3GPP TSG CN Plenary Meeting #17 **NP-020431**
4 - 6 September 2002, Biarritz, FRANCE

Source: **CN5 (OSA)**

Title: **Rel-5 CRs 29.198-04-3 OSA API; Multi-Party CC SCF**

Agenda item: **8.2**

Document for: **APPROVAL**

Doc-1st-Level	Spec	CR	Rev	Phase	Subject	Cat	Version-Current	Doc-2nd-Level	Workitem
NP-020431	29.198-04-3	001	-	Rel-5	Correction of error in Call Forward on Busy sequence diagram	F	5.0.0	N5-020605	OSA2
NP-020431	29.198-04-3	002	-	Rel-5	Correct inconsistencies in IpCallLeg state transition diagrams	F	5.0.0	N5-020754	OSA2
NP-020431	29.198-04-3	003	-	Rel-5	Clarification of the overlapping criteria definition and eventType mapping to IN TDPs	F	5.0.0	N5-020756	OSA2
NP-020431	29.198-04-3	004	-	Rel-5	Add support for Carrier selection	F	5.0.0	N5-020759	OSA2
NP-020431	29.198-04-3	005	-	Rel-5	Correction on use of NULL in Call Control API	A	5.0.0	N5-020766	OSA2

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-04-3** **CR** **001** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘ (U)SIM ☐ ME/UE ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Correction of error in Call Forward on Busy sequence diagram
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
REL-4 (Release 4)
REL-5 (Release 5)

Reason for change:	⌘	In the Call Forward on Busy the text says that the B-leg is continued, but the sequence shows the A-leg being continued.
Summary of change:	⌘	Changed the sequence to conform with the text.
Consequences if not approved:	⌘	Unclear specification. Confusion.

Clauses affected:	⌘	7.1.3
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

How to create CRs using this form:
Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

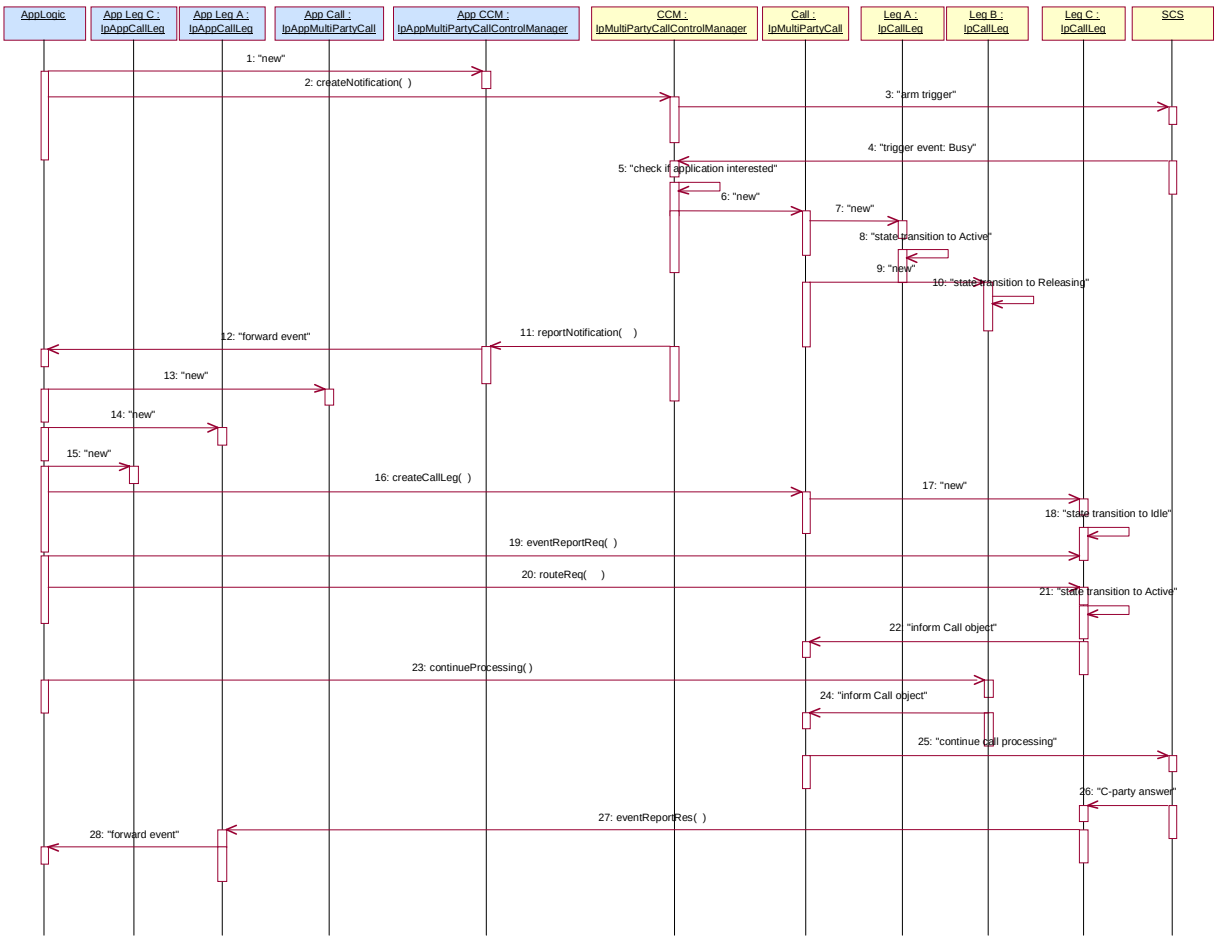
Proposed Changes

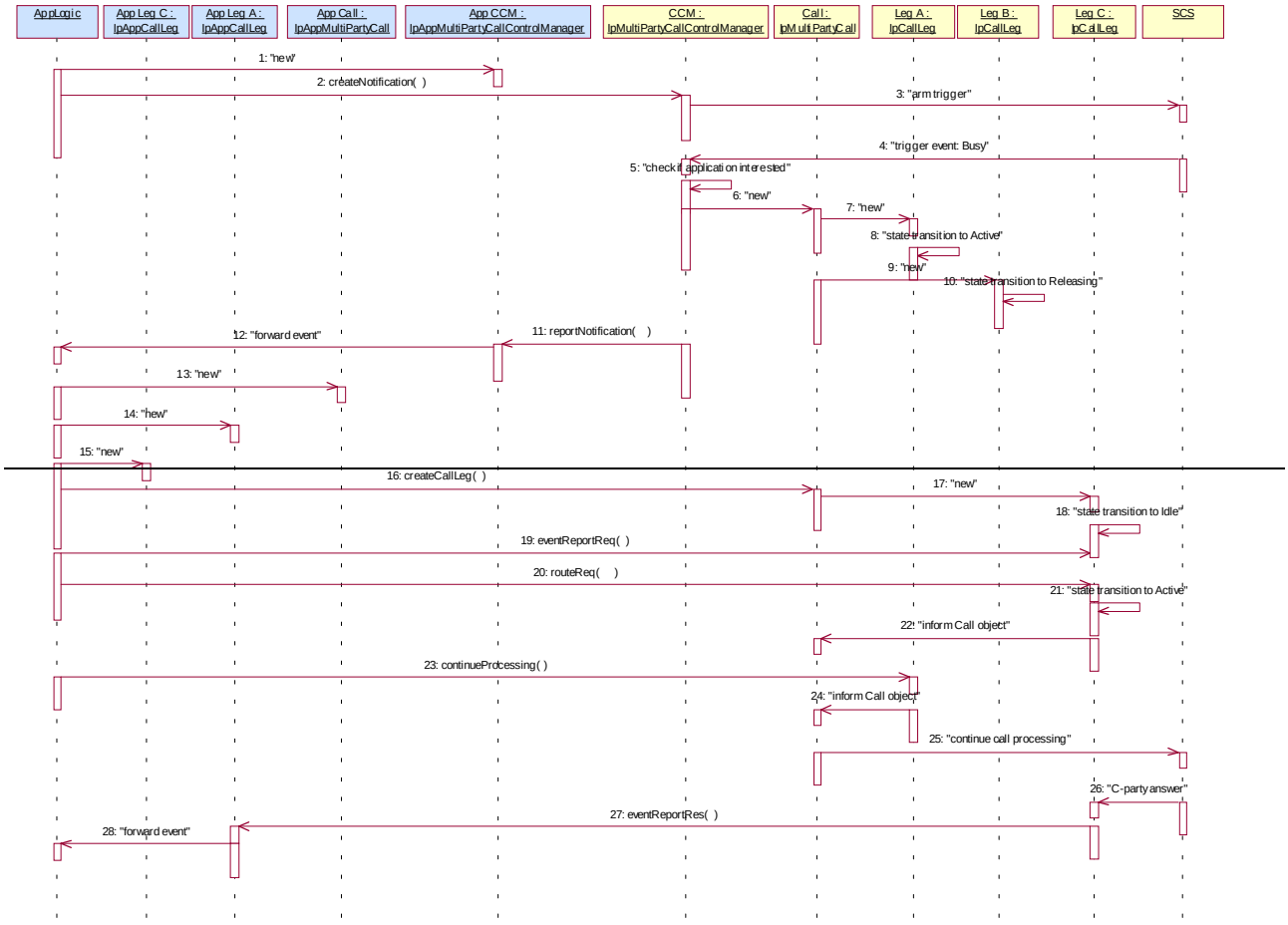
The following changes are proposed to 29.198-04:

7.1.3 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 Initiating.

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: 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~~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.

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020754**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-04-3** **CR** **002** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘ (U)SIM ☐ ME/UE ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Correct inconsistencies in IpCallLeg state transition diagrams
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
REL-4 (Release 4)
REL-5 (Release 5)

Reason for change:	⌘	In the descriptions of the state transition diagrams some inconsistencies and unclarities are found
Summary of change:	⌘	Clarifications in the text. Some missing exit events added.
Consequences if not approved:	⌘	Unclear specification leading to misunderstanding when writing applications.
Clauses affected:	⌘	7.4.3
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

How to create CRs using this form:
Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

Introduction

Some errors were encountered in the legState STDs, specifically that a routeReq in analysing would continue the processing of the leg and that in active after the address analysed event a connection would be set up to the remote party.

Some unclarities were found, e.g., the propagated origating release should be detected as a terminating release on the terminating leg. Also deassign was missing from the releasing states as exit event.

Some inconsistencies were solved; e.g., the timer expiry was only mentioned for some of the states.

Furthermore, various small textual changes are proposed that should make the text more understandable.

Proposed Changes

The following changes are proposed to 29.198-04:

7.4.3 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 ~~is to~~must send 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.

NOTE1: 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.

~~A terminating release on the remote party is not directly visible on the originating leg, but the network can optionally propagate the termination of the remote party. In analog, an originating release on the calling leg can optionally be propagated to a terminating leg. In both cases the call leg STD will transit to Releasing state as a result of the propagated release.~~
~~Also when the call is released, all the legs will transit to the releasing state. 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.~~

NOTE2: 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.

7.4.3.1 Originating Call Leg

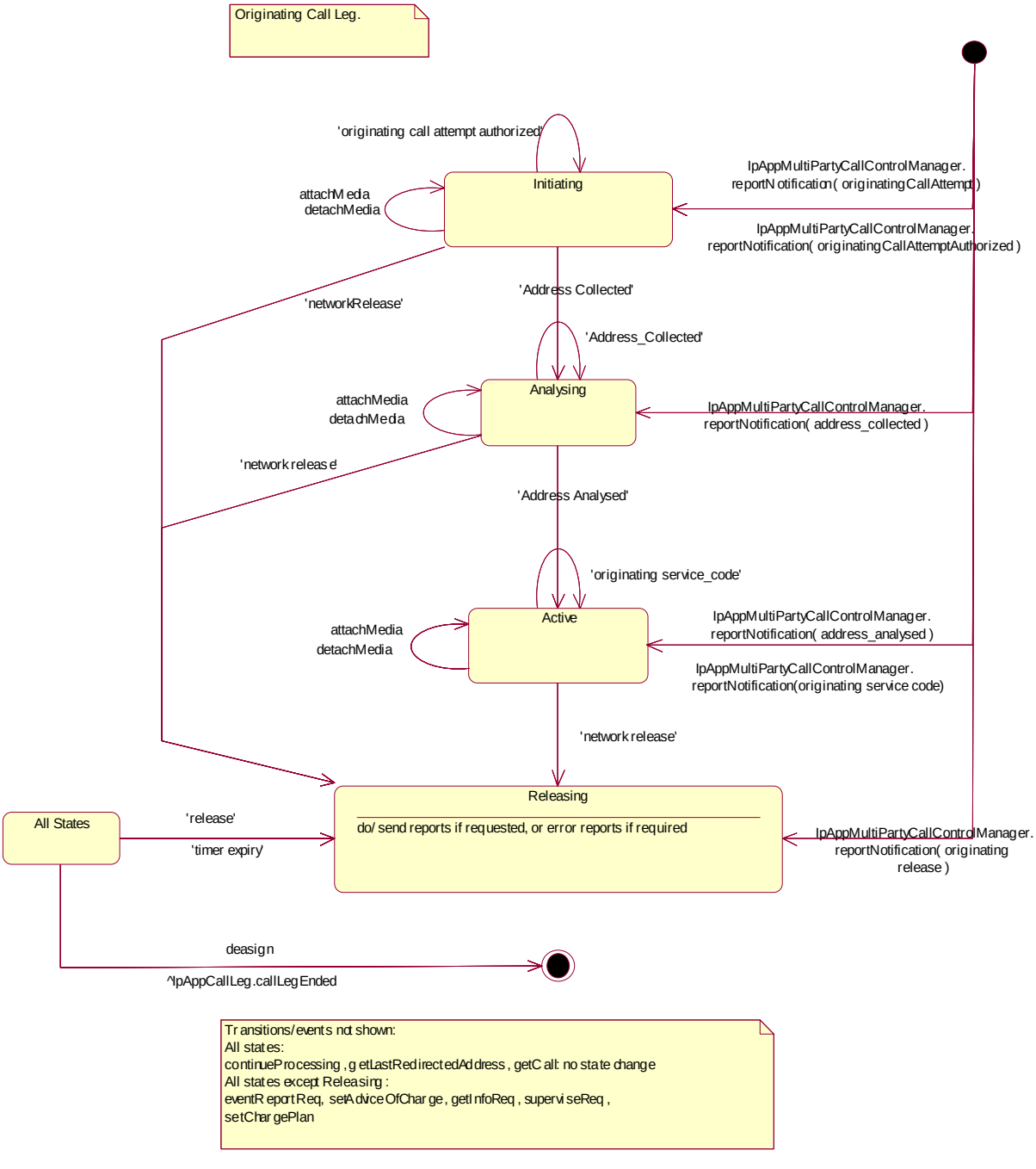


Figure : Originating Leg

7.4.3.1.1 Initiating State

Entry events:

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

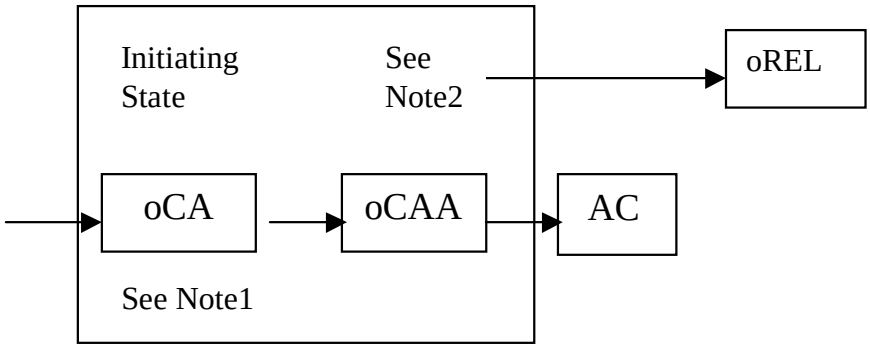
~~iii)~~ Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Originating_Call_Attempt_Authorised" initial notification criterion.

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: **o**riginating 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 a "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 ~~the~~ processing ~~leg~~ 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.

7.4.3.1.2 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:

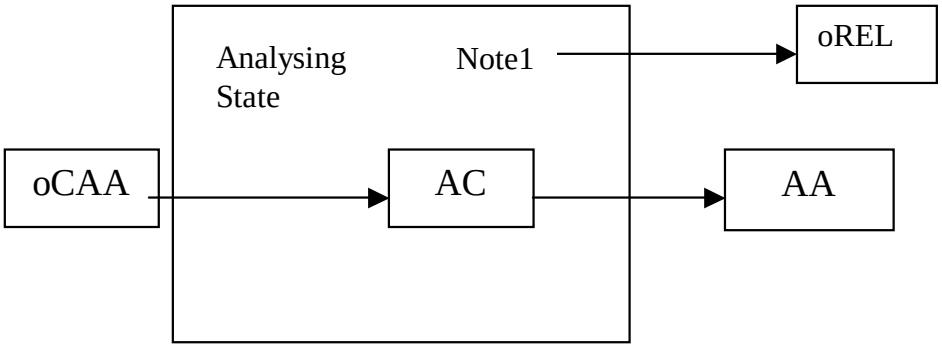
In this state the destination address provided by the calling party is collected and analysed.

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 ~~is recursive~~can be repeated, ~~e.g. the application could ask for 3 digits to be collected and when report request can be done repeatedly,~~ e.g. the application may ~~for example~~ 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 a "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 a 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~~leg~~ is suspended for the leg.
- Detection of a "originating release" indication as a result of a premature disconnect from the calling party.

7.4.3.1.3 Active State

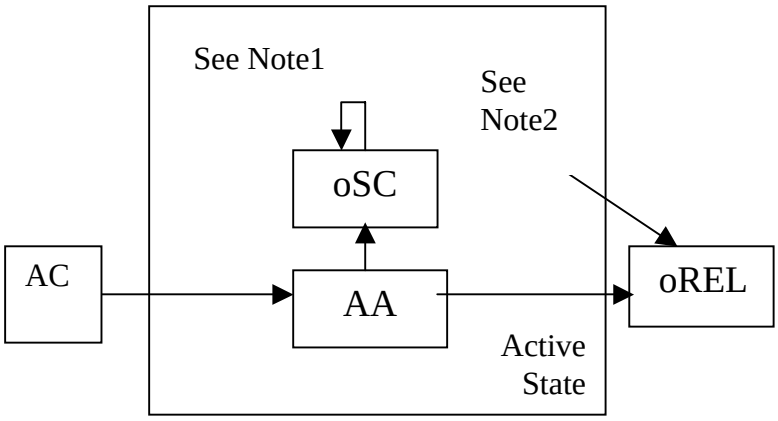
Entry events:

- 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 ~~In~~ this state the routing information is interpreted, the authority of the calling party to establish this connection is verified. Note that ~~and the~~ 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~~create~~ the connection to terminating leg automatically based on the received information.
- In this state a connection to the calling party is established.
- ~~Detection of a "terminating release" indication (not visible to the application) from remote party caused by a network release event propagated from a terminating party, possibly resulting in an "originating release" indication and causing the originating call leg STD to transit to Releasing state:~~

~~- Detection of a disconnect from the calling party.~~

~~- Receipt of a deassign() method.~~

~~- Receipt of a release() method.~~

- 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 ~~or a propagated disconnect from the called party.~~and a ~~"terminating release" indication as a result of a disconnect from called 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~~leg~~ is suspended.

7.4.3.1.4 Releasing State

Entry events:

- Detection of an "Originating_Release" indication as a result of the network release initiated by calling party.

- Propagated release from~~or~~ called party ~~(if propagated)~~.

- 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.

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

In this state the following functions are applicable:

- The detection of a “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~~leg~~ 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.

7.4.3.1.5 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

7.4.3.2 Terminating Call Leg

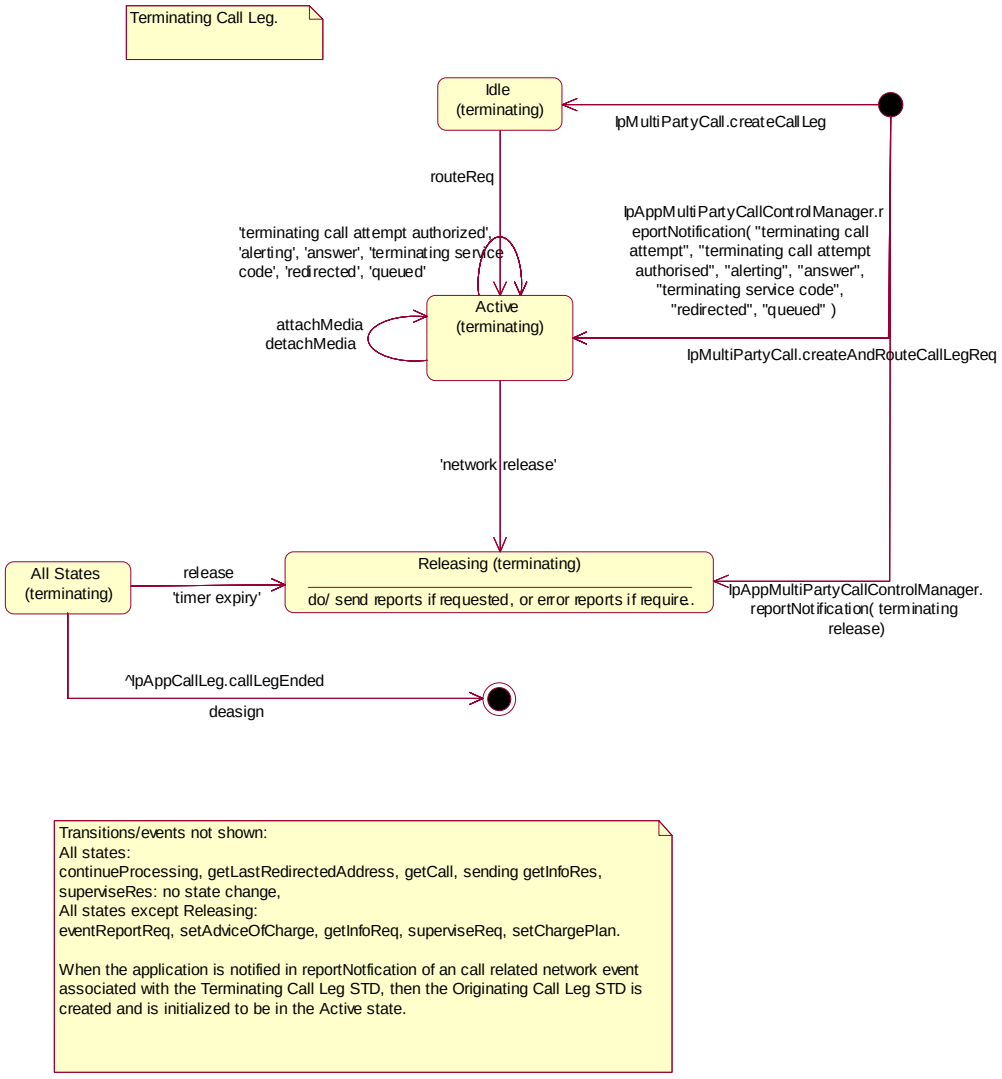


Figure : Terminating Leg

7.4.3.2.1 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.
- ~~Resumption of call leg processing occurs on receipt of a routeReq() method.~~

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.
- ~~Detection of a~~ Ppropagation~~ed~~ of network release event ~~being an "originating release" indication~~ as a result of a ~~premature~~ 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~~leg~~ is suspended for the leg.

7.4.3.2.2 Active (terminating) State

Entry events:

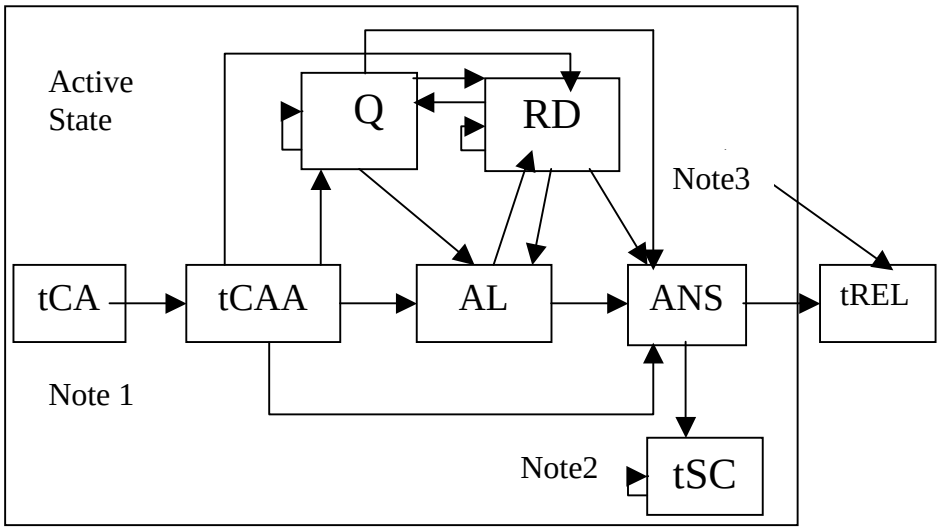
- Receipt of a~~n~~ routeReq will result in actually routing the call leg object.
- Receipt of a createAndRouteCallLegReq() method to start an application initiated call leg connection.
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for the following trigger criteria: "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~~n~~ "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.

- ~~Detection of a~~ propagation of~~ed~~ network release event ~~being an "originating release" indication~~ as a result of the following events:

  ~~vi)~~ Detection of a premature disconnect from the calling party.

- Receipt of a deassign() method.

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

~~-~~ Detection of a propagation of~~ed~~ network release event ~~being an "originating release" indication~~ as a result of a disconnect from the calling party .

- Detection of a network release event being~~or~~ 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~~leg~~ is suspended for the leg.

7.4.3.2.3 Releasing (terminating) State

Entry events:

- ~~Detection of a P~~propagation of~~ed~~ network release ~~event being an "originating release" indication as a result of the network release initiated~~ disconnect ~~by~~ from the calling party.

- Detection of a network release event being ~~or~~ a "terminating release" indication as a result of the network release initiated by called party..

- ~~Detection of a R~~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.

- ~~Detection of a~~ propagation~~ed~~ of network release event ~~being an "originating release" indication~~ as a result of the following events:

  ~~vi)~~ 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) ~~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~~leg~~ 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.

7.4.3.2.4 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

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020756**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-04-3** **CR** **003** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘ (U)SIM ☐ ME/UE ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Clarification of the overlapping criteria definition and eventType mapping to IN TDPs
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

Use one of the following releases:
2 *(GSM Phase 2)*
R96 *(Release 1996)*
R97 *(Release 1997)*
R98 *(Release 1998)*
R99 *(Release 1999)*
REL-4 *(Release 4)*
REL-5 *(Release 5)*

Reason for change:	⌘	The definition of overlapping criteria when requesting for notifications is too restricted, especially compared to the definition in Generic Call Control. As a result of this incorrect definition of overlapping criteria, it was found necessary to clarify that there is no exact mapping from eventypes used in notifications, to IN TDPs.
Summary of change:	⌘	The definition of overlapping criteria in MPCC is generalised. Clarification note is added on the relation between eventTypes and IN TDPs.
Consequences if not approved:	⌘	Misunderstanding about the eventTypes used in MPCC, Limited application provisioning: applications are much more restricted in requesting notifications (“triggers”) than should be.

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

How to create CRs using this form:
Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be

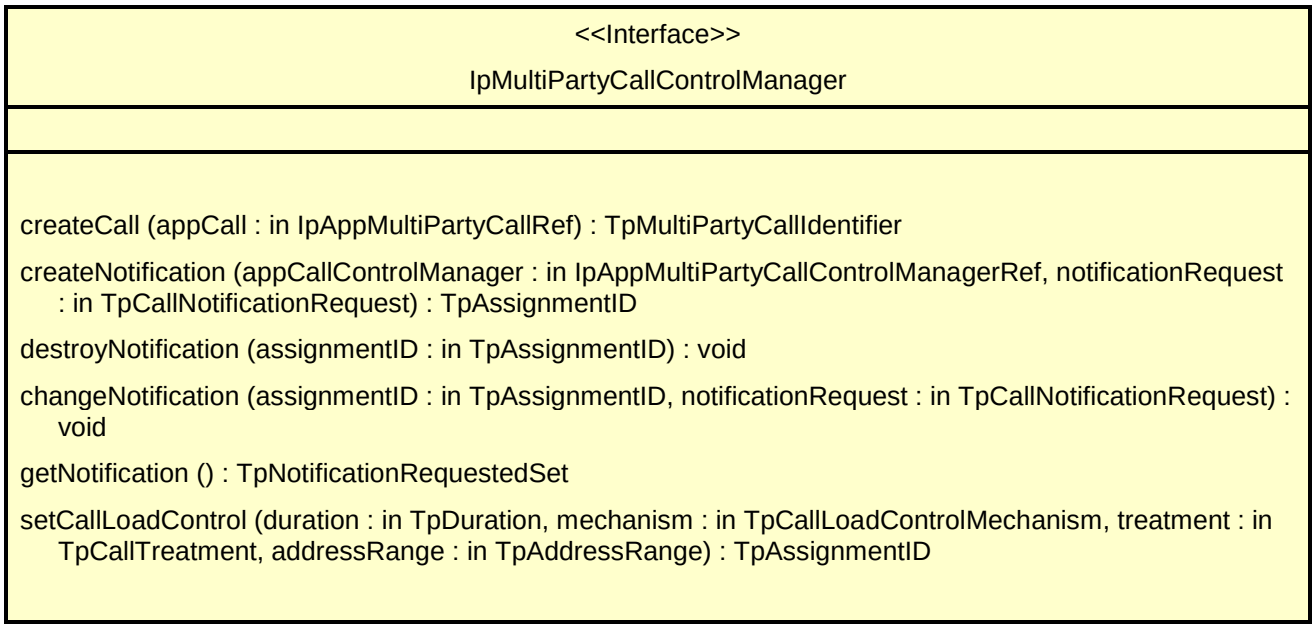
downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

4.1.1 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.



Method

createCall()

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

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

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

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, 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 both originating and terminating ranges overlap and the same number plan is used.~~

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.

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

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 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

****

Second Correction

****

4.1.1.1 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 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.

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020759**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-04-3** **CR** **004** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘ (U)SIM ☐ ME/UE ☐ Radio Access Network ☐ Core Network ☐

Title:	⌘	Add support for Carrier selection
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	19/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
REL-4 (Release 4)
REL-5 (Release 5)

Reason for change:	⌘	Addition of support for Carrier selection
Summary of change:	⌘	Addition of element in datatype TpCallAppInfo that can be used to provide carrier related information for a call
Consequences if not approved:	⌘	Applications are unable to specify alternative carriers.

Clauses affected:	⌘	9.2.19, 9.2.20, 9.2.45 - 9.2.48
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	Companion Rel-5 CR 29.198-01 (N5-020758)

How to create CRs using this form:
Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

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.

9.2.45 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 [32] or ANSI ISUP T.113 [33] 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).

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-04-3** **CR** **005** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: ⌘ (U)SIM ☐ ME/UE ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Correction on use of NULL in Call Control API
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	16/08/2002
Category:	⌘	**A**
Release:	⌘	*REL-5*

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

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
REL-4 (Release 4)
REL-5 (Release 5)

Reason for change:	⌘	OMG IDL does not support NULL as a valid value for a data type; attempts to send a null value result in a marshalling exception and a gateway can never receive the call.
Summary of change:	⌘	Occurrences of the use of NULL as a valid setting for Call Control API parameters have been replaced. Use of null modified to define appropriate behaviour in NOTIFY mode.
Consequences if not approved:	⌘	Failure to correct the API shall result in vendor specific interpretation and interoperability issues.

Clauses affected:	⌘	6.2
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	Rel-5 equivalent of 762 Rel-4 CR 29.198-04 (Mirror CR). Split of approved Rel-5 CR N5-020763 into CR for subpart 3 (the present CR) & CR for subpart 2 (N5-020765).

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.

<<Interface>> IpAppMultiPartyCallControlManager
reportNotification (callReference : in TpMultiPartyCallIdentifier, callLegReferenceSet : in TpCallLegIdentifierSet, notificationInfo : in TpCallNotificationInfo, assignmentID : in TpAssignmentID) : TpAppMultiPartyCallBack callAborted (callReference : in TpSessionID) : void managerInterrupted () : void managerResumed () : void callOverloadEncountered (assignmentID : in TpAssignmentID) : void callOverloadCeased (assignmentID : in TpAssignmentID) : void

6.2.2 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.

Returns appCallBack: Specifies references to the application interface which implements the callback interface for the new call and/or new call leg. ~~This parameter may be null if the notification is being given in NOTIFY mode.~~ If the application has previously explicitly passed a reference to the callback interface using a setCallback() invocation, this parameter may be set to P_APP_CALLBACK_UNDEFINED, or if supplied must be the same as that provided during the setCallback().

This parameter will be set to P_APP_CALLBACK_UNDEFINED if the notification is in NOTIFY mode.

Parameters

`callReference : in TpMultiPartyCallIdentifier`

Specifies the reference to the call interface to which the notification relates. ~~This parameter will be null i~~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, ~~this parameter will be null~~ 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