALL_RESTRICTED P_UNAVAILABLE_RESOURCES P_NO_ANSWER
P_CALL_EVENT_ORIGINATING_RELEASE	All pending network events for the call leg are disarmed
P_CALL_EVENT_TERMINATING_RELEASE	All pending network events for the call leg are disarmed
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	P_CALL_EVENT_ORIGINATING_SERVICE_CODE *) see NOTE 2
P_CALL_EVENT_TERMINATING_SERVICE_CODE	P_CALL_EVENT_TERMINATING_SERVICE_CODE *) see NOTE 2
NOTE 2: Only the detected service code or the range to which the service code belongs is disarmed.	

NOTE 3: ON MAPPING EVENTYPES TO IN TRIGGER DETECTION POINTS (TDPs):
When the eventtypes as defined above are used for requesting the initial notification (with createNotification), not all events have a one to one correspondence with a Trigger Detection Point (TDP). For instance, when the underlying network is ITU-T CS2 based, one cannot distinghuish in createNotification whether the P_CALL_EVENT_ORIGINATING_RELEASE is intended to be on the Originating side (O_BCSM) or the Terminating side (T_BCSM) of the call. Likewise , the P_CALL_EVENT_ANSWER, P_CALL_EVENT_ALERTING and the P_CALL_EVENT_TERMINATING_RELEASE.

The basic assumption is that the operator is responsible for provisioning of triggers in the network as in this domain full awarness exists of all other services and applications.Therefore, createNotification does not automatically lead to immediate provisioning of these triggers. And thus in createNotification it is not necessary to indicate whether the initial notification should be on the originating or terminating side of the call.

9.2.25 TpAdditionalCallEventCriteria

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

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpInt32	MinAddressLength
P_CALL_EVENT_ADDRESS_ANALYSED	NULL	Undefined
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	TpCallServiceCodeSet	OriginatingServiceCode
P_CALL_EVENT_ORIGINATING_RELEASE	TpReleaseCauseSet	OriginatingReleaseCauseSet
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_TERMINATING_RELEASE	TpReleaseCauseSet	TerminatingReleaseCauseSet
P_CALL_EVENT_REDIRECTED	NULL	Undefined
P_CALL_EVENT_TERMINATING_SERVICE_CODE	TpCallServiceCodeSet	TerminatingServiceCode
P_CALL_EVENT_QUEUED	NULL	Undefined

9.2.26 TpCallEventInfo

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

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

9.2.27 TpCallAdditionalEventInfo

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

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpAddress	CollectedAddress
P_CALL_EVENT_ADDRESS_ANALYSED	TpAddress	CalledAddress
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	TpCallServiceCode	OriginatingServiceCode
P_CALL_EVENT_ORIGINATING_RELEASE	TpReleaseCause	OriginatingReleaseCause
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_TERMINATING_RELEASE	TpReleaseCause	TerminatingReleaseCause
P_CALL_EVENT_REDIRECTED	TpAddress	ForwardAddress
P_CALL_EVENT_TERMINATING_SERVICE_CODE	TpCallServiceCode	TerminatingServiceCode
P_CALL_EVENT_QUEUED	NULL	Undefined

9.2.28 TpCallNotificationRequest

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

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

9.2.29 TpCallNotificationScope

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

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

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.
OriginatingAddress	TpAddressRange	Defines the origination address or address range for which the notification is requested.

9.2.30 TpCallNotificationInfo

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

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

9.2.31 TpCallNotificationReportScope

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

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddress	Contains the destination address of the call.
OriginatingAddress	TpAddress	Contains the origination address of the call.

9.2.32 TpNotificationRequested

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

Sequence Element Name	Sequence Element Type
AppCallNotificationRequest	TpCallNotificationRequest
AssignmentID	TpInt32

9.2.33 TpNotificationRequestedSet

Defines a numbered Set of Data Elements of TpNotificationRequested.

9.2.34 TpReleaseCause

Defines the reason for which a call is released.

Name	Value	Description
P_UNDEFINED	0	The reason of release is not known, because no info was received from the network.
P_USER_NOT_AVAILABLE	1	The user is not available in the network. This means that the number is not allocated or that the user is not registered.
P_BUSY	2	The user is busy.
P_NO_ANSWER	3	No answer was received.
P_NOT_REACHABLE	4	The user terminal is not reachable.
P_ROUTING_FAILURE	5	A routing failure occurred. For example an invalid address was received.
P_PREMATURE_DISCONNECT	6	The user disconnected the call / call leg during the setup phase.
P_DISCONNECTED	7	A disconnect was received.
P_CALL_RESTRICTED	8	The call was subject of restrictions.
P_UNAVAILABLE_RESOURCE	9	The request could not be carried out as no resources were available.
P_GENERAL_FAILURE	10	A general network failure occurred.
P_TIMER_EXPIRY	11	The call / call leg was released because an activity timer expired.
P_UNSUPPORTED_MEDIA	12	The call / call leg was released either because the message body of the request is in a format not supported or because the media is not supported.

9.2.35 TpReleaseCauseSet

Defines a Numbered Set of Data Elements of TpReleaseCause.

9.2.36 TpCallLegIdentifier

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

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

9.2.37 TpCallLegIdentifierSet

Defines a Numbered Set of Data Elements of TpCallLegIdentifier.

9.2.38 TpCallLegAttachMechanism

Defines how a CallLeg should be attached to the call.

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

9.2.39 TpCallLegConnectionProperties

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

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

9.2.40 TpCallLegInfoReport

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

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

9.2.41 TpCallLegInfoType

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

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

9.2.42 TpCallLegSuperviseTreatment

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

Name	Value	Description
P_CALL_LEG_SUPERVISE_RELEASE	01h	Release the call leg when the call leg supervision timer expires
P_CALL_LEG_SUPERVISE_RESPOND	02h	Notify the application when the call leg supervision timer expires
P_CALL_LEG_SUPERVISE_APPLY_TONE	04h	Send a warning tone on the call leg when the call leg supervision timer expires. If call leg release is requested, then the call leg will be released following the tone after an administered time period

9.2.43 TpCallHighProbabilityCompletion

This data type is identical to a TpInt32, and defines the probability of completion under network congestion. The values of this data type are region specific. In general, a value of 0 indicates no special treatment (default), a priority value between 1, 2, 3, ..., n indicates special treatment, where 1 is the highest priority and n the lowest priority other than no special treatment.

9.2.44 TpNotificationRequestedSetEntry

Defines the Sequence of Data Elements that specify a set of requested notifications and an indication whether more notifications can be requested.

Sequence Element Name	Sequence Element Type	Description
NotificationRequestSet	TpNotificationRequestedSet	Numbered set of requested notifications.
Final	TpBoolean	Indication whether the set of notifications is the final set (TRUE)or if there are more notifications available (FALSE).

9.2.45 TpCarrierSet

Defines a Numbered Set of Data Elements of TpCarrier. In case the set is empty, the SCF will assume default processing.

9.2.46 TpCarrier

Defines the Sequence of Data Elements that indicates carrier information. It consists of the carrier selection field followed by the Carrier ID information to be used for routing a call to a carrier.

Sequence Element Name	Sequence Element Type
CarrierID	TpCarrierID
CarrierSelectionField	TpCarrierSelectionField

9.2.47 TpCarrierID

This data type is identical to a TpOctetSet. For encoding of the field, depending on the network, either ITU-T Recommendation Q.763 [8] or ANSI ISUP T.113 [9] applies.

9.2.48 TpCarrierSelectionField

Defines the type of Carrier Selection Field-related specific information. This parameter indicates how the selected carrier is provided (e.g. pre-subscribed).

Name	Value	Description
P_CIC_UNDEFINED	0	No indication
P_CIC_NO_INPUT	1	The carrier identification code (CIC) is pre subscribed (not provided by the calling party).
P_CIC_INPUT	2	The carrier identification code (CIC) is pre subscribed and provided by the calling party.
P_CIC_UNDETERMINED	3	The selected carrier identification code (CIC) is pre subscribed, but no indication is present of whether it is provided by the calling party (undetermined).
P_CIC_NOT_PRESCRIBED	4	The selected carrier identification code (CIC) is provided by calling party (not pre subscribed).

Annex A (normative): OMG IDL Description of Multi-Party Call Control SCF

The OMG IDL representation of this interface specification is contained in text files mpcc_data.idl and mpcc_interfaces.idl (contained in archive 2919804-3V590IDL.ZIP) which accompany the present document.

Annex B (informative): W3C WSDL Description of Multi-Party Call Control SCF

Significant changes have occurred in Web Services technologies and understanding of how to best apply Web Services as a realisation of OSA. These changes are not reflected and therefore this realisation is removed. A future activity may provide a replacement for the content of this annex, reflective of current technology and usage expected.

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 Part 1 of this specification series. 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 2919804-3V590J2SE.ZIP that accompanies the present document.

The J2EE™ representation of this interface specification is provided as Java™ Code, contained in archive 2919804-3V590J2EE.ZIP that accompanies the present document.

Annex D (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	1.0.0
June 2001	CN_12	NP-010327	--	--	Approved at TSG CN#12 and placed under Change Control	2.0.0	4.0.0
Sep 2001	CN_13	NP-010467	001	--	Changing references to JAIN	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	002	--	Correction of text descriptions for methods enableCallNotification and createNotification	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	003	--	Specify the behaviour when a call leg times out	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	004	--	Removal of Faulty state in MPCCS Call State Transition Diagram and method callFaultDetected in MPCCS in OSA R4	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	005	--	Missing TpCallAppInfoSet description in OSA R4	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	006	--	Redirecting a call leg vs. creating a call leg clarification in OSA R4	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	007	--	Introduction of MPCC Originating and Terminating Call Leg STDs for IpCallLeg	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	008	--	Corrections to SetChargePlan() Addition of PartyToCharge parmeter	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	009	--	Corrections to SetChargePlan()	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	010	--	Remove distinction between final- and intermediate-report	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	011	--	Inclusion of TpMediaType	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	012	--	Corrections to GCC STD	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	013	--	Introduction of sequence diagrams for MPCC services	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	014	--	The use of the REDIRECT event needs to be illustrated	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	015	--	Corrections to SetCallChargePlan()	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	016	--	Add one additional error indication	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	017	--	Corrections to Call Control – GCCS Exception handling	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	018	--	Corrections to Call Control – Errors in Exceptions	4.0.0	4.1.0
Dec 2001	CN_14	NP-010597	019	--	Replace Out Parameters with Return Types	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	020	--	Removal of time based charging property	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	021	--	Make attachMedia() and detachMedia() asynchronous	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	022	--	Correction of treatment datatype in superviseReq on call leg	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	023	--	Corrections to Call Control Data Types	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	024	--	Correction to Call Control (CC)	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	025	--	Amend the Generic Call Control introductory part	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	026	--	Correction in TpCallEventType	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	027	--	Addition of missing description of RouteErr()	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	028	--	Misleading description of createAndRouteCallLegErr()	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	029	--	Correction to values of TpCallNotificationType, TpCallLoadControlMechanismType	4.1.0	4.2.0
Dec 2001	CN_14	NP-010695	030	--	Correction of method getLastRedirectionAddress	4.1.0	4.2.0
Mar 2002	CN_15	NP-020106	031	--	Add P_INVALID_INTERFACE_TYPE exception to IpService.setCallback() and IpService.setCallbackWithSessionID()	4.2.0	4.3.0
Mar 2002	CN_15	NP-020106	032	--	Correction of Event Subscription/Notification Data Type	4.2.0	4.3.0
Mar 2002	CN_15	NP-020106	033	--	Correction of parameter name in IpCallLeg.routeReq() and in IpCallLeg.setAdviceOfCharge()	4.2.0	4.3.0
Mar 2002	CN_15	NP-020106	034	--	Clarification of ambiguous Event handling rules	4.2.0	4.3.0
Jun 2002	CN_16	NP-020180	035	--	Correction to TpCallChargePlan	4.3.0	4.4.0
Jun 2002	CN_16	NP-020180	036	--	Correction to CAMEL Service Property values	4.3.0	4.4.0
Jun 2002	CN_16	NP-020181	037	--	Addition of support for Java API technology realisation	4.4.0	5.0.0
Jun 2002	CN_16	NP-020182	038	--	Addition of support for WSDL realisation	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	039	--	Addition of support for Emergency Telecommunications Service	4.4.0	5.0.0
Jun 2002	CN_16	NP-020183	040	--	Addition of support for Network Controlled Notifications MPCC	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	041	--	Changes to getNotification()	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	042	--	Addition of P_UNSUPPORTED_MEDIA release cause to TpReleaseCause	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	043	--	Addition of CAMEL Phase 4 Service Property values	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	044	--	Addition of indication whether SCS supports initially multiple routeReqs in parallel	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	045	--	Explicit exception for continueProcessing when not in interrupted mode	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	046	--	Indication needed that supervision will be ended when call or callLeg is deassigned	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	047	--	Clarify ambiguous Supervision duration	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	048	--	Detach/Attach request illegal during pending Attach/Detach request	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	049	--	Correction of Multi-Party Call Control properties	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	050	--	Correcting the sequence diagram descriptions in GCC and MPCC	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	051	--	Correcting erroneous description of UI behaviour in call control	4.4.0	5.0.0

Jun 2002	CN_16	NP-020187	052	--	Correcting the descriptions of sequence diagrams that don't match the diagram	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	053	--	Correcting erroneous references to GCC in MPCC	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	054	--	Addition of the Multi-media APIs to Call control SCF (29.198-4)	4.4.0	5.0.0
Jun 2002	CN_16	NP-020187	055	--	Updating Clause 4 for Release 5	4.4.0	5.0.0
Jun 2002	CN_16	NP-020188	056	--	Spliting of 29.198-04 into 4 separate TSs (sub-parts)	4.4.0	5.0.0
Sep 2002	CN_17	NP-020431	001	--	29.198-04-3 Correction of error in Call Forward on Busy sequence diagram	5.0.0	5.1.0
Sep 2002	CN_17	NP-020431	002	--	Correct inconsistencies in IpCallLeg state transition diagrams	5.0.0	5.1.0
Sep 2002	CN_17	NP-020431	003	--	Clarification of the overlapping criteria definition and eventType mapping to IN TDPs	5.0.0	5.1.0
Sep 2002	CN_17	NP-020431	004	--	Add support for Carrier selection	5.0.0	5.1.0
Sep 2002	CN_17	NP-020431	005	--	Correction on use of NULL in Call Control API	5.0.0	5.1.0
Sep 2002	CN_17	NP-020395	006	--	Add text to clarify relationship between 3GPP and ETSI/Parlay OSA specifications	5.0.0	5.1.0
Mar 2003	CN_19	NP-030031	007	--	Correction of status of MPCC methods	5.1.0	5.2.0
Mar 2003	CN_19	NP-030031	008	--	Inconsistent description of use of secondary callback	5.1.0	5.2.0
Mar 2003	CN_19	NP-030020	009	--	Correction to TpReleaseCauseSet in Multi Party Call Control IDL	5.1.0	5.2.0
Mar 2003	CN_19	NP-030130	010	--	Correction of definition of the P_MAX_CALLLEGS_PER_CALL	5.1.0	5.2.0
Jun 2003	CN_20	NP-030238	011	--	Correction of the description for callEventNotify & reportNotification	5.2.0	5.3.0
Sep 2003	CN_21	NP-030352	014	--	Correction to Java Realisation Annex	5.3.0	5.4.0
Dec 2003	CN_22	NP-030544	015	--	Correction of description in superviseRes	5.4.0	5.5.0
Dec 2003	CN_22	NP-030550	016	--	Correction of description of TpNotificationRequestedSetEntry	5.4.0	5.5.0
Apr 2004	CN_23bis	NP-040155	020	--	Correct Java Code to conform with Java Rulebook in TS 29.198-01 and to remove errors	5.5.0	5.6.0
Jun 2004	CN_24	NP-040256	022	--	Correct the P_TRIGGERING_ADDRESSES service property	5.6.0	5.7.0
Jun 2004	CN_24	NP-040257	025	--	Correction of callbacks sequence and timing conditions in MPCCS	5.6.0	5.7.0
Sep 2004	CN_25	NP-040355	028	--	Correct J2EE source	5.7.0	5.8.0
Dec 2004	CN_26	NP-040485	034	--	Removal of OSA API SCFs description in W3C WSDL	5.8.0	5.9.0
Dec 2004	--	--	--	--	Added missing code attachments	5.9.0	5.9.1

History

Document history		
V5.0.0	June 2002	Publication
V5.1.0	September 2002	Publication
V5.2.0	March 2003	Publication
V5.3.0	June 2003	Publication
V5.4.0	September 2003	Publication
V5.5.0	December 2003	Publication
V5.6.0	May 2004	Publication
V5.7.0	August 2004	Publication
V5.8.0	September 2004	Publication
V5.9.0	December 2004	Publication (Withdrawn)
V5.9.1	December 2004	Publication