3GPP TSG CN Plenary Meeting #13 **NP-010466**
Beijing, China, 19th –21st September 2001

Source: **CN5 (OSA)**

Title: **CRs 29.198-03 Rel-4**

Agenda item: **8.5**

Document for: **Approval**

Doc-1st-Level	Doc-2nd-Level	Spec	CR	Rev	Phase	Subject	Cat	Version-Current	Version-New	Meeting-2nd-Level	Workitem
NP-010466	N5-010676	29.198-03	002		Rel-4	Changing references to JAIN	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010534	29.198-03	003		Rel-4	Update to the definitions of method svcUnavailableInd	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010537	29.198-03	004		Rel-4	Only one subject per method invocation for fault and load management	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010538	29.198-03	005		Rel-4	Fault management is missing some *Err methods	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010539	29.198-03	006		Rel-4	Method balance on Fault management interfaces	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010673	29.198-03	007		Rel-4	Change "TpString" into "TpOctetSets" in authentication and access	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010686	29.198-03	008		Rel-4	Replacement of register/unregisterLoadController	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010688	29.198-03	009		Rel-4	Redundant Framework Heartbeat Mechanism	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010689	29.198-03	010		Rel-4	Add a releaseInterface() method to IpAccess	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010691	29.198-03	011		Rel-4	Removal of serviceID from queryAppLoadReq()	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010695	29.198-03	012		Rel-4	Addition of listInterfaces() method	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010697	29.198-03	013		Rel-4	Introduction and use of new Service Instance ID	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010699	29.198-03	014		Rel-4	P_UNAUTHORISED_PARAMETER_VALUE thrown if non-accessible serviceID is provided	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010703	29.198-03	015		Rel-4	Introduction of Service Instance Lifecycle Management	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010708	29.198-03	016		Rel-4	Add support for multi-vendorship	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010712	29.198-03	017		Rel-4	Removal of P_SERVICE_ACCESS_TYPE	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010713	29.198-03	018		Rel-4	Confusing meaning of prescribedMethod	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010714	29.198-03	019		Rel-4	A client should only have one instance of a given service	F	4.1.0	4.2.0	N5-12	OSA1
NP-010466	N5-010715	29.198-03	020		Rel-4	Some methods on the IpApp interfaces should throw exceptions	F	4.1.0	4.2.0	N5-12	OSA1

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010676**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 002** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

*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:	⌘	Changing references to JAIN			
Source:	⌘	CN5			
Work item code:	⌘	OSA1	***Date:***	⌘	30/08/2001
Category:	⌘	**F**	***Release:***	⌘	REL-4

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:	⌘	Incorrect references to JAIN.
Summary of change:	⌘	Correct references to the JAIN.
Consequences if not approved:	⌘	Potential legal ramifications

Clauses affected:	⌘	1		
Other specs affected:	⌘	☒ Other core specifications □ Test specifications □ O&M Specifications	⌘	All other parts of TS 29.198 Rel-4
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.

1 Scope

The present document is 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 Framework aspects of the interface. All aspects of the Framework are defined in the present document, these being:

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

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

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with a number of JAIN™ Community member companies~~the JAIN consortium~~.

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010534**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**003**	⌘	ev	**-**	⌘	Current version:	**4.1.0** ⌘

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:	⌘	Update to the definitions of method svcUnavailableInd		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	F	***Release:*** ⌘	REL-4

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 descriptions of the svcUnavailableInd method contain inconsistent use of the terms service and service instance when referring to the service instance that becomes unavailable. Within the method description, both terms are used interchangeably as if they were the same thing. In the context of the method, it is understood that the method is meant to refer to an instance of a service, rather than a service. Therefore, some rewording is required to rectify this inconsistency.
Summary of change:	⌘	The descriptions of the methods used to indicate service instance unavailability in the supporting interfaces (IpAppFaultManager and IpFaultManager) lack consistent use of the terms, and therefore should be corrected. Within the descriptions of the methods in each of the interfaces, Lucent proposes to change the word *service* to *service instance* where the service in question is meant to be an instance of it.
Consequences if not approved:	⌘	The description for the method svcUnavailableInd makes inconsistent use of the terms service and service instance. 29.198-3 will be ambiguous and difficult to implement correctly – interworking will be jeopardised. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	8.3.1, 8.3.2	
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.

Resultant changes

8.3.1 Interface Class IpAppFaultManager

Method

`svcUnavailableInd()`

The framework invokes this method to inform the client application that it can no longer use its instance of the indicated service. On receipt of this request, the client application must act to reset its use of the specified service (using the normal mechanisms, such as the discovery and authentication interfaces, to stop use of this service instance and begin use of a different service instance).

8.3.2 Interface Class IpFaultManager

Method

`svcUnavailableInd()`

This method is used by the client application to inform the framework that it can no longer use its instance of the indicated service (either due to a failure in the client application or in the service instance itself). On receipt of this request, the framework should take the appropriate corrective action. The framework assumes that the session between this client application and service instance is to be closed and updates its own records appropriately as well as attempting to inform the service instance and/or its administrator. Attempts by the client application to continue using this session should be rejected.

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010537**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 004** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

*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:	⌘	Only one subject per method invocation for fault and load management
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	30/08/2001
Category:	⌘	**F**
Release:	⌘	REL-4

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 method descriptions for the IpLoadManager and IpFaultManager interfaces, it is stated that the subject of the method can be framework and/or services. However, the definition of the parameters only allows one or the other, not both. Also, the use of the term "service", in some places, should actually be "service instance".
Summary of change:	⌘	Lucent proposes to clarify that these methods can only be used for one subject/entity at a time. Lucent also proposes to modify the sequence diagram in 6.2.7 to clarify that the fault is with the service instance belonging to that client application.
Consequences if not approved:	⌘	The descriptions for the methods on the IpLoadManager and IpFaultManager interfaces will be incorrect and confusing. 29.198-3 will be ambiguous and difficult to implement correctly – interworking will be jeopardised. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	6.2.7, 8.3.1, 8.3.2, and 8.3.8
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.

Resulting changes

Lucent would like to make the following changes to the method descriptions for the IpLoadManager and IpFaultManager and IpAppFaultManager interfaces to clarify that two invocations must be made if the results are required for both the framework and services.

8.3.1 Interface Class IpAppFaultManager

Method

genFaultStatsRecordRes()

This method is used by the framework to provide fault statistics to a client application in response to a genFaultStatsRecordReq method invocation on the IpFaultManager interface.

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

serviceIDs : in TpServiceIDList

Specifies the framework ~~and/~~or services that are included in the general fault statistics record.~~ ~~If the serviceIDs parameter is an empty list, then the fault statistics are for the framework.
~~ The framework is designated by a null value.~~

8.3.2 Interface Class IpFaultManager

Method

activityTestReq()

The application invokes this method to test that the framework or its instance of a service is operational. On receipt of this request, the framework must carry out a test on itself or on the client's instance of the specified service, to check that it is operating correctly. The framework reports the test result by invoking the activityTestRes method on the IpAppFaultManager interface.

Parameters

activityTestID : in TpActivityTestID

The identifier provided by the client application to correlate the response (when it arrives) with this request.

svcID : in TpServiceID

Identifies either the framework or a service for testing. The framework is designated by a null value.

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID

Method

genFaultStatsRecordReq()

This method is used by the application to solicit fault statistics from the framework. On receipt of this request the framework must produce a fault statistics record, for either the framework ~~and/~~or for the client's instances of the specified services during the specified time interval, which is returned to the client application using the genFaultStatsRecordRes operation on the IpAppFaultManager interface.

Parameters

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. A null value leaves this to the discretion of the framework.

serviceIDs : in TpServiceIDList

Specifies either the framework ~~and/~~or services to be included in the general fault statistics record. ~~The framework is designated by a null value.~~ If this parameter is not an empty list, the fault statistics records of the client's instances of the specified services are returned, otherwise the fault statistics record of the framework is returned.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID

8.3.8 Interface Class IpLoadManager

Method

queryLoadReq()

The client application uses this method to request the framework to provide load statistic records for the framework or for its instances of the individual services~~ used by the application~~.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which load statistics records should be reported. If this parameter is not an empty list, the load statistics records of the client's instances of the specified services are returned, otherwise the load statistics record of the framework is returned.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistics records should be reported.

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

Method

registerLoadController()

The client application uses this method to register to receive notifications of load level changes associated with either the framework ~~and/~~or with its instances of the individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework ~~and~~ or SCFs to be registered for load control. To register for framework load control~~ only~~, the serviceIDs ~~is null~~parameter must be an empty list.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID

Method

unregisterLoadController()

The client application uses this method to unregister for notifications of load level changes associated with either the framework ~~and/~~or with its instances of the individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework ~~and/~~or the services for which load level changes should no longer be reported. To unregister for framework load control, the serviceIDs parameter must be an empty list.~~The framework is designated by a null value.~~
Raises

TpCommonExceptions,P_INVALID_SERVICE_ID

Method

resumeNotification()

The client application uses this method to request the framework to resume sending it load management notifications associated with either the framework ~~and/~~or with its instances of the individual services used by the application; e.g. after a period of suspension during which the application handled a temporary overload condition.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework ~~and/~~or the services for which the sending of notifications of load level changes by the framework should be resumed. To resume notifications for the framework, the serviceIDs parameter must be an empty list. ~~The framework is designated by a null value.~~

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

Method

suspendNotification()

The client application uses this method to request the framework to suspend sending it load management notifications associated with either the framework ~~and/~~or with its instances of the individual services used by the application; e.g. while the application handles a temporary overload condition.

Parameters

serviceIDs : in TpServiceIDList

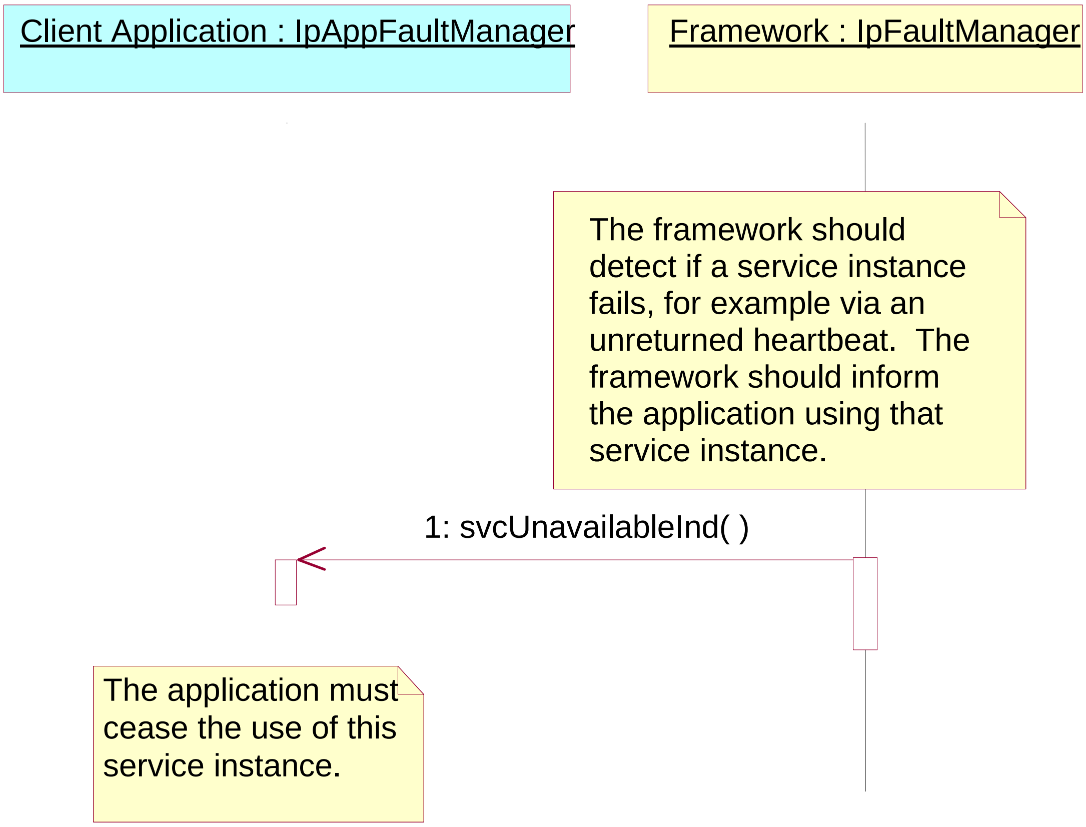
Specifies the framework ~~and/~~or the services for which the sending of notifications by the framework should be suspended. To suspend notifications for the framework, the serviceIDs parameter must be an empty list. ~~The framework is designated by a null value~~

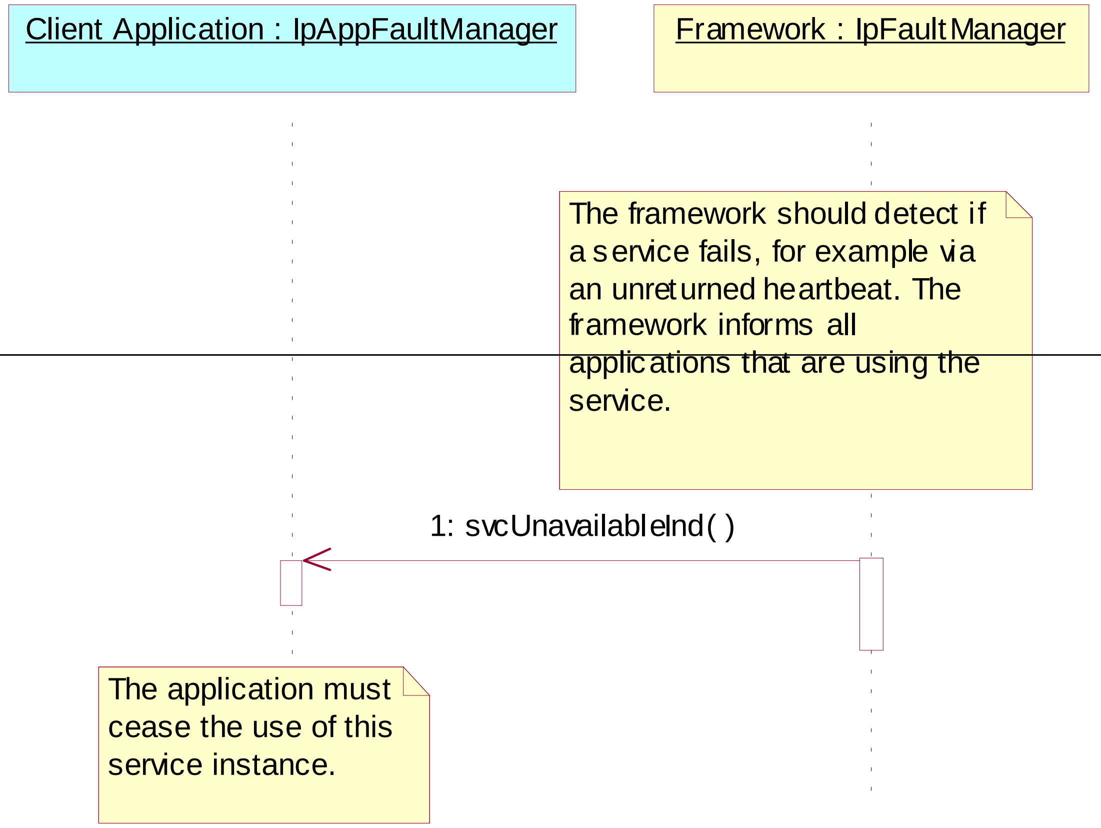
Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

6.2.7 Fault Management: Framework detects a Service failure

The framework has detected that ~~the~~ a service instance has failed (probably by the use of the heartbeat mechanism). The framework updates its own records and informs ~~any~~ the client application~~s~~ ~~that are~~ using the service instance to stop.


Client Application : IpAppFaultManager
Framework : IpFaultManager
The framework should detect if a service instance fails, for example via an unreturned heartbeat. The framework should inform the application using that service instance.
1: svcUnavailableInd( )
The application must cease the use of this service instance.


Client Application : IpAppFaultManager
Framework : IpFaultManager
The framework should detect if a service fails, for example via an unreturned heartbeat. The framework informs all applications that are using the service.
1: svcUnavailableInd( )
The application must cease the use of this service instance.

1: The framework informs ~~each~~ the client application that is using the service instance that the service is unavailable. The client application is then expected to abandon use of this service instance and access a different service instance via the usual means (e.g. discovery, selectService etc.). The client application should not need to re-authenticate in order to discover and use an alternative service instance. The framework will also need to make the relevant updates to its internal records to make sure the service instance is removed from service and no client applications are still recorded as using it.

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010538**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 005** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

*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:	⌘ Fault management is missing some *Err methods		
Source:	⌘ CN5		
Work item code:	⌘ OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘ **F**	***Release:*** ⌘	REL-4

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 fault management interfaces there are two sets of asynchronous methods, genFaultStatsRecordReq and *activityTestReq, which don't have corresponding Err methods. Lucent believes that this creates an imbalance, as the Res methods exist, but the Err methods don't.
method interface	Addition of method genFaultStatsRecordErr is made to all the interfaces which currently have method genFaultStatsRecordRes, along with a new data type TpFaultStatisticsError. An addition of method *activityTestErr is also made to the interfaces which currently have method *activityTestRes.
Consequences if not approved: ⌘	The imbalance of methods is left in the interfaces. There are no methods to correspond to *Err methods., and subsequently there is no way to report an error. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected: ⌘	8.3.1, 8.3.2, and 15.4.2	
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.

Resulting changes

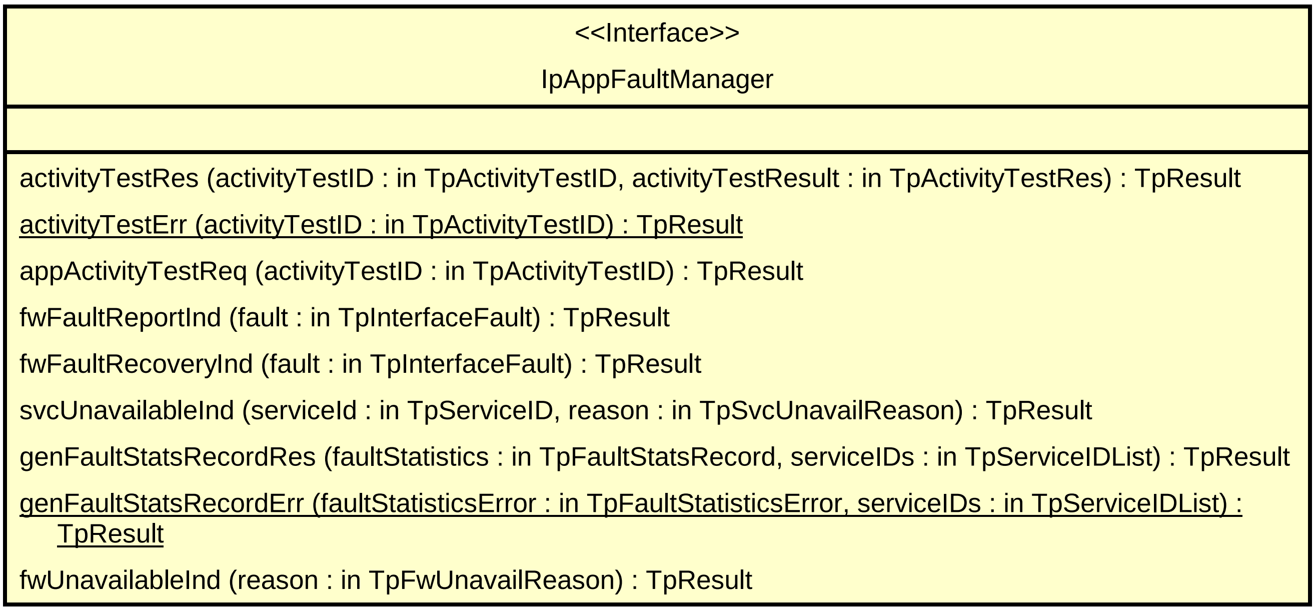
The following data type needs to be added to section 15.4:

15.4.21 TpFaultStatisticsError

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

Name	Value	Description
`P_FAULT_INFO_ERROR_UNDEFINED`	`0`	`Undefined error`
`P_FAULT_INFO_UNAVAILABLE`	`1`	`Fault statistics unavailable`

8.3.1 Interface Class IpAppFaultManager



<<Interface>>

IpAppFaultManager

activityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : TpResult

activityTestErr (activityTestID : in TpActivityTestID) : TpResult

appActivityTestReq (activityTestID : in TpActivityTestID) : TpResult

fwFaultReportInd (fault : in TpInterfaceFault) : TpResult

fwFaultRecoveryInd (fault : in TpInterfaceFault) : TpResult

svcUnavailableInd (serviceId : in TpServiceID, reason : in TpSvcUnavailReason) : TpResult

genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord, serviceIDs : in TpServiceIDList) : TpResult

genFaultStatsRecordErr (faultStatisticsError : in TpFaultStatisticsError, serviceIDs : in TpServiceIDList) : TpResult

fwUnavailableInd (reason : in TpFwUnavailReason) : TpResult

Method

`activityTestErr()`

The framework uses this method to indicate that an error occurred during an application-initiated activity test.

Parameters

`activityTestID : in TpActivityTestID`

Used by the application to correlate this response (when it arrives) with the original request.

Method

`genFaultStatsRecordErr()`

This method is used by the framework to indicate an error fulfilling the request to provide fault statistics, in response to a genFaultStatsRecordReq method invocation on the IpFaultManager interface.

Parameters

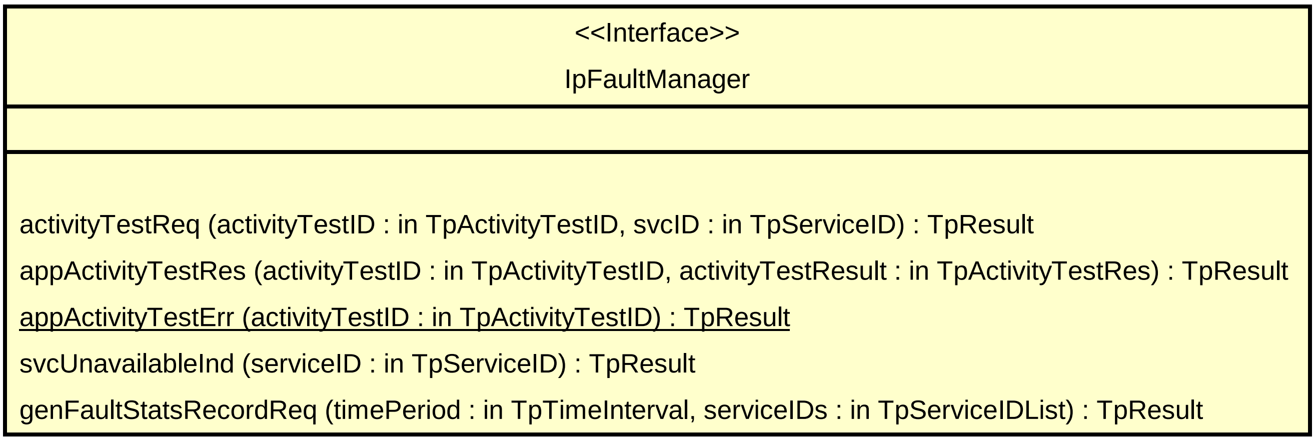
`faultStatisticsError : in TpFaultStatsError`

The fault statistics error.

`serviceIDs : in TpServiceIDList`

Specifies the framework or services that were included in the general fault statistics record request. If the serviceIDs parameter is an empty list, then the fault statistics were requested for the framework.

8.3.2 Interface Class IpFaultManager



<<Interface>>

IpFaultManager

activityTestReq (activityTestID : in TpActivityTestID, svcID : in TpServiceID) : TpResult

appActivityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : TpResult

appActivityTestErr (activityTestID : in TpActivityTestID) : TpResult

svcUnavailableInd (serviceID : in TpServiceID) : TpResult

genFaultStatsRecordReq (timePeriod : in TpTimeInterval, serviceIDs : in TpServiceIDList) : TpResult

Method

`appActivityTestErr()`

The client application uses this method to indicate that an error occurred during a framework-requested activity test.

Parameters

`activityTestID : in TpActivityTestID`

Used by the framework to correlate this response (when it arrives) with the original request.

Raises

`TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID`

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010539**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 006** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

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:	⌘	Method balance on Fault management interfaces		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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 fault management interfaces currently are not very consistent. The presence of the indicator methods is not completely consistent across interfaces, some seem to be missing, resulting in an imbalance.
Summary of change:	⌘	Add method appUnavailableInd to the IpAppFaultManagement and IpFaultManagement interfaces.
Consequences if not approved:	⌘	The fault management interfaces will remain inconsistent with regards to the indicator methods. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	8.3.1 and 8.3.2	
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.

The proposals put forward by this contribution make references to IpFw and IpSvc interfaces that are not currently within the scope of 3GPP but are within the scope of ETSI and Parlay. N5-010705 proposes the introduction of these interfaces into 29.198-3. This CR problem description and proposed solution must be read with the above in mind.

Problem

The fault management interfaces currently are not very consistent in two respects, firstly there seems to be an inconsistent use of a method naming convention, and secondly the presence of methods is not completely consistent across interfaces, a few seem to be missing. The following details the problem discussion.

1. Lack of method naming convention – there are a few methods across the interfaces that convey a common behaviour, yet are named differently. For instance, method svcUnavailableInd on interface IpAppFaultManager, method svcRemovalInd on interface IpFwFaultManager, method svcUnavailableInd on interface IpFaultManager, and method svcUnavailableInd on interface IpSvcFaultManager all share common logical behaviour. However, svcRemovalInd in this case is not named in accordance with a pattern used by the others.

2. Missing methods – some methods on an interface relate directly to other methods on another interface to correlate to define a system-wide behaviour. For example, method svcUnavailableInd on interface IpSvcFaultManager is invoked as a result of method svcUnavailableInd on interface IpFaultManager being invoked. There are two methods in the interfaces that should have corresponding methods as such, but do not. Namely, appUnavailableInd on IpSvcFaultManager, and appRemovalInd on interface IpSvcFaultManager.

The table below summarises the methods on the fault management interfaces that are the subject of the problems discussed. The underline highlights the methods not named in a corresponding way to the others. "***" refers to a missing method, with arrows indicating direction of correlation between methods from different interfaces.

IpAppFaultManagement	*IpFaultManagement*	*IpFwFaultManagement*	*IpSvcFaultManagement*
SvcUnavailableInd ←--- *** ←--------------------	-------------------------------- -------------------------------- svcUnavailableInd --------- *** -------------------------	--- svcRemovalInd --- appUnavailableInd ----------------------------- -----------------------------	 -→ svcUnavailableInd -→ appRemovalInd

Proposal

With regard to naming convention, Lucent proposes to

- Rename method svcRemovalInd on interface IpFwFaultManagement to svcUnavailableInd. According to the description to the method, we believe that *unavailable* is more correctly related to the method's behaviour, as it states " … used by the service to inform the framework that it is about to become *unavailable* for use. ...", while removal refers that the service is removed which is not true.
- Rename method appRemovalInd on IpSvcFaultManagement to appUnavailableInd. Similarly, according to the description to the method, we believe that *unavailable* is more correctly related to the method's behaviour, as it states " … to inform the service that a client application is ceasing its current use of the service …", while removal refers more toward saying that the application is removed.

With regard to method consistency, Lucent proposes to

- Add method appUnavailableInd to interface IpAppFaultManagement, this method is to correspond to the existing method appUnavailableInd on interface IpFwFaultManagement.
- Add method appUnavailableInd to interface IpFaultManagement, this method is to correspond to the existing method appUnavailableInd (was appRemolvalInd before the renaming) on interface IpSvcFaultManagment.

Resultant changes

8.3.1 Interface Class IpAppFaultManager

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

Method

appUnavailableInd()

The framework invokes this method to indicate to the application that the service instance has detected that it is not responding. On receipt of this indication, the application must end its current session with the service instance.

Parameters

No parameters were identified for this method.

8.3.2 Interface Class IpFaultManager

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

Method

appUnavailableInd()

This method is used by the application to inform the framework that the it is ceasing its use of the service instance. This may be a result of the application detecting a failure. The framework assumes that the session between this client application and service instance is to be closed and updates its own records appropriately as well as attempting to inform the service instance and/or its administrator.

Parameters

`serviceID : in TpServiceID`

Identifies the affected application.

Raises

`TpCommonExceptions`

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010673**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 007** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

*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:	⌘	Change "TpString" into "TpOctetSets" in authentication and access		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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:	⌘	It was noted that some data that is to be sent transparently over the API is at the moment still defined as TpString. As a string is not guaranteed to be translated during transmission at the previous meeting in San Diego (May 2001) CN5 introduced a TpOctetSet that maps to a sequence of CORBA octets. This data-type has been accepted to be used for the transparent charging data. However, this data-type is also needed in the authentication and signing of Service Level Agreements.
Summary of change:	⌘	Datatypes of data to be sent transparently over the API will be changed from TpString to TpOctetSet.
Consequences if not approved:	⌘	Authentication and service level agreement signing will not work.

Clauses affected:	⌘	8.1.1, 8.1.2, 8.1.5, 8.1.6	
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:

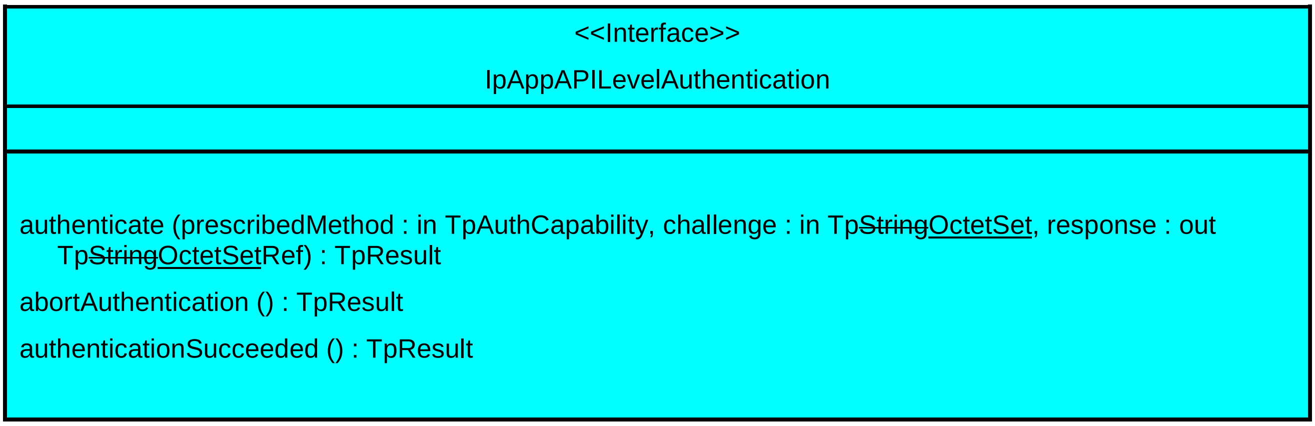
1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

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 IpAppAPILevelAuthentication

Inherits from: IpInterface.



<<Interface>>

IpAppAPILevelAuthentication

authenticate (prescribedMethod : in TpAuthCapability, challenge : in Tp~~String~~OctetSet, response : out Tp~~String~~OctetSetRef) : TpResult

abortAuthentication () : TpResult

authenticationSucceeded () : TpResult

Method

authenticate()

This method is used by the framework to authenticate the client application using the mechanism indicated in prescribedMethod. The client application must respond with the correct responses to the challenges presented by the framework. The number of exchanges and the order of the exchanges is dependent on the prescribedMethod. (These may be interleaved with authenticate() calls by the client application on the IpAPILevelAuthentication interface. This is defined by the prescribedMethod.)

Parameters

prescribedMethod : in TpAuthCapability

see selectEncryptionMethod() on the IpAPIlLevelAuthentication interface. This parameter contains the agreed method for authentication. If this is not the same value as returned by selectEncryptionMethod(), then an error code (P_INVALID_AUTH_CAPABILITY) is returned.

challenge : in Tp~~String~~OctetSet

The challenge presented by the framework to be responded to by the client application. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

response : out Tp~~String~~OctetSetRef

This is the response of the client application to the challenge of the framework in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Method

abortAuthentication()

The framework uses this method to abort the authentication process. This method is invoked if the framework wishes to abort the authentication process, (unless the application responded incorrectly to a challenge in which case no further communication with the application should occur.) If this method has been invoked, calls to the requestAccess operation on IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client application has been properly authenticated.

Parameters

No Parameters were identified for this method

Method

authenticationSucceeded()

The Framework uses this method to inform the client application of the success of the authentication attempt.

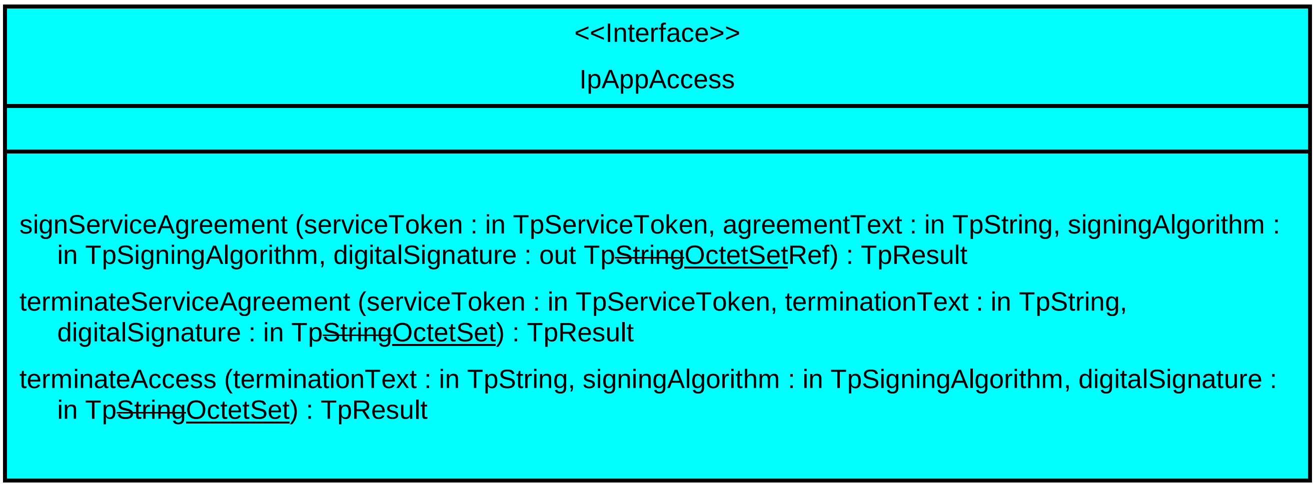
Parameters

No Parameters were identified for this method

4.1.2 Interface Class IpAppAccess

Inherits from: IpInterface.

The Access client application interface is used by the Framework to perform the steps that are necessary in order to allow it to service access.



<<Interface>>

IpAppAccess

signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : out Tp~~String~~OctetSetRef) : TpResult

terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in Tp~~String~~OctetSet) : TpResult

terminateAccess (terminationText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : in Tp~~String~~OctetSet) : TpResult

Method

signServiceAgreement()

This method is used by the framework to request that the client application sign an agreement on the service. It is called in response to the client application calling the selectService() method on the IpAccess interface of the framework. The framework provides the service agreement text for the client application to sign. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application agrees, it signs the service agreement, returning its digital signature to the framework.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance to which this service agreement corresponds. (If the client application selects many services, it can determine which selected service corresponds to the service agreement by matching the service token.) If the serviceToken is invalid, or not known by the client application,then an error code (P_INVALID_SERVICE_TOKEN) is returned.

agreementText : in TpString

This is the agreement text that is to be signed by the client application using the private key of the client application. If the agreementText is invalid, then an error code (P_INVALID_AGREEMENT_TEXT) is returned.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the client application, an error code (P_INVALID_SIGNING_ALGORITHM) is returned.

digitalSignature : out Tp~~String~~OctetSetRef

The digitalSignature is the signed version of a hash of the service token and agreement text given by the framework.

Method

terminateServiceAgreement()

This method is used by the framework to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or unknown to the client application, an error code (P_INVALID_SERVICE_TOKEN) is returned.

terminationText : in TpString

This is the termination text that describes the reason for the termination of the service agreement.

digitalSignature : in Tp~~String~~OctetSet

This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Method

terminateAccess()

The terminateAccess operation is used by the framework to end the client application's access session.

After terminateAccess() is invoked, the client application will no longer be authenticated with the framework. The client application will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail. If at any point the framework's level of confidence in the identity of the client becomes too low, perhaps due to re-authentication failing, the framework should terminate all outstanding service agreements for that client application, and should take steps to terminate the client application's access session WITHOUT invoking terminateAccess() on the client application. This follows a generally accepted security model where the framework has decided that it can no longer trust the application and will therefore sever ALL contact with it.

Parameters

terminationText : in TpString

This is the termination text describes the reason for the termination of the access session.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the client application, an error code (P_INVALID_SIGNING_ALGORITHM) is returned.

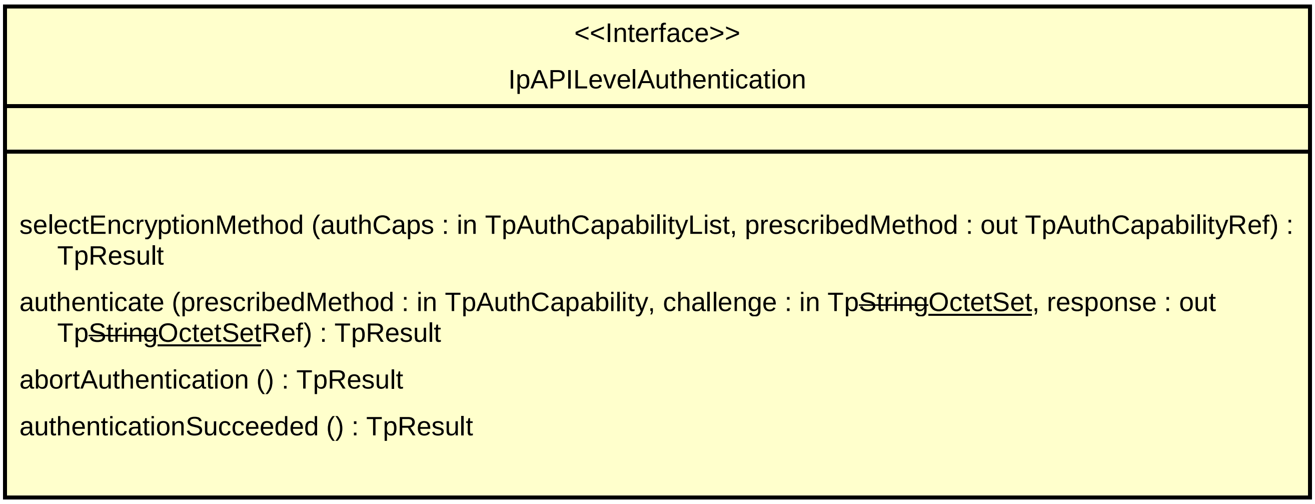
digitalSignature : in Tp~~String~~OctetSet

This is a signed version of a hash of the termination text. The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the access session is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

4.1.3 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

The API Level Authentication Framework interface is used by client application to perform its part of the mutual authentication process with the Framework necessary to be allowed to use any of the other interfaces supported by the Framework.



<<Interface>>

IpAPILevelAuthentication

selectEncryptionMethod (authCaps : in TpAuthCapabilityList, prescribedMethod : out TpAuthCapabilityRef) : TpResult

authenticate (prescribedMethod : in TpAuthCapability, challenge : in Tp~~String~~OctetSet, response : out Tp~~String~~OctetSetRef) : TpResult

abortAuthentication () : TpResult

authenticationSucceeded () : TpResult

Method

selectEncryptionMethod()

The client application uses this method to initiate the authentication process. The framework returns its preferred mechanism. This should be within capability of the client application. If a mechanism that is acceptable to the framework within the capability of the client application cannot be found, the framework returns an error code (P_NO_ACCEPTABLE_AUTH_CAPABILITY).

Parameters

authCaps : in TpAuthCapabilityList

This is the means by which the authentication mechanisms supported by the client application are conveyed to the framework.

prescribedMethod : out TpAuthCapabilityRef

This is returned by the framework to indicate the mechanism preferred by the framework for the authentication process. If the value of the prescribedMethod returned by the framework is not understood by the client application, it is considered a catastrophic error and the client application must abort.

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_NO_ACCEPTABLE_AUTH_CAPABILITY

Method

authenticate()

This method is used by the client application to authenticate the framework using the mechanism indicated in prescribedMethod. The framework must respond with the correct responses to the challenges presented by the client application. The clientAppID received in the initiateAuthentication() can be used by the framework to reference the correct public key for the client application (the key management system is currently outside of the scope of the OSA APIs). The number of exchanges and the order of the exchanges is dependent on the prescribedMethod.

Parameters

prescribedMethod : in TpAuthCapability

see selectEncryptionMethod(). This parameter contains the method that the framework has specified as acceptable for authentication. If this is not the same value as returned by selectEncryptionMethod(), then the framework returns an error code (P_INVALID_AUTH_CAPABILITY).

challenge : in TpOctetSet~~String~~

The challenge presented by the client application to be responded to by the framework. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

response : out Tp~~String~~OctetSetRef

This is the response of the framework to the challenge of the client application in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AUTH_CAPABILITY

Method

abortAuthentication()

The client application uses this method to abort the authentication process. This method is invoked if the client application no longer wishes to continue the authentication process, (unless the application responded incorrectly to a challenge in which case no further communication with the application should occur.) If this method has been invoked, calls to the requestAccess operation on IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client application has been properly authenticated.

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions,P_ACCESS_DENIED

Method

authenticationSucceeded()

The client application uses this method to inform the framework of the success of the authentication attempt.

Parameters

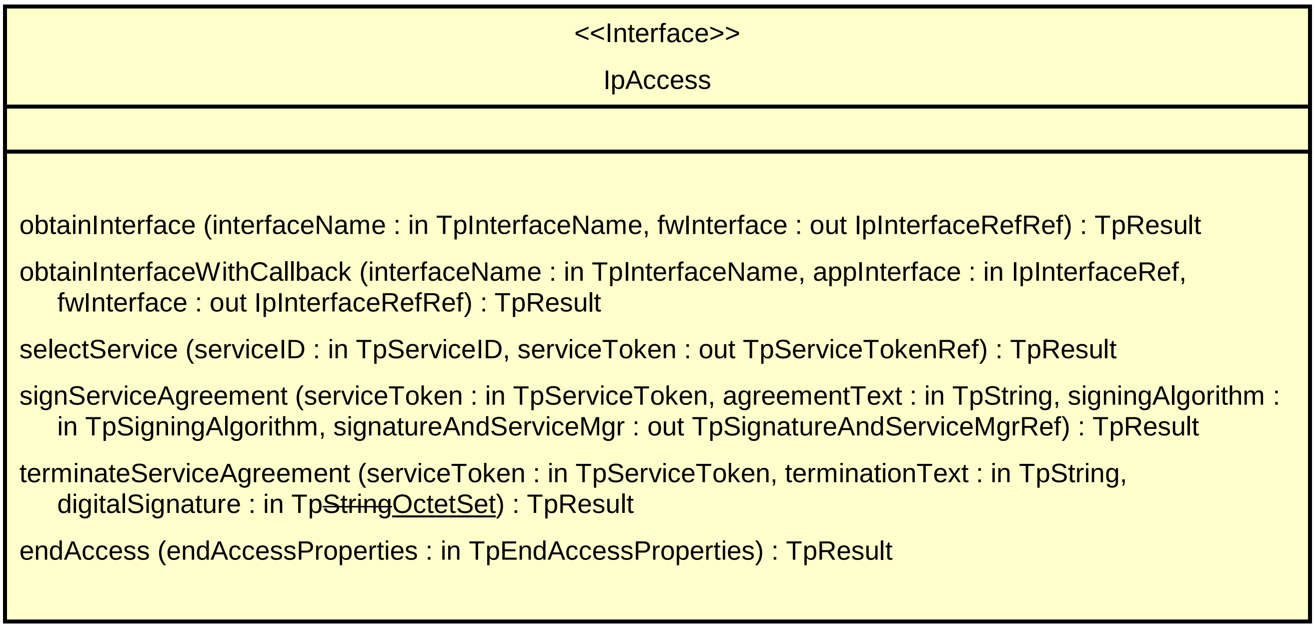
No Parameters were identified for this method

Raises

TpCommonExceptions,P_ACCESS_DENIED

4.1.4 Interface Class IpAccess

Inherits from: IpInterface.



<<Interface>>

IpAccess

obtainInterface (interfaceName : in TpInterfaceName, fwInterface : out IpInterfaceRefRef) : TpResult

obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, appInterface : in IpInterfaceRef, fwInterface : out IpInterfaceRefRef) : TpResult

selectService (serviceID : in TpServiceID, serviceToken : out TpServiceTokenRef) : TpResult

signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm, signatureAndServiceMgr : out TpSignatureAndServiceMgrRef) : TpResult

terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in Tp~~String~~OctetSet) : TpResult

endAccess (endAccessProperties : in TpEndAccessProperties) : TpResult

Method

obtainInterface()

This method is used to obtain other framework interfaces. The client application uses this method to obtain interface references to other framework interfaces. (The obtainInterfacesWithCallback method should be used if the client application is required to supply a callback interface to the framework.)

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

fwInterface : out IpInterfaceRefRef

This is the reference to the interface requested.

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME

Method

obtainInterfaceWithCallback()

This method is used to obtain other framework interfaces. The client application uses this method to obtain interface references to other framework interfaces, when it is required to supply a callback interface to the framework. (The obtainInterface method should be used when no callback interface needs to be supplied.)

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

appInterface : in IpInterfaceRef

This is the reference to the client application interface, which is used for callbacks. If an application interface is not needed, then this method should not be used. (The obtainInterface method should be used when no callback interface needs to be supplied.) If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

fwInterface : out IpInterfaceRefRef

This is the reference to the interface requested.

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME,P_INVALID_INTERFACE_TYPE

Method

selectService()

This method is used by the client application to identify the service that the client application wishes to use. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned.

Parameters

serviceID : in TpServiceID

This identifies the service required. If the serviceID is not recognised by the framework, an error code (P_INVALID_SERVICE_ID) is returned.

serviceToken : out TpServiceTokenRef

This is a free format text token returned by the framework, which can be signed as part of a service agreement. This will contain operator specific information relating to the service level agreement. The serviceToken has a limited lifetime. If the lifetime of the serviceToken expires, a method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will automatically expire if the client application or framework invokes the endAccess method on the other's corresponding access interface.

Raises

TpCommonExceptions,P_ACCESS_DENIED, P_INVALID_SERVICE_ID

Method

signServiceAgreement()

This method is used by the client application to request that the framework sign an agreement on the service, which allows the client application to use the service. If the framework agrees, both parties sign the service agreement, and a reference to the service manager interface of the service is returned to the client application. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance requested by the client application. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

agreementText : in TpString

This is the agreement text that is to be signed by the framework using the private key of the framework. If the agreementText is invalid, then an error code (P_INVALID_AGREEMENT_TEXT) is returned.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the framework, an error code (P_INVALID_SIGNING_ALGORITHM) is returned.

signatureAndServiceMgr : out TpSignatureAndServiceMgrRef

This contains the digital signature of the framework for the service agreement, and a reference to the service manager interface of the service.

```
        structure TpSignatureAndServiceMgr {
              digitalSignature:  TpString;
              serviceMgrInterface:  IpInterfaceRef;
           };
```

The digitalSignature is the signed version of a hash of the service token and agreement text given by the client application.

The serviceMgrInterface is a reference to the service manager interface for the selected service.

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AGREEMENT_TEXT,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED

Method
terminateServiceAgreement()

This method is used by the client application to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or has expired, an error code (P_INVALID_SERVICE_TOKEN) is returned.

terminationText : in TpString

This is the termination text describes the reason for the termination of the service agreement.

digitalSignature : in Tp~~String~~OctetSet

This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement().The framework uses this to check that the terminationText has been signed by the client application. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNATURE

Method
endAccess()

The endAccess operation is used by the client to request that its access session with the framework is ended. After it is invoked, the client application will no longer be authenticated with the framework. The client application will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail.

Parameters

endAccessProperties : in TpEndAccessProperties

This is a list of properties that can be used to tell the framework the actions to perform when ending the access session (e.g. existing service sessions may be stopped, or left running). If a property is not recognised by the framework, an error code (P_INVALID_PROPERTY) is returned.

Raises

TpCommonExceptions,P_ACCESS_DENIED, P_INVALID_PROPERTY

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010686**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**008**	⌘ ev	**-**	⌘ Current version:	**4.1.0** ⌘

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:	⌘ Replacement of register/unregisterLoadController		
Source:	⌘ CN5		
Work item code:	⌘ OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘ **F**	***Release:*** ⌘	REL-4
	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 IpLoadManager interface contains methods called registerLoadController and unregisterLoadController. When the description of these methods is looked at, it can be seen that these methods are concerned with the creation and destruction of requests for load control change notifications.
Summary of change:	⌘	Lucent proposes to rename these methods as createLoadLevelNotification and destroyLoadLevelNotification, following a naming convention set for notifications for FW event notification and also for service-based notifications.
Consequences if not approved:	⌘	The method names will remain inconsistent with the other notification methods within the framework and in the service APIs. 29.198-3 will be ambiguous and difficult to implement correctly – interworking will be jeopardised. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	6.2.5, 8.3.8, and 9.3.3	
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.

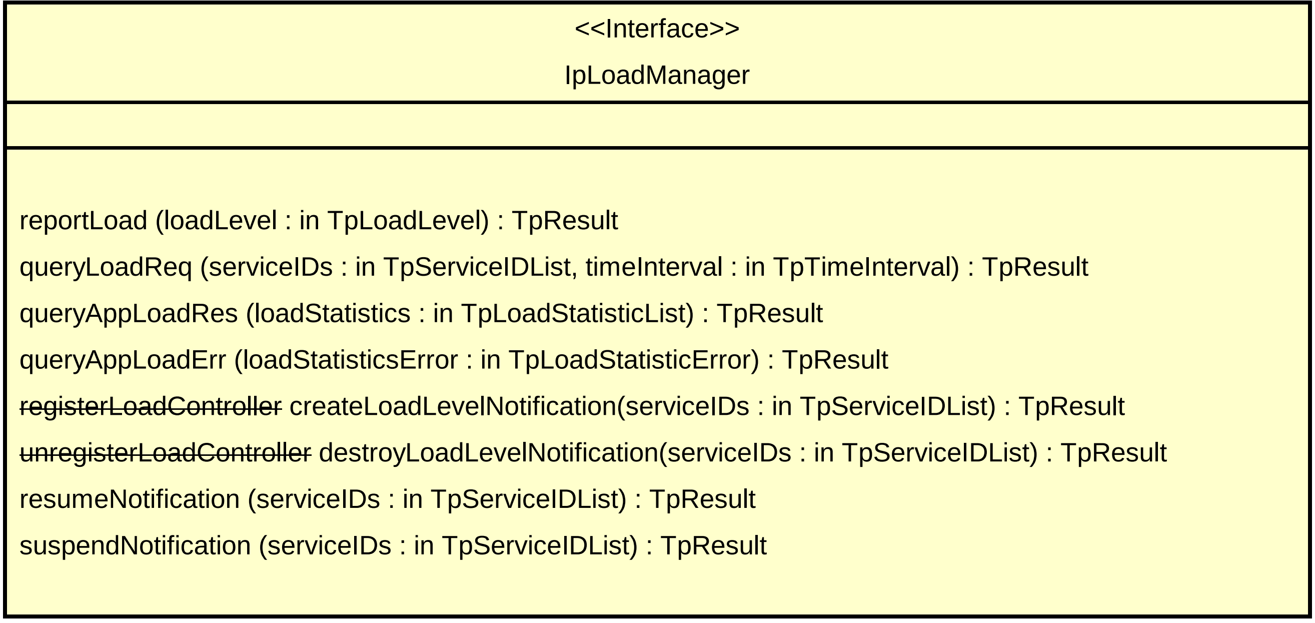
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

Resultant Changes

As a result of these changes, and also the introduction of loadLevelNotification in a previous meeting, the sequence diagram 6.2.5 and the State Transition Diagram in 9.3.3 are changed.

8.3.8 Interface Class IpLoadManager

Inherits from: IpInterface.
The framework API should allow the load to be distributed across multiple machines and across multiple component processes, according to a load management policy. The separation of the load management mechanism and load management policy ensures the flexibility of the load management services. The load management policy identifies what load management rules the framework should follow for the specific client application. It might specify what action the framework should take as the congestion level changes. For example, some real-time critical applications will want to make sure continuous service is maintained, below a given congestion level, at all costs, whereas other services will be satisfied with disconnecting and trying again later if the congestion level rises. Clearly, the load management policy is related to the QoS level to which the application is subscribed. The framework load management function is represented by the IpLoadManager interface. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. To handle responses and reports, the client application developer must implement the IpAppLoadManager interface to provide the callback mechanism. The application supplies the identity of this callback interface at the time it obtains the framework's load manager interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.



<<Interface>>

IpLoadManager

reportLoad (loadLevel : in TpLoadLevel) : TpResult

queryLoadReq (serviceIDs : in TpServiceIDList, timeInterval : in TpTimeInterval) : TpResult

queryAppLoadRes (loadStatistics : in TpLoadStatisticList) : TpResult

queryAppLoadErr (loadStatisticsError : in TpLoadStatisticError) : TpResult

~~registerLoadController~~ createLoadLevelNotification(serviceIDs : in TpServiceIDList) : TpResult

~~unregisterLoadController~~ destroyLoadLevelNotification(serviceIDs : in TpServiceIDList) : TpResult

resumeNotification (serviceIDs : in TpServiceIDList) : TpResult

suspendNotification (serviceIDs : in TpServiceIDList) : TpResult

Method

~~registerLoadController~~createLoadLevelNotification()

The client application uses this method to register to receive notifications of load level changes associated with the framework and/or with individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and SCFs to be registered for load control. To register for framework load control only, the serviceIDs is null.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID

Method

~~unregisterLoadController~~destroyLoadLevelNotification()

The client application uses this method to remove its request for notifications of load level changes associated with the framework and/or with individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which load level changes should no longer be reported. The framework is designated by a null value.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID

6.2.5 Load Management: Application callback registration and load control

This sequence diagram shows how an application registers itself and the framework invokes load management function based on policy.

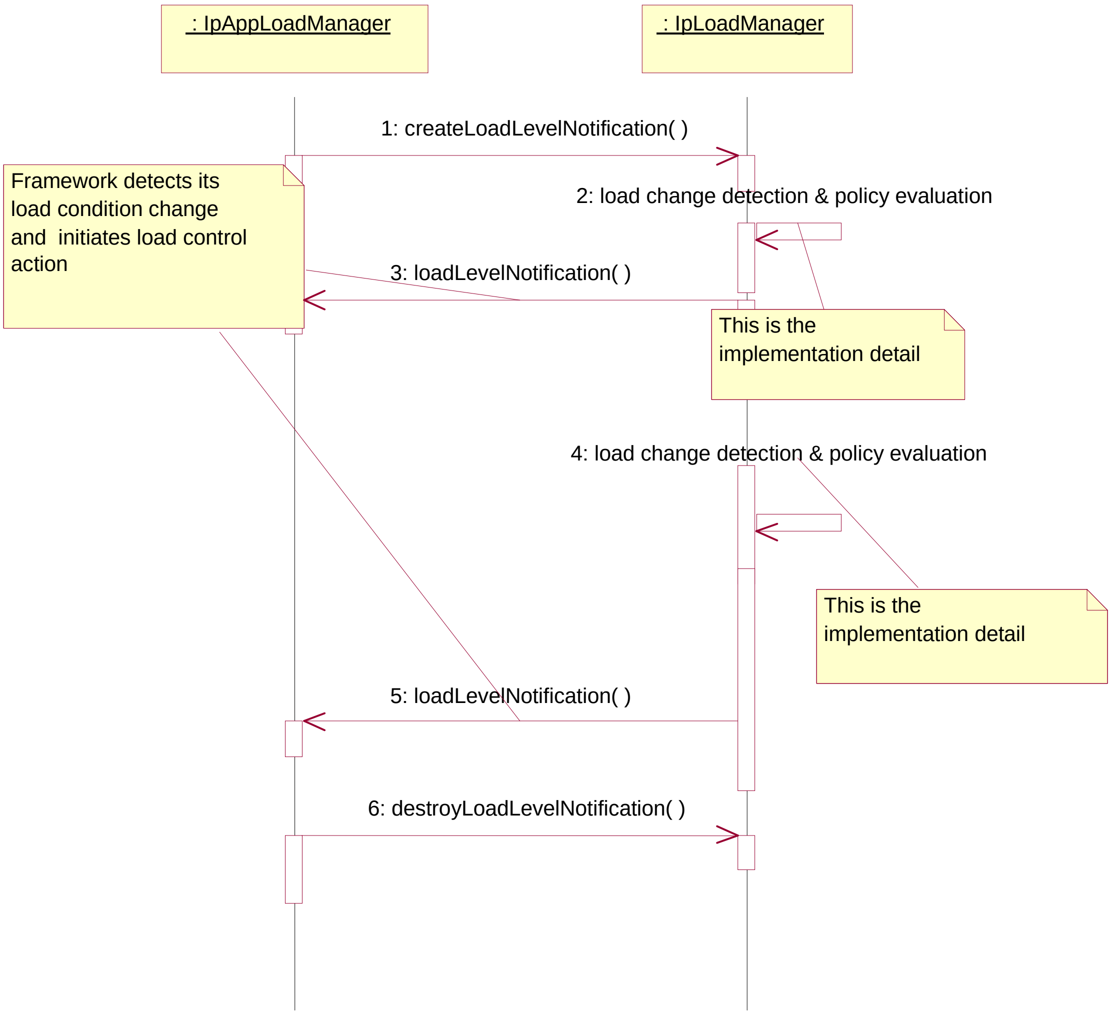

: IpAppLoadManager
: IpLoadManager
1: createLoadLevelNotification( )
Framework detects its load condition change and initiates load control action
2: load change detection & policy evaluation
3: loadLevelNotification( )
This is the implementation detail
4: load change detection & policy evaluation
This is the implementation detail
5: loadLevelNotification( )
6: destroyLoadLevelNotification( )

Figure 1

9.3.3 State Transition Diagram for IpLoadManager

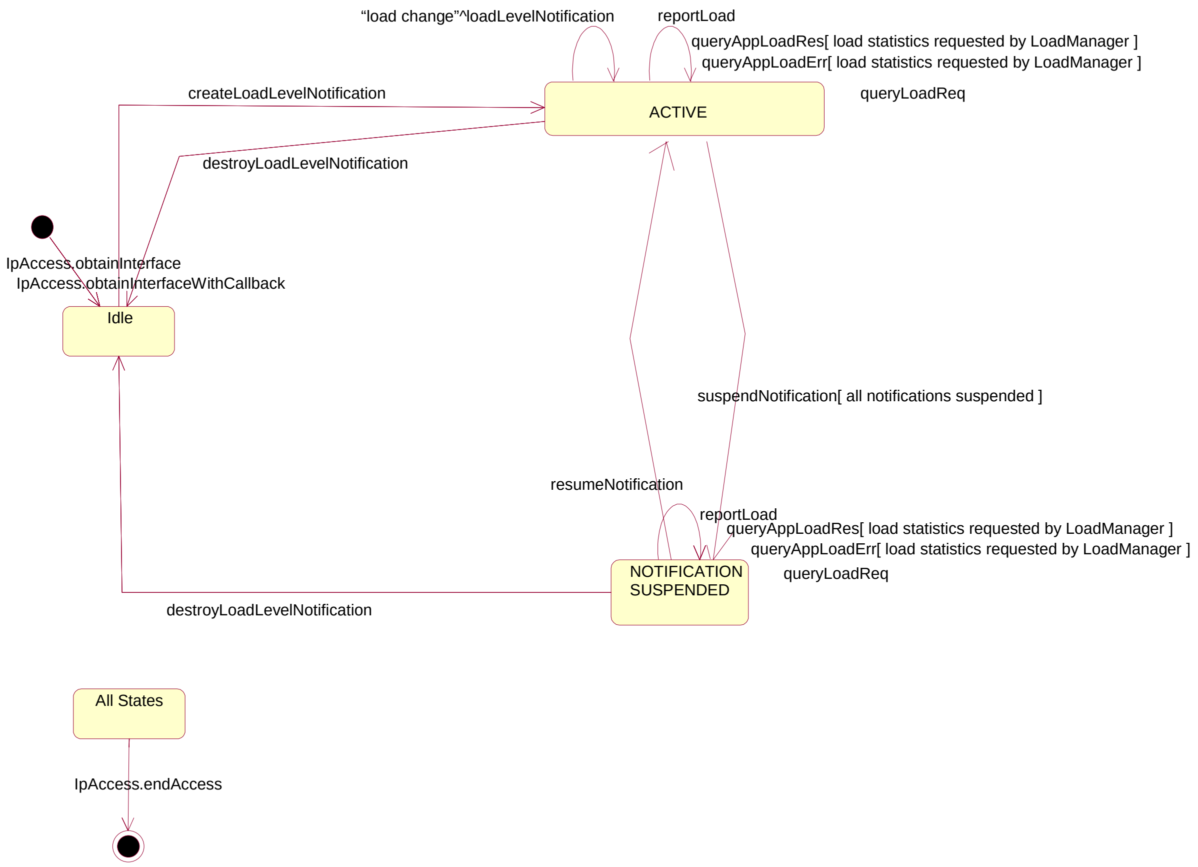

"load change"^loadLevelNotification
reportLoad
queryAppLoadRes[load statistics requested by LoadManager]
queryAppLoadErr[load statistics requested by LoadManager]
queryLoadReq
createLoadLevelNotification
ACTIVE
destroyLoadLevelNotification
IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
suspendNotification[all notifications suspended]
resumeNotification
reportLoad
queryAppLoadRes[load statistics requested by LoadManager]
queryAppLoadErr[load statistics requested by LoadManager]
queryLoadReq
NOTIFICATION SUSPENDED
destroyLoadLevelNotification
All States
IpAccess.endAccess

Figure 2

IDLE
In this state the application has obtained an interface reference to the IpLoadManager from the IpAccess interface.

ACTIVE
In this state the application has indicated its interest in notifications by performing a createLoadLevelNotification() invocation on the IpLoadManager. The load manager can now request the application to supply load statistics information (by invoking queryAppLoadReq()). Furthermore the LoadManager can request the application to control its load (by invoking loadLevelNotification(), resumeNotification() or suspendNotification() on the application side of interface). In case the application detects a change in load level, it reports this to the LoadManager by calling the method reportLoad().

NOTIFICATION SUSPENDED
Due to, e.g. a temporary load condition, the application has requested the load manager to suspend sending the load level notification information.

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010688**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**009**	⌘	ev	**-**	⌘	Current version: **4.1.0** ⌘

*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:	⌘	Redundant Framework Heartbeat Mechanism		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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:	⌘	Current mechanism is redundant
Summary of change:	⌘	The mechanism proposed involves requesting a heartbeat, and then monitoring for that heartbeat. If the beat is not received then it can be decided that the subject of the monitor is no longer running. Policies will then determine the behaviour.
Consequences if not approved:	⌘	The heartbeat mechanism will remain redundant and confusing. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	8.3 and 6.2.6	
Other specs affected:	⌘	□ Other core specifications □ Test specifications □ O&M Specifications	⌘
Other comments:	⌘		

Problem

The current heartbeat mechanism seems unintuitive. In the current mechanism, a service or client application, if it doesn't trust it's availability, can request the framework to monitor it. The result of this is that the framework sends a message to the service/client application at a specified interval. If the service/client application does not respond to this message, they are deemed to have failed the heartbeat and consequently are probably no longer running.

This is effectively a polling mechanism and seems to be redundant. There are already activityTestReq methods on the Ip*FaultManager interfaces which perform exactly this functionality. This leaves two options:

- Remove the heartbeat mechanism from the specification; or
- Rework the heartbeat mechanism into something different and useful.

Also, it doesn't really seem sensible for something to request the monitoring of itself. Why would it do this? If an interface believes that it is not going to perform correctly, then it should use other means of informing the other party, rather than requesting them to keep on checking it periodically.

One of the parameters to the enable*Heartbeat() methods is a duration. This is supposed to represent the interval between heart beats. It is confusing for this parameter to be of type TpDuration, as the definition of TpDuration states that if a –2 value is passed in, then the duration is infinite (definitely no use within a heartbeat mechanism)!

Proposal

Lucent proposes to modify the existing heartbeat functionality so that it is no longer redundant. The mechanism we propose involves requesting a heartbeat, and then monitoring for that heartbeat. If the beat is not received then it can be decided that the subject of the monitor is no longer running. Policies will then determine the behaviour.

We would also like to propose removing the session ID from these interfaces. It doesn't seem to make sense for an entity to have parallel heartbeat sessions with the same framework/service/client application. A method exists to change the heartbeat interval. An entity can request a long interval for a steady background heartbeat, shorten the interval if it desires more frequent pulses, and then restore the previous interval afterwards. There seems to be no need for parallel heartbeat sessions, and the interleaving of heartbeats that would result.

Also, the relationship between an IpHeartBeat(Mgmt) interface and an application is a 1-2-1 relationship (no client application ID is passed across the interface). Every time a new heartbeat interface is obtained using obtainInterfaceWithCallBack() a new interface instance will be created.

Resulting changes

Lucent would like to modify the description of the heartbeat process.

Lucent would like to rename the send() method on IpHeartBeat and IpAppHeartBeat to something more representative of its function. We would propose renaming it to pulse().

We would like to change the duration parameter in the enable*Heartbeat() methods to "interval" of type TpInt32.

Lucent would also like to remove the sessionID parameter from the interfaces.

We feel that the State Transition Diagrams given in sections 9.3.1 and 9.3.2 of 29.198-3 do not add any extra information and that these interfaces are not really state-based, anyway. We therefore propose removing these interfaces from the specification.

8.3.3 Interface Class IpAppHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the ~~Framework~~ client application by the ~~Client application~~framework. ~~Since the OSA APIs are inherently synchronous, the heartbeats themselves are synchronous for~~

~~efficiency reasons. The return of the TpResult is interpreted as a heartbeat response.~~

<<Interface>>

IpAppHeartBeatMgmt

enableAppHeartBeat (~~duration~~ interval : in ~~TpDuration~~TpInt32, fwInterface : in IpHeartBeatRef~~, session : in TpSessionID~~) : TpResult

disableAppHeartBeat (~~session : in TpSessionID~~) : TpResult

~~changeTimePeriod~~ changeInterval (~~duration~~ interval : in ~~TpDuration~~TpInt32~~, session : in TpSessionID~~) : TpResult

Method

enableAppHeartBeat()

With this method, the framework ~~registers at~~instructs the client application ~~for heartbeat supervision of itself~~to begin sending its heartbeat to the specified interface at the specified interval.

Parameters

~~duration~~ interval : in ~~TpDuration~~TpInt32

The time interval in milliseconds between the heartbeats.

fwInterface : in IpHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

~~session : in TpSessionID~~

~~Identifies the heartbeat session.~~

Method

disableAppHeartBeat()

~~Allows~~ Instructs the ~~stop of the heartbeat supervision of the application~~client application to cease the sending of its heartbeat.

Parameters

~~session : in TpSessionID~~

~~Identifies the heartbeat session.~~None identified.

Method

~~changeTimePeriod~~changeInterval()

Allows the administrative change of the heartbeat ~~period~~interval.

Parameters

~~duration~~ interval : in ~~TpDuration~~TpInt32

The time interval in milliseconds between the heartbeats.

~~session : in TpSessionID~~

~~Identifies the heartbeat session.~~

8.3.4 Interface Class IpAppHeartBeat

Inherits from: IpInterface.

The Heartbeat Application interface is used by the Framework to ~~supervise~~ send the client a~~A~~pplication its heartbeat. ~~The return of the TpResult is interpreted as a heartbeat response.~~

<<Interface>>

IpAppHeartBeat

~~send~~ pulse (~~session : in TpSessionID~~) : TpResult

Method

~~send~~pulse()

Th~~is is the method the~~ framework uses ~~in case it supervises the client application. The sender must raise an exception if no result comes back after a certain, user-defined time..~~ this method to send its heartbeat to the client application. The application will be expecting a pulse at the end of every interval specified in the parameter to the IpHeartBeatMgmt.enableHeartbeat() method. If the pulse() is not received within the specified interval, then the framework can be deemed to have failed the heartbeat.

Parameters

~~session : in TpSessionID~~

~~Identifies the heartbeat session.~~None identified.

8.3.5 Interface Class IpHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the ~~client application~~framework by a client application.~~ Since the APIs are inherently synchronous, the heartbeats themselves are synchronous for efficiency reasons. The return of the TpResult is interpreted as a heartbeat response.~~

<<Interface>>

IpHeartBeatMgmt

enableHeartBeat (~~duration~~ interval : in ~~TpDuration~~TpInt32, appInterface : in IpAppHeartBeatRef~~, session : out TpSessionIDRef~~) : TpResult

disableHeartBeat (~~session : in TpSessionID~~) : TpResult

~~changeTimePeriod~~ changeInterval (~~duration~~ interval : in ~~TpDuration~~TpInt32~~, session : in TpSessionID~~) : TpResult

Method

enableHeartBeat()

With this method, the client application ~~registers at~~instructs the framework ~~for heartbeat supervision of itself~~to begin sending its heartbeat to the specified interface at the specified interval.

Parameters

~~duration~~ interval : in ~~TpDuration~~TpInt32

The ~~duration~~time -interval in milliseconds between the heartbeats.

appInterface : in IpAppHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

~~session : out TpSessionIDRef~~

~~Identifies the heartbeat session. In general, the application has only one session. In case of framework supervision by the client application (see the application interfaces), the application may maintain more than one session.~~

Raises

TpCommonExceptions~~, P_INVALID_SESSION_ID~~

Method

disableHeartBeat()

~~Allows the stop of the heartbeat supervision of the application~~Instructs the framework to cease the sending of its heartbeat.

Parameters

~~session : in TpSessionID~~

~~Identifies the heartbeat session.~~None identified.

Raises

TpCommonExceptions~~, P_INVALID_SESSION_ID~~

Method

~~changeTimePeriod~~changeInterval()

Allows the administrative change of the heartbeat ~~period~~interval.

Parameters

~~duration~~ interval : in ~~TpDuration~~TpInt32

The time interval in milliseconds between the heartbeats.

~~session : in TpSessionID~~

~~Identifies the heartbeat session.~~

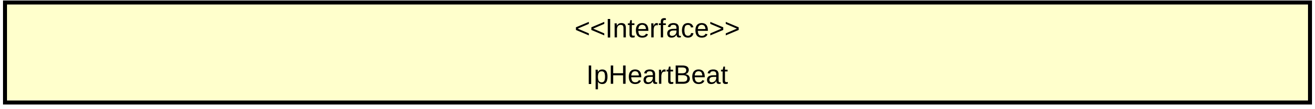
Raises

TpCommonExceptions~~, P_INVALID_SESSION_ID~~

8.3.6 Interface Class IpHeartBeat

Inherits from: IpInterface.

The Heartbeat Framework interface is used by the client application to ~~supervise the Framework~~send its heartbeat.



~~send~~ pulse (~~session : in TpSessionID~~) : TpResult

Method

~~send~~pulse()

~~This is the method the client application uses in case it supervises the framework. The sender must raise an exception if no result comes back after a certain, user-defined time.~~ The client application uses this method to send its heartbeat to the framework. The framework will be expecting a pulse at the end of every interval specified in the parameter to the IpAppHeartBeatMgmt.enableAppHeartbeat() method. If the pulse() is not received within the specified interval, then the framework can be deemed to have failed the heartbeat.

Parameters

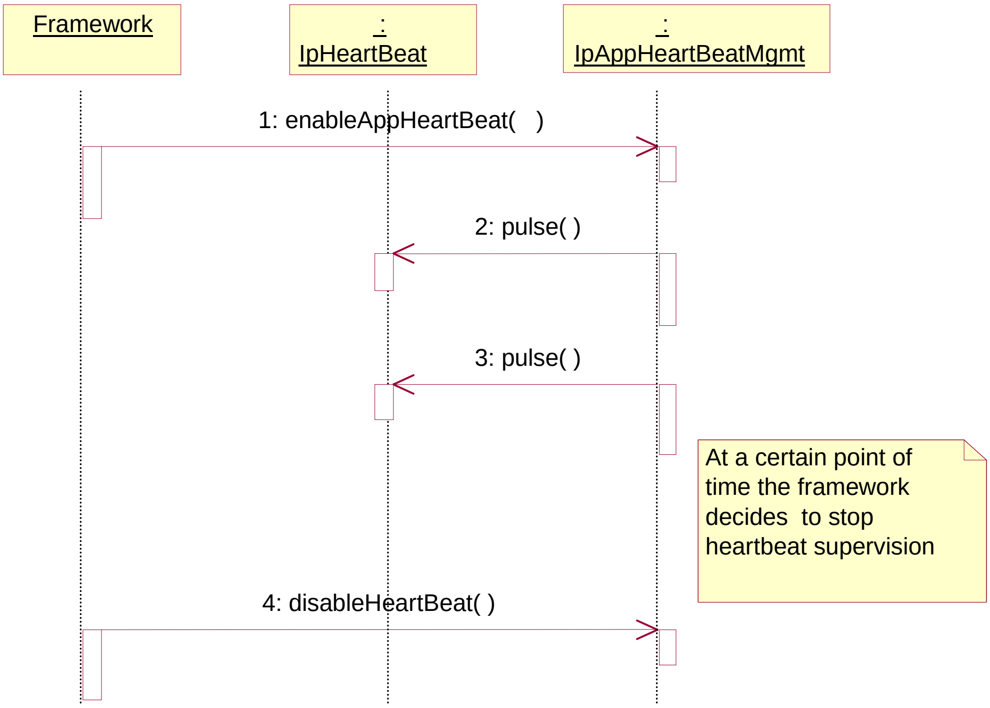
~~session : in TpSessionID~~

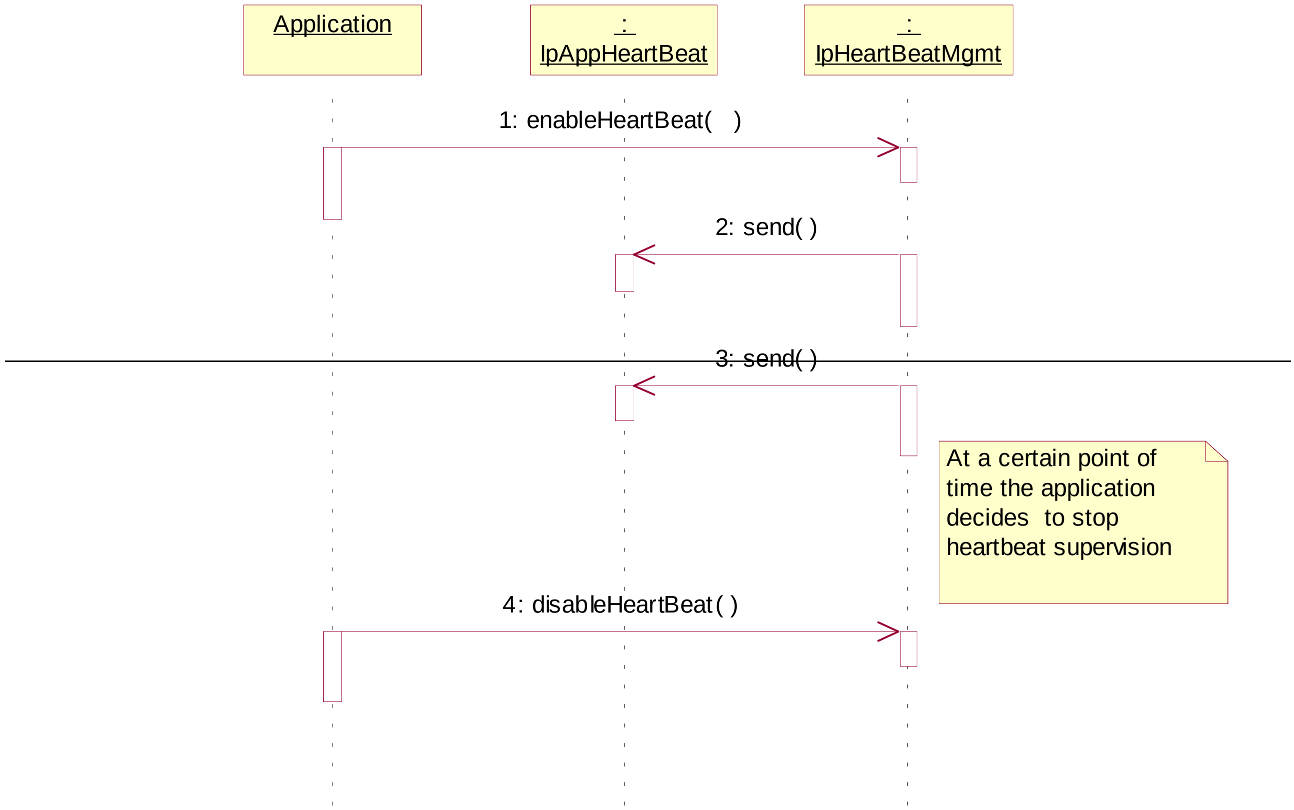
~~Identifies the heartbeat session. In general, the application has only one session.~~ None identified.

Raises

TpCommonExceptions

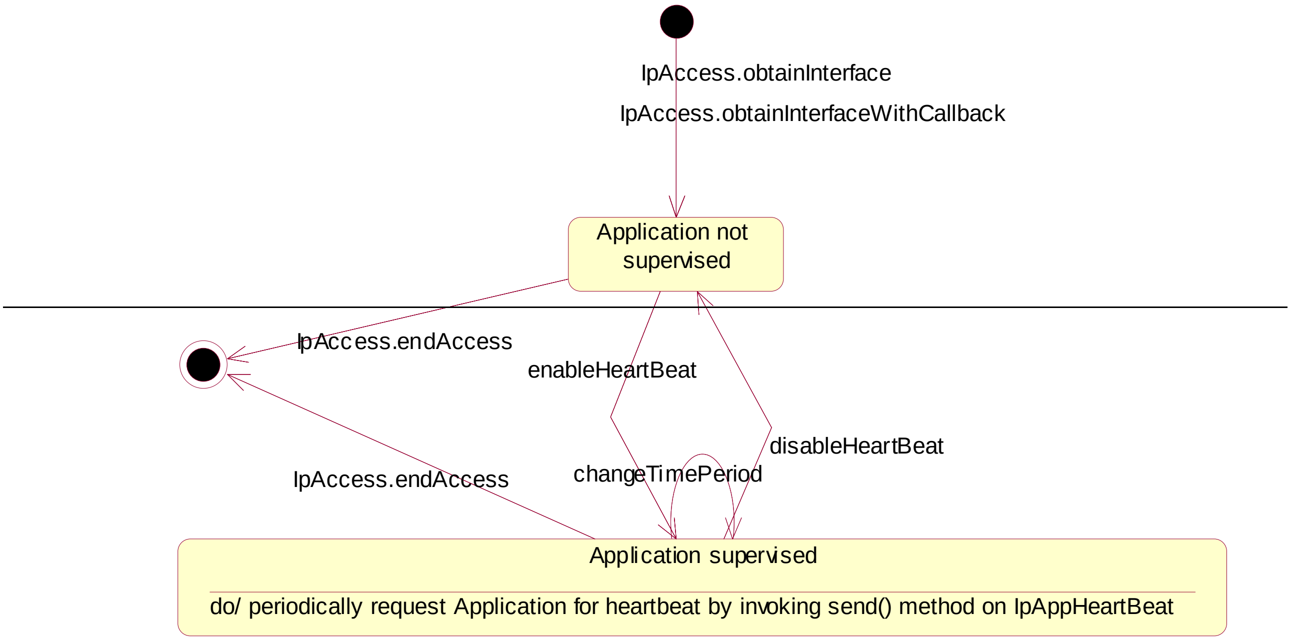
6.2.6 Heartbeat Management: Start/perform/end heartbeat supervision ~~of~~ of the application


Framework
: IpHeartBeat
: IpAppHeartBeatMgmt
1: enableAppHeartBeat( )
2: pulse( )
3: pulse( )
At a certain point of time the framework decides to stop heartbeat supervision
4: disableHeartBeat( )



In this sequence diagram, the framework has decided that it wishes to monitor the application, and has therefore requested the application to commence sending its heartbeat. The application responds by sending its heartbeat at the specified interval. The framework then decides that it is satisfied with the application's health and disables the heartbeat mechanism. If the heartbeat was not received from the application within the specified interval, the framework can decide that the application has failed the heartbeat and can then perform some recovery action.

~~9.3.1 State Transition Diagrams for IpHeartBeatMgmt~~


IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Application not supervised
IpAccess.endAccess
enableHeartBeat
disableHeartBeat
changeTimePeriod
IpAccess.endAccess
Application supervised
do/ periodically request Application for heartbeat by invoking send() method on IpAppHeartBeat

~~Figure : State Transition Diagram for IpHeartBeatMgmg~~

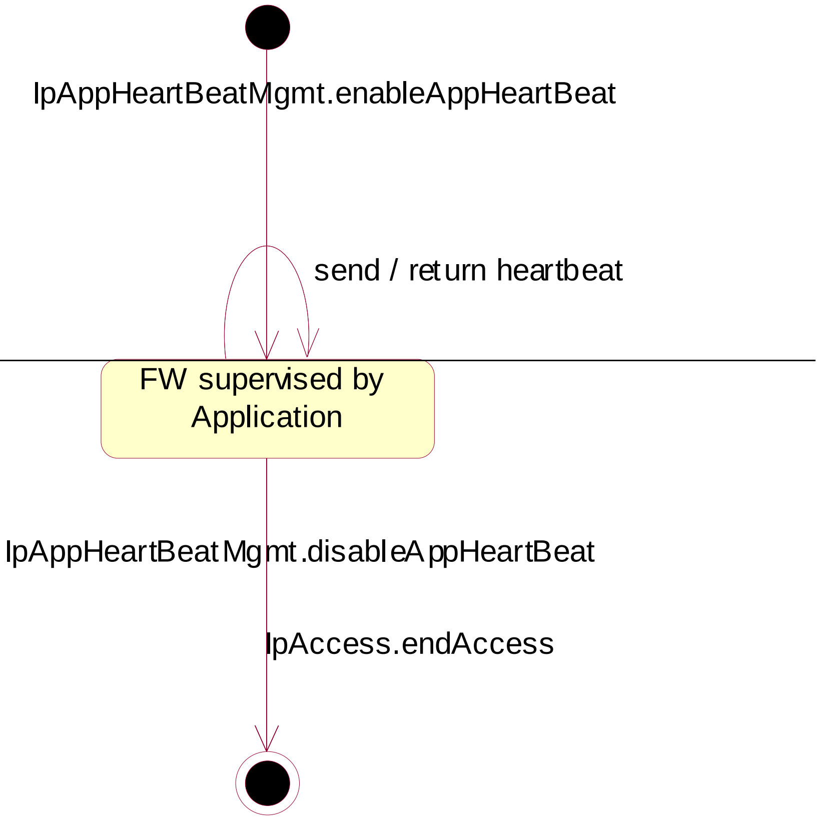
~~9.3.1.1 Application not supervised State~~

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

~~9.3.1.2 Application supervised State~~

~~In this state the application has registered for heartbeat supervision by the Framework. Periodically the Framework will request for the application heartbeat by calling the send method on the IpAppHeartBeat interface.~~

~~9.3.2 State Transition Diagrams for IpHeartBeat~~


IpAppHeartBeatMgmt.enableAppHeartBeat
send / return heartbeat
FW supervised by Application
IpAppHeartBeatMgmt.disableAppHeartBeat
IpAccess.endAccess

~~Figure : State Transition Diagram for IpHeatBeat~~

~~9.3.2.1FW supervised by Application State~~

~~In this state the Framework has requested the application for heartbeat supervision on itself. Periodically the application calls the send() method and the Framework returns it's heartbeat result.~~

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010689**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 010** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

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 a releaseInterface() method to IpAccess		
Source: ⌘	CN5		
Work item code: ⌘	OSA1	***Date:*** ⌘	30/08/2001
Category: ⌘	**F**	***Release:*** ⌘	REL-4

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: ⌘	An entity using the framework currently uses obtainInterface or obtainInterfaceWithCallback to obtain interfaces such as the load management interfaces, or the heartbeat interfaces. Currently the only way in which these interfaces can be released is by ending the access session. When the access session is ended, all interface references given out during the session are no longer accessible. However, an entity using the framework may not need or desire to keep ALL of the interfaces it has obtained for the entire lifetime of an access session, which could last quite some time. This is holding framework resources unnecessarily.
Summary of change: ⌘	Lucent proposes the addition of a releaseInterface method to the IpAccess and IpFwAccess interfaces. This will allow the entity using the framework to be able to release any of the interfaces it has obtained at any point during the access session, thus removing the need to hold the interfaces (and the accompanying resources) unnecessarily.
Consequences if not approved: ⌘	Framework resources will be retained unnecessarily until the ending of the access session. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected: ⌘	8.1.6	
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.

Resulting changes

One new method and its description should be added to the IpAccess interface.

8.1.6 Interface Class IpAccess

Inherits from: IpInterface.

<<Interface>>

IpAccess

obtainInterface (interfaceName : in TpInterfaceName, fwInterface : out IpInterfaceRefRef) : TpResult

obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, appInterface : in IpInterfaceRef, fwInterface : out IpInterfaceRefRef) : TpResult

selectService (serviceID : in TpServiceID, serviceToken : out TpServiceTokenRef) : TpResult

signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm, signatureAndServiceMgr : out TpSignatureAndServiceMgrRef) : TpResult

terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpString) : TpResult

endAccess (endAccessProperties : in TpEndAccessProperties) : TpResult

releaseInterface(interfaceName : in TpInterfaceName) : TpResult

Method

releaseInterface()

The client application uses this method to release a framework interface that was obtained during this access session.

Parameters

interfaceName : in TpInterfaceName

This is the name of the framework interface which is being released. If the interfaceName is invalid, the framework throws the P_INVALID_INTERFACE_NAME exception. If the interface has not been given to the client application during this access session, then the P_TASK_REFUSED exception will be thrown.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010691**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-3**	**CR**	**011**	⌘	ev	**-**	⌘	Current version:	**4.1.0**	⌘

*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:	⌘	Removal of serviceID from queryAppLoadReq()		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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:	⌘	One of the parameters to queryAppLoadReq() in IpAppLoadManager interface is *serviceIDs* of type TpServiceIDList, which the specification currently states as representing “the application and/or the services for which load statistic records should be reported”. However, there is no need for this parameter as it makes little sense for the Framework to request an APL to provide Load statistics records for the services it is using. The requested information would have been passed to the APL via the Framework itself anyway, and therefore the Framework must already be aware of this. If it doesn’t have this information then it would be a better approach for it to request the information from the service instances directly.
Summary of change:	⌘	*serviceIDs* parameter is removed from IpAppLoadManager.queryAppLoadReq().
Consequences if not approved:	⌘	*ServiceIDs* parameter in IpAppLoadManager.queryAppLoadReq() is has no function and confuses the API. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	8.3.7		
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:

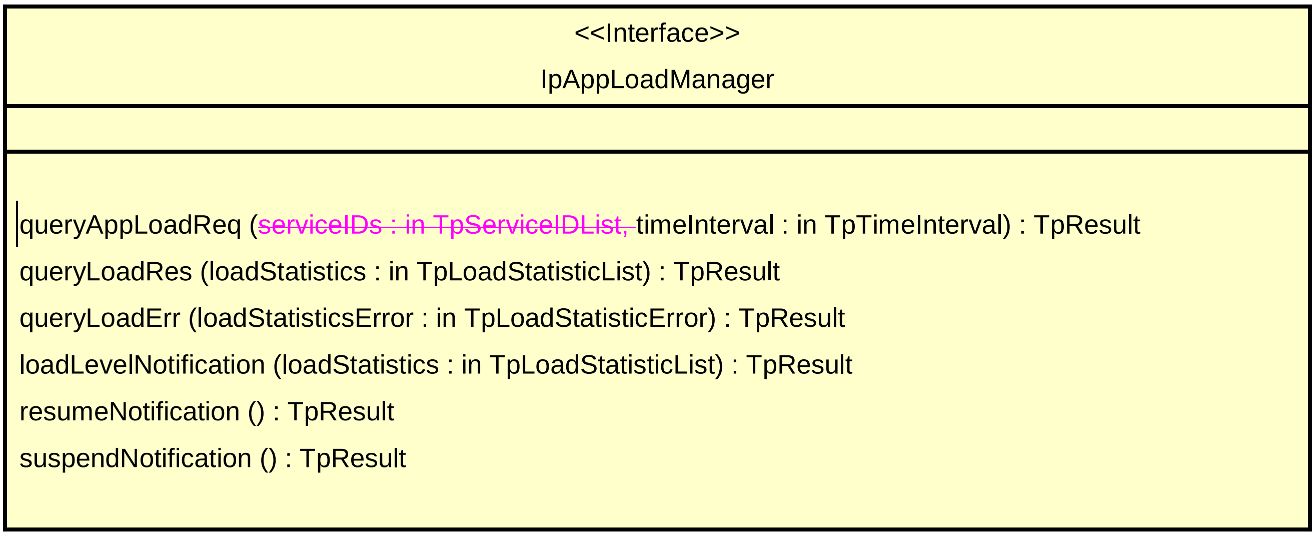
1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

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.

Resultant changes

8.3.7 Interface Class IpAppLoadManager



<<Interface>>

IpAppLoadManager

queryAppLoadReq (~~serviceIDs : in TpServiceIDList,~~ timeInterval : in TpTimeInterval) : TpResult

queryLoadRes (loadStatistics : in TpLoadStatisticList) : TpResult

queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : TpResult

loadLevelNotification (loadStatistics : in TpLoadStatisticList) : TpResult

resumeNotification () : TpResult

suspendNotification () : TpResult

Method

`queryAppLoadReq()`

The framework uses this method to request the application to provide load statistic records for the application ~~and/or for individual services used by the application~~.

Parameters

~~`serviceIDs : in TpServiceIDList`~~

~~Specifies the application and/or the services for which load statistic records should be reported. The application is designated by a null value.~~

`timeInterval : in TpTimeInterval`

Specifies the time interval for which load statistic records should be reported.

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010695**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**012**	⌘	ev	**-**	⌘	Current version:	**4.1.0** ⌘

*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:	⌘	Addition of listInterfaces() method
Source:	⌘	CN5
Work item code:	⌘	OSA1
Date:	⌘	30/08/2001
Category:	⌘	**F**
Release:	⌘	REL-4

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:	⌘	Assuming a client is aware of the interfaces that the framework supports, then the client is able to obtain the interface reference using obtainInterface()/obtainInterfaceWithCallback with the interfaceName as the parameter. However, the client only has knowledge of the availability of the pre-arranged interfaces and currently has no way to discover dynamically which interfaces are actually supported by the framework.
method interface		method listInterfaces() along with its description are added to IpAccess. TpInterfaceNameList type is also introduced.
Consequences if not approved:	⌘	A client application is not able to discover interfaces at run time, and therefore has access to only pre-arranged interfaces. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	Chapter 8.1.6 and 15.1	
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.

Proposal

The proposal is to add a listInterfaces() method to the IpAccess interface, which would enable the client to obtain a list of the interfaces supported by the framework. This prevents the client from having to invoke obtainInterface() for the interface they want in the hope that it is available to them.

Resultant Changes

The effect of this proposal is a change to the IpAccess interface and Framework Data Definitions. One new method and description are added. One new data definition is added.

8.1.6 Interface Class IpAccess

Interface Class IpAccess

<<Interface>> IpAccess
obtainInterface(interfaceName: in TpInterfaceName, fwInterface: out IparlayInterfaceRefRef): TpResult obtainInterfaceWithCallback(interfaceName: in TpInterfaceName, appInterface: in IparlayInterfaceRef, fwInterface: out IparlayInterfaceRefRef): TpResult selectService(serviceID: in TpServiceID, serviceToken: out TpServiceTokenRef): TpResult signServiceAgreement(serviceToken: in TpServiceToken, agreementText: in TpString, signingAlgorithm: in TpSigningAlgorithm, signatureAndServiceMgr: out TpSignatureAndServiceMgrRef): TpResult terminateServiceAgreement(serviceToken: in TpServiceToken, terminationText: in TpString, digitalSignature: in TpString): TpResult endAccess(endAccessProperties: in TpEndAccessProperties) : TpResult listInterfaces(frameworkInterfaces: out TpInterfaceList) : TpResult

Method

`listInterfaces()`

The client application uses this method to obtain the names of all interfaces supported by the framework. It can then obtain the interfaces it wishes to use using either obtainInterface() or obtainInterfaceWithCallback().

Parameters

`frameworkInterfaces : out TpInterfaceNameList`

The frameworkInterfaces parameter contains a list of interfaces that the framework makes available.

Raises

`TpCommonExceptions, P_ACCESS_DENIED`

15.1.33 TpInterfaceNameList

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

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010697**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 013** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

*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:	⌘	Introduction and use of new Service Instance ID		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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 individual service instances need to set up access sessions with the Framework and obtain separate Integrity Management interfaces. This then allows them to provide load, fault and heartbeat information to the Framework on an individual basis. Currently, this is not possible, and the specification is confused over what should be performed on a per-service basis and what should be performed on a per-service instance basis.
Summary of change:	⌘	This requires that a new serviceInstanceID is introduced. The serviceInstanceID is generated by the Framework and passed to the Service Factory. It is used when the service instance exchanges integrity management interfaces with the framework. This service instance ID will be used by the framework to correlate requests for service integrity management statistics with the service instance for that client application.
Consequences if not approved:	⌘	29.198-3 will be ambiguous and difficult to implement correctly – interworking will be jeopardised. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	15.1, 12.1.1	
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.

Proposal to introduce a new Service Instance ID data type and use it in place of serviceID for authentication and access

Problem Description

It is unclear from the specification what the nature of the relationship is between service instances and the Framework.

One view is that the Framework is unaware of the existence of the instances themselves and only has knowledge of the “service”. Essentially this means that an entity representing the service (and identified by the serviceID) sets up an access session with the Framework and obtains all of the Integrity Management interfaces.

The problem is that the Framework cannot then identify which instance of a service an invocation of Integrity Management methods belongs on and therefore information such as load level and fault statistics can only apply collectively to all instances of a service. In addition, if an application wants to indicate that it can’t use an instance and therefore wants to end its session the “service” has to determine which instance is being terminated as the Framework has no way of doing this.

The other view is that the individual instances set up access sessions with the Framework and obtain separate Integrity Management interfaces. This allows them to provide load, fault and heartbeat information to the Framework on an individual basis.

Solution

We believe that the Integrity Management functionality must happen on a per-instance basis for useful information to be available to the application that is using the service instance. Therefore the instance needs to have an access session available when it is created.

To do this it will need to authenticate with the Framework (using keys that perhaps have been passed to it by the Factory on creation) and then requestAccess. It will need to identify itself to the Framework in order that the Framework can determine which interfaces belong to which instance.

This requires that a new serviceInstanceID is introduced. The serviceInstanceID is generated by the Framework and passed to the Service Factory.

It could either be used in the intiateAuthentication call in place of serviceID, or it could be added to either the requestAccess or obtainInterface* methods, as outlined in the options below.

Option 1 – Use serviceInstanceID in initiateAuthentication

The Framework would need to correlate the serviceInstanceID to a key in order to complete the authentication process. This could possibly be done by only assigning keys at the serviceID domain level, thus using the same key for all instances.

Option 2 – Add serviceInstanceID to requestAccess

In other words, the serviceID is used to authenticate but then the access session being started by the service instance is identified to the Framework using the instance ID. From then on, any interfaces obtained by the instance can be correlated by the Framework.

It is possible that a single entity representing all of the interfaces could do the authentication (eg the Factory) and then pass the authentication interface reference to each of the instances it creates.

One drawback to this approach is that the method signature for requestAccess changes and the new version is only appropriate on the FW-SVC side. Another possible issue is what happens if the single entity becomes no longer authenticated and all of the access sessions are killed? In that case the single entity would need to be able to inform the instances.

Option 3 – Add serviceInstanceID to obtainInterface*

Again, the serviceID is used to authenticate but requestAccess is effectively done under the guise of the serviceID. The instance ID is then only used as each required interface is obtained.

It would be possible for a single entity to set up a single access session and provide a reference to the FW interface to each instance (again at creation of the instance). Again a problem arises if the access session is lost, as all interfaces obtained during that session are also lost. The single entity would require the ability to inform the instances and give them a new access interface reference.

Again a drawback is the need to update the method signatures.

Recommendation

Our recommendation is that Option 1 is chosen as having the smallest impact on the API.

Resultant Changes

The following changes are required: -

1. A new data type – TpServiceInstanceID
2. A new parameter of this type added to createServiceManager()
3. A new TpDomainIDType value

Note that the definition of TpServiceID is currently misleading and it is suggested that the update shown below is made.

15.1.15 TpServiceID

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies a registered ~~an instance of a~~ SCF interface. The string is automatically generated by the Framework

15.1.33 TpServiceInstanceID

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies an instance of a registered SCF interface. The string is automatically generated by the Framework

15.1.4 TpDomainIDType

Defines either the framework or the type of entity attempting to access the framework

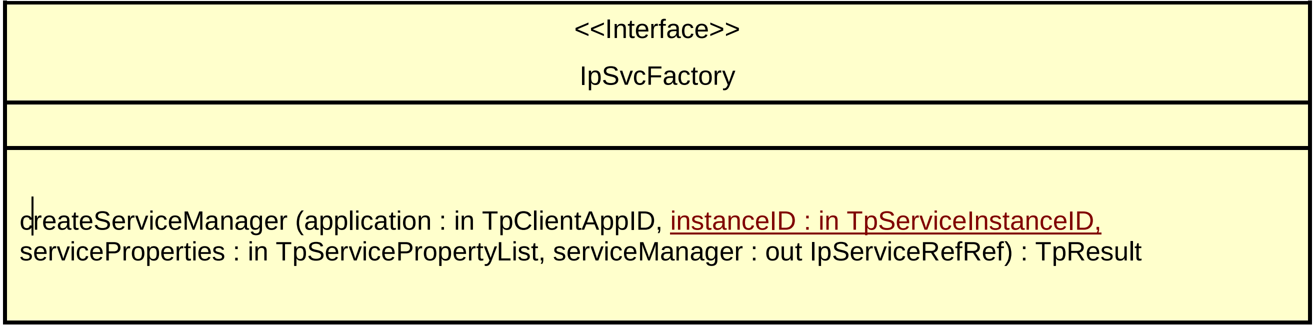
Name	Value	Description
P_FW	0	The framework
P_CLIENT_APPLICATION	1	A client application
P_ENT_OP	2	An enterprise operator
P_SERVICE_INSTANCE~~REGISTERED_SERVICE~~	3	A registered service
P_SERVICE_SUPPLIER	4	A service supplier

12.2 Service Factory Interface Classes

The IpSvcFactory interface allows the framework to get access to a service manager interface of a service. It is used during the signServiceAgreement, in order to return a service manager interface reference to the application. Each service has a service manager interface that is the initial point of contact for the service. E.g., the generic call control service uses the IpCallControlManager interface.

12.2.1 Interface Class IpSvcFactory

Inherits from: IpInterface.



<<Interface>>

IpSvcFactory

createServiceManager (application : in TpClientAppID, instanceID : in TpServiceInstanceID, serviceProperties : in TpServicePropertyList, serviceManager : out IpServiceRefRef) : TpResult

Method

`createServiceManager()`

This method returns a new service manager interface reference for the specified application. The service instance will be configured for the client application using the properties agreed in the service level agreement.

Parameters

`application : in TpClientAppID`

Specifies the application for which the service manager interface is requested.

`instanceID : in TpServiceInstanceID`

Specifies the Service Instance ID to be associated with the instance of the service. The Service Factory should pass this ID to the instance it creates.

`serviceProperties : in TpServicePropertyList`

Specifies the service properties and their values that are to be used to configure the service instance. These properties form a part of the service level agreement. An example of these properties is a list of methods that the client application is allowed to invoke on the service interfaces.

`serviceManager : out IpServiceRefRef`

Specifies the service manager interface reference for the specified application ID.

Raises

`TpCommonExceptions, P_INVALID_PROPERTY`

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010699**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 014** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

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: ⌘	P_UNAUTHORISED_PARAMETER_VALUE thrown if non-accessible serviceID is provided		
Source: ⌘	CN5		
Work item code: ⌘	OSA1	***Date:*** ⌘	30/08/2001
Category: ⌘	**F**	***Release:*** ⌘	REL-4
	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: ⌘	A number of methods on the IpLoadManager and IpFaultManager interfaces take either a single serviceID or a list of serviceIDs as a parameter. An APL should only be allowed to invoked these methods if it has access to the specified service(s) (signed a service agreement), as it is insecure to allow an APL to query the fault or load statistics for someone else's instance of a service.
Summary of change: ⌘	It is proposed that any method on the IpLoadManager and IpFaultManager interfaces that accepts either a single serviceID or a list of serviceIDs as a parameter should check that the APL has signed a service agreement for the specified service(s). If not, then the P_UNAUTHORISED_PARAMETER_VALUE exception should be thrown, with the string field of the exception indicating the serviceID at fault.
Consequences if not approved: ⌘	Applications would be able to query information on other people's instances of a service. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected: ⌘	8.3.2, 8.3.8	
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.

Problem

A number of methods on the IpLoadManager and IpFaultManager interfaces take either a single serviceID or a list of serviceIDs as a parameter. An APL should only be allowed to invoked these methods if it has access to the specified service(s) (signed a service agreement), as it doesn't seem to make sense to allow an APL to query the fault or load statistics for someone else's instance of a service.

Proposal

When a service agreement has been signed, that is when the FW operator agrees to provide a service to the application. This agreement most likely includes some kind of quality of service parameters (e.g. the maximum number of faults allowed before the agreement is to be terminated). It is only then that an APL should have access to this data.

Lucent proposes that any method on the IpLoadManager and IpFaultManager interfaces that accepts either a single serviceID or a list of serviceIDs as a parameter should check that the APL has signed a service agreement for the specified service(s). If not, then the P_~~SERVICE_ACCESS_DENIED~~UNAUTHORISED_PARAMETER_VALUE exception should be thrown, with the string field of the exception indicating the serviceID at fault.

Resulting changes

8.3.2 Interface Class IpFaultManager

Method

activityTestReq()

The application invokes this method to test that the framework or a service is operational. On receipt of this request, the framework must carry out a test on itself or on the specified service, to check that it is operating correctly. The framework reports the test result by invoking the activityTestRes method on the IpAppFaultManager interface. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

activityTestID : in TpActivityTestID

The identifier provided by the client application to correlate the response (when it arrives) with this request.

svcID : in TpServiceID

Identifies either the framework or a service for testing. The framework is designated by a null value.

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

svcUnavailableInd()

This method is used by the client application to inform the framework that it can no longer use the indicated service (either due to a failure in the client application or in the service). On receipt of this request, the framework should take the appropriate corrective action. The framework assumes that the session between this client application and service instance is to be closed and updates its own records appropriately as well as attempting to inform the service instance and/or its administrator. Attempts by the client application to continue using this session should be rejected. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

serviceID : in TpServiceID

Identifies the service that the application can no longer use.

Raises

TpCommonExceptions ,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

genFaultStatsRecordReq()

This method is used by the application to solicit fault statistics from the framework. On receipt of this request the framework must produce a fault statistics record, for the framework and/or for specified services during the specified time interval, which is returned to the client application using the genFaultStatsRecordRes operation on the IpAppFaultManager interface. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. A null value leaves this to the discretion of the framework.

serviceIDs : in TpServiceIDList

Specifies the framework and/or services to be included in the general fault statistics record. The framework is designated by a null value.

Raises

TpCommonExceptions–, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

8.3.8 Interface Class IpLoadManager

Method

queryLoadReq()

The client application uses this method to request the framework to provide load statistic records for the framework or for individual services used by the application.———If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which load statistics records should be reported. If this parameter is not an empty list, load statistics records of the specified services are returned, otherwise the load statistics record of the framework is returned.

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistics records should be reported.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

Method

registerLoadController()

The client application uses this method to register to receive notifications of load level changes associated with the framework and/or with individual services used by the application. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and SCFs to be registered for load control. To register for framework load control only, the serviceIDs is null.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

unregisterLoadController()

The client application uses this method to unregister for notifications of load level changes associated with the framework and/or with individual services used by the application. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which load level changes should no longer be reported. The framework is designated by a null value.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

resumeNotification()

The client application uses this method to request the framework to resume sending it load management notifications associated with the framework and/or with individual services used by the application; e.g. after a period of suspension during which the application handled a temporary overload condition. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which the sending of notifications of load level changes by the framework should be resumed. The framework is designated by a null value.

Raises

TpCommonExceptions,_P_INVALID_SERVICE_ID,_P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

Method

suspendNotification()

The client application uses this method to request the framework to suspend sending it load management notifications associated with the framework and/or with individual services used by the application; e.g. while the application handles a temporary overload condition. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework and/or the services for which the sending of notifications by the framework should be suspended. The framework is designated by a null value

Raises

TpCommonExceptions,_P_INVALID_SERVICE_ID,_P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010703**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**015**	⌘	ev	**-**	⌘	Current version:	**4.1.0** ⌘

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:	⌘	Introduction of Service Instance Lifecycle Management		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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:	⌘	There is no method that the Framework can invoke on the Service instance to indicate that the Session has been terminated unless the Service implements the IpSvcFaultManager interface. Even if that interface is supported the Framework shouldn't be using those methods if, for example, the Service Agreement should be terminated because the contract governing its terms expires. There needs to be a method that the framework can invoke to terminate the service instance.
Summary of change:	⌘	It is proposed that a new interface, IpServiceInstanceLifecycleManager, is introduced. This interface will replace the IpSvcFactory interface. The existing method on IpSvcFactory (createServiceManager) is carried over to the new interface. In addition a new method – destroyServiceManager – is supported.
Consequences if not approved:	⌘	There will be no way to terminate a service instance which does not implement the fault management interfaces. This means that the service instance cannot be terminated and will therefore hold resources indefinitely. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	6.4, 10.1, 10.2, 12.2, 13.2	
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.

Current Situation

A service instance is brought into existence when the Application invokes signServiceAgreement on the Framework. The Framework uses a Service Factory to create the instance, and a reference to it is returned to the Application.

It could be considered then that a Session between the Application and Service instance has been started.

This Session is considered to be ended under the following conditions :-

The Application invokes IpAccess.terminateServiceAgreement()
The Application invokes IpFaultManager.svcUnavailableInd()
The Service instance invokes IpFwFaultManager.appUnavailableInd()
The Service instance invokes IpFwFaultManager.svcRemovalInd()
The Framework invokes IpSvcFaultManager.svcUnavailableInd() – as a result of item 2 above
The Framework invokes IpSvcFaultManager.appRemovalInd() – possibly as a result of item 2 above

(If the service instance invokes either of the above IpFwFaultManager methods the Framework needs to pass that information on to the application, which can then decide whether to end the service agreement or not. Currently, the IpAppFaultManager interface is lacking an appRemovalInd method so this cannot be done. Likewise, the IpFaultManager is lacking a svcUnavailableInd which the application could invoke to inform the framework that it is having trouble with the service.)

Issue

There is currently no method that the Framework can invoke on the Service instance to indicate that the Session has been terminated unless the Service implements the IpSvcFaultManager interface. Even if that interface is supported the Framework shouldn't be using those methods if, for example, the Service Agreement should be terminated because the contract governing its terms expires. There needs to be a method that the framework can invoke to terminate the service instance.

Proposal

The original proposal in N5-010577 is superceded as a result of discussions held at the meeting in Sophia Antipolis. The new proposal follows.

It is proposed that a new interface, IpServiceInstanceLifecycleManager, is introduced. This interface will replace the IpSvcFactory interface.

The existing method on IpSvcFactory (createServiceManager) is carried over to the new interface. In addition a new method - destroyServiceManager – is supported.

When an Application wants to end the session it has with a Service it invokes terminateServiceAgreement() on the Framework's IpAccess interface and the Framework will invoke destroyServiceManager() on the Service's Service Instance Lifecycle manager.

Resultant Changes

❑ New IpServiceInstanceLifecycleManager interface
❑ IpSvcFactory interface removed
❑ The announceServiceAvailability method must be updated

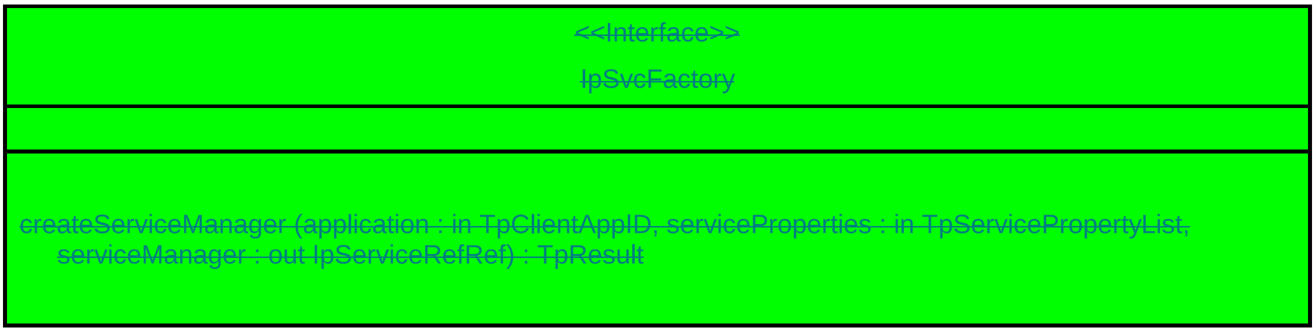
In addition numerous sequence diagrams, class diagrams and text which include the IpSvcFactory interface or refer to the service factory need to be updated.

Note: The service factory is referenced in the sequence diagram in section 10.2.1. This has been changed in this contribution to reference the service instance lifecycle manager, but the change will not show up properly.

Note that N5-010697 introduces the service instance id to the createServiceManager method of IpSvcFactory so this should be carried over into the new IpServiceInstanceLifecycleManager.

~~12.2.1 Interface Class IpSvcFactory~~

~~Inherits from: IpInterface.~~



~~<<Interface>>~~

~~IpSvcFactory~~

~~createServiceManager (application : in TpClientAppID, serviceProperties : in TpServicePropertyList, serviceManager : out IpServiceRefRef) : TpResult~~

~~Method~~

~~createServiceManager()~~

~~This method returns a new service manager interface reference for the specified application. The service instance will be configured for the client application using the properties agreed in the service level agreement.~~

~~Parameters~~

~~application : in TpClientAppID~~

~~Specifies the application for which the service manager interface is requested.~~

~~serviceProperties : in TpServicePropertyList~~

~~Specifies the service properties and their values that are to be used to configure the service instance. These properties form a part of the service level agreement. An example of these properties is a list of methods that the client application is allowed to invoke on the service interfaces.~~

~~serviceManager : out IpServiceRefRef~~

~~Specifies the service manager interface reference for the specified application ID.~~

~~Raises~~

~~TpCommonExceptions,P_INVALID_PROPERTY~~

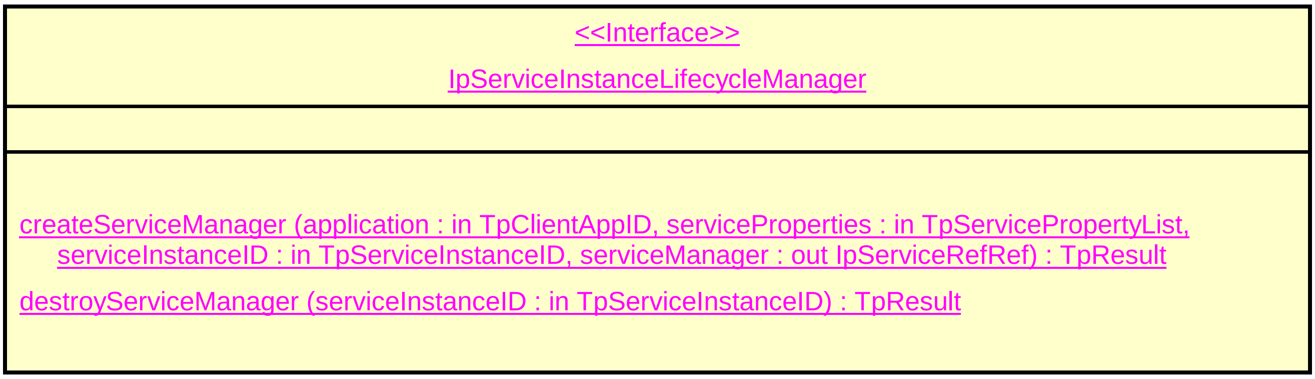
12.2 Service ~~Factory~~ Instance Lifecycle Manager Interface Classes

The ~~IpSvcFactory~~ IpServiceInstanceLifecycleManager interface allows the framework to get access to a service manager interface of a service. It is used during the signServiceAgreement, in order to return a service manager interface reference to the application. Each service has a service manager interface that is the initial point of contact for the service. E.g., the generic call control service uses the IpCallControlManager interface.

12.2.1 Interface Class IpServiceInstanceLifecycleManager

Inherits from: IpInterface.

The IpServiceInstanceLifecycleManager interface allows the Framework to create and destroy Service Manager Instances.



<<Interface>>

IpServiceInstanceLifecycleManager

createServiceManager (application : in TpClientAppID, serviceProperties : in TpServicePropertyList, serviceInstanceID : in TpServiceInstanceID, serviceManager : out IpServiceRefRef) : TpResult

destroyServiceManager (serviceInstanceID : in TpServiceInstanceID) : TpResult

Method

createServiceManager()

This method returns a new service manager interface reference for the specified application. The service instance will be configured for the client application using the properties agreed in the service level agreement.

Parameters

application : in TpClientAppID

Specifies the application for which the service manager interface is requested.

serviceProperties : in TpServicePropertyList

Specifies the service properties and their values that are to be used to configure the service instance. These properties form a part of the service level agreement. An example of these properties is a list of methods that the client application is allowed to invoke on the service interfaces.

serviceInstanceID : in TpServiceInstanceID

Specifies the Service Instance ID that the new Service Manager is to be identified by.

serviceManager : out IpServiceRefRef

Specifies the service manager interface reference for the specified application ID.

Raises

TpCommonExceptions,P_INVALID_PROPERTY

Method

destroyServiceManager()

This method destroys an existing service manager interface reference. This will result in the client application being unable to use the service manager any more.

Parameters

serviceInstance : in TpServiceInstanceID

Identifies the Service Instance to be destroyed.

Raises

TpCommonExceptions

6.4 Trust and Security Management Sequence Diagrams

6.4.1 Service Selection

The following figure shows the process of selecting an SCF.
After discovery the Application gets a list of one or more SCF versions that match its required description. It now needs to decide which service it is going to use; it also needs to actually get a way to use it.
This is achieved by the following two steps:

1: Service Selection: first step - selectService
In this first step the Application identifies the SCF version it has finally decided to use. This is done by means of the serviceID, which is the agreed identifier for SCF versions. The Framework acknowledges this selection by returning to the Application a new identifier for the service chosen: a service token, that is a private identifier for this service between this Application and this network, and is used for the process of signing the service agreement.
Input is:
· in serviceID
This identifies the SCF required.
And output:
· out serviceToken
This is a free format text token returned by the framework, which can be signed as part of a service agreement. It contains operator specific information relating to the service level agreement.
2: Service Selection: second step - signServiceAgreement
In this second step an agreement is signed that allows the Application to use the chosen SCF version. And once this contractual details have been agreed, then the Application can be given the means to actually use it. The means are a reference to the manager interface of the SCF version (remember that a manager is an entry point to any SCF). By calling the createServiceManager operation on the lifecycle manager~~service factory~~ the Framework retrieves this interface and returns it to the Application. The service properties suitable for this application are also fed to the SCF (via the ~~service factory~~lifecycle manager interface) in order for the SCS to instantiate an SCF version that is suitable for this application.
Input:
· in serviceToken
This is the identifier that the network and Application have agreed to privately use for a certain version of SCF.
· in agreementText
This is the agreement text that is to be signed by the Framework using the private key of the Framework.
· in signingAlgorithm
This is the algorithm used to compute the digital signature.
Output:
· out signatureAndServiceMgr
This is a reference to a structure containing the digital signature of the Framework for the service agreement, and a reference to the manager interface of the SCF.

10.1 Service Registration Sequence Diagrams

10.1.1 New SCF Registration

The following figure shows the process of registering a new Service Capability Feature in the Framework. Service Registration is a two step process:

1: Registration: first step - register service
The purpose of this first step in the process of registration is to agree, within the network, on a name to call, internally, a newly installed SCF version. It is necessary because the OSA Framework and SCF in the same network may come from different vendors. The goal is to make an association between the new SCF version, as characterized by a list of properties, and an identifier called serviceID.
This service ID will be the name used in that network (that is, between that network's Framework and its SCSs), whenever it is necessary to refer to this newly installed version of SCF (for example for announcing its availability, or for withdrawing it later).
The following input parameters are given from the SCS to the Framework in this first registration step:
· in serviceTypeName
This is a string with the name of the SCF, among a list of standard names (e.g. "P_MPCC").
· in servicePropertyList

This is a list of types TpServiceProperty; each TpServiceProperty is a triplet (ServicePropertyName, ServicePropertyValueList, ServicePropertyMode).

- ServicePropertyName is a string that defines a valid SFC property name (valid SCF property names are listed in the SCF data definition).
- ServicePropertyValueList is a numbered set of types TpServicePropertyValue; TpServicePropertyValue is a string that describes a valid value of a SCF property (valid SCF property values are listed in the SCF data definition).
- ServicePropertyMode is the value of the property modes (e.g. "mandatory", meaning that all properties of this SCF must be given values at service registration time).

The following output parameter results from service registration:

- out serviceID

This is a string, automatically generated by the Framework of this network, based on the following:

- a string that contains a unique number, generated by the Framework;
- a string that identifies the SCF name (e.g. "P_MPCC");
- a concatenation of strings that identify the SCF specialization, if any.

This is the name by which the newly installed version of SCF, described by the list of properties above, is going to be identified internally in this network.

2: Registration: second step - announce service availability

At this point the network's Framework is aware of the existence of a new SCF, and could let applications know - but they would have no way to use it. Installing the SCS logic and assigning a name to it does not make this SCF available. In order to make the SCF available an "entry point", called ~~service factory~~lifecycle manager, is used. The role of the lifecycle manager~~service factory~~ is to control the life cycle of an interface, or set of interfaces, and provide clients with the references that are necessary to invoke the methods offered by these interfaces. The starting point for a client to use an SCF is to obtain an interface reference to a lifecycle manager~~service factory~~ of the desired SCF.

A Network Operator, upon completion of the first registration phase, and once it has an identifier to the new SCF version, will instantiate a lifecycle manager~~service factory~~ for it that will allow client to use it. Then it will inform the Framework of the value of the interface associated to the new SCF. After the receipt of this information, the Framework makes the new SCF (identified by the pair [serviceID, serviceInstanceLifecycleManager~~serviceFactory~~Ref]) discoverable.

The following input parameters are given from the SCS to the Framework in this second registration step:

- in serviceID

This is the identifier that has been agreed in the network for the new SCF; any interaction related to the SCF needs to include the serviceID, to know which SCF it is.

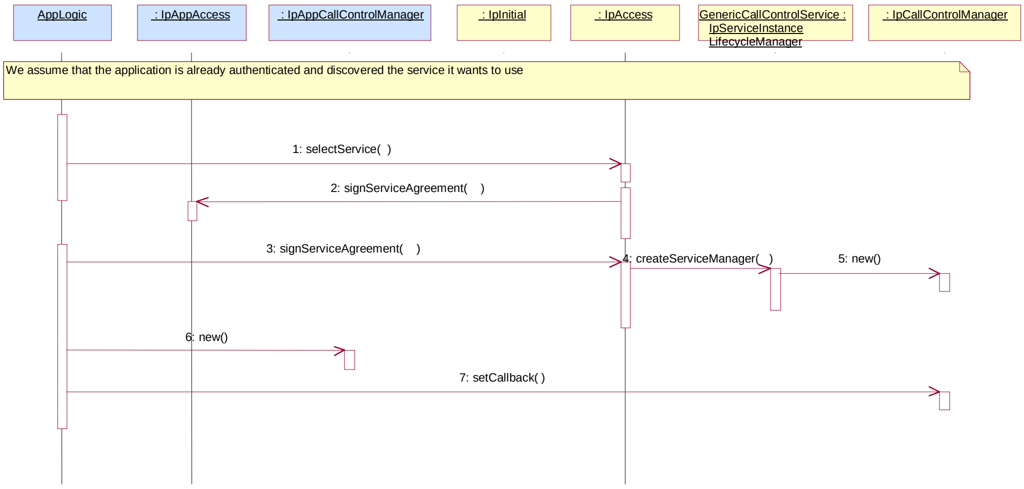
- in ~~serviceFactoryRef~~serviceInstanceLifecycleManagerRef

This is the interface reference at which the ~~service factory~~lifecycle manager of the new SCF is available. Note that the Framework will have to invoke the method createServiceManager() in this interface, any time between now and when it accepts the first application requests for discovery, so that it can get the service manager interface necessary for applications as an entry point to any SCF.

10.2 Service ~~Factory~~ Instance Lifecycle Manager Sequence Diagrams

10.2.1 Sign Service Agreement

This sequence illustrates how the application can get access to a specified service. It only illustrates the last part: the signing of the service agreement and the corresponding actions towards the service. For more information on accessing the framework, authentication and discovery of services, see the corresponding sections.


AppLogic
: IpAppAccess
: IpAppCallControlManager
: IpInitial
: IpAccess
GenericCallControlService : IpServiceInstance LifecycleManager
: IpCallControlManager
We assume that the application is already authenticated and discovered the service it wants to use
1: selectService( )
2: signServiceAgreement( )
3: signServiceAgreement( )
4: createServiceManager( )
5: new()
6: new()
7: setCallback( )

1: The application selects the service, using a serviceID for the generic call control service. The serviceID could have been obtained via the discovery interface. A ServiceToken is returned to the application.

2: The framework signs the service agreement.

3: The client application signs the service agreement. As a result a service manager interface reference (in this case of type IpCallControlManager) is returned to the application.

4: Provided the signature information is correct and all conditions have been fulfilled, the framework will request the service identified by the serviceID to return a service manager interface reference. The service manager is the initial point of contact to the service.

5: The ~~service factory~~lifecycle manager creates a new manager interface instance (a call control manager) for the specified application. It should be noted that this is an implementation detail. The service implementation may use other mechanism to get a service manager interface instance.

6: The application creates a new IpAppCallControlManager interface to be used for callbacks.

7: The Application sets the callback interface to the interface created with the previous message.

13.2 Service ~~Factory~~ Instance Lifecycle Manager State Transition Diagrams

There are no State Transition Diagrams defined for the Service ~~Factory~~Instance Lifecycle Manager.

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010708**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**016**	⌘	ev	**-**	⌘	Current version:	**4.1.0**	⌘

*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 multi-vendorship		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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:	⌘	By having specified only the interfaces for registration and service factory, the 3GPP specification does not allow multi-vendorship as it is not possible for an SCS based on the TS 29.198 from one vendor to work with the FW implementation of another vendor. At this moment there is e.g. no support for obtaining access to the OSA framework, the integrity management, etc between an SCS and the FW in the TS 29.198. Therefore there is no way to fulfill multi-vendorship. In addition, in order to support e.g. registration of new Service Capabilities that are within the same domain as the Framework it is required to unambigously specify on how trusted SCSs obtain access to the OSA Framework. At this moment, however, the specification is not clear on how trusted entities would gain access to the OSA Framework. Therefore, there is currently also no support for multi-vendorship within a single domain.
Summary of change:	⌘	The multi-vendorship requirement can be fulfilled when the complete set of interfaces between Framework and Services, as present in the scope of ETSI SPAN 12 and Parlay 3.0, is adopted in the TS 29.198. Therefore in this CR the complete set of interfaces between SCS and FW is introduced. This allows support for multi-vendorship (in multiple domains). For the multi-vendorship within a single domain a new sequence in the Trust and Security Management is introduced (section 10.1.1) to show how trusted parties obtain access to the OSA Framework.
Consequences if not approved:	⌘	No support for multi-vendorship.

Clauses affected:	⌘	10		
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.

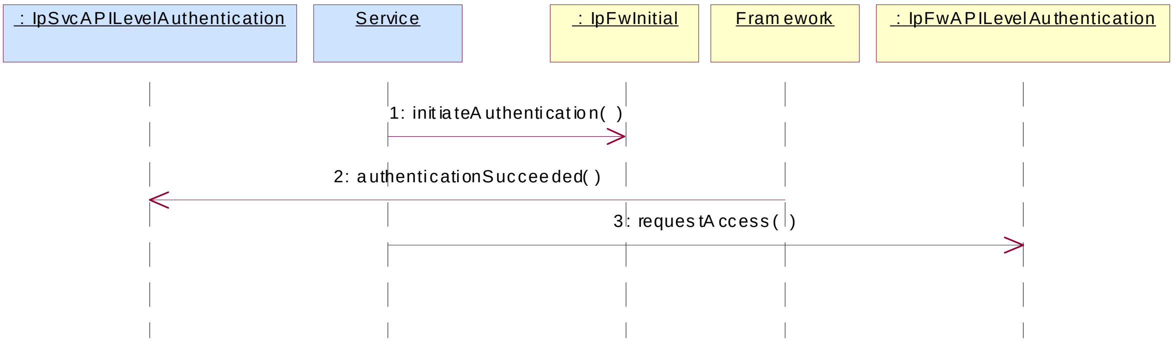
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

10 Framework-to-Service Sequence Diagrams

10.1 Trust and Security Management Sequence Diagrams

10.1.1 Initial Access for trusted parties

The following figure shows a trusted party (SCS), typically within the same domain as the Framework, accessing the OSA Framework for the first time. Trusted parties don't need to be authenticated and after contacting the Initial interface the Framework will indicate that no further authentication is needed and that the SCS can immediately gain access to other framework interfaces. This is done by invoking the requestAccess method.



: IpSvcAPILevelAuthentication
Service
: IpFwInitial
Framework
: IpFwAPILevelAuthentication
1: initiateAuthentication( )
2: authenticationSucceeded( )
3: requestAccess( )

1: Initiate Authentication

The Application invokes initiateAuthentication on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

2: Authentication Succeeded

Based on the domainID information that was supplied in the Iniate Authentication step, the Framework knows it deals with a trusted party and no further authentication is needed. Therefore the Framework provides the authentication succeeded indication.

3: Request Access

The Application invokes requestAccess on the Framework's API Level Authenticaiton interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

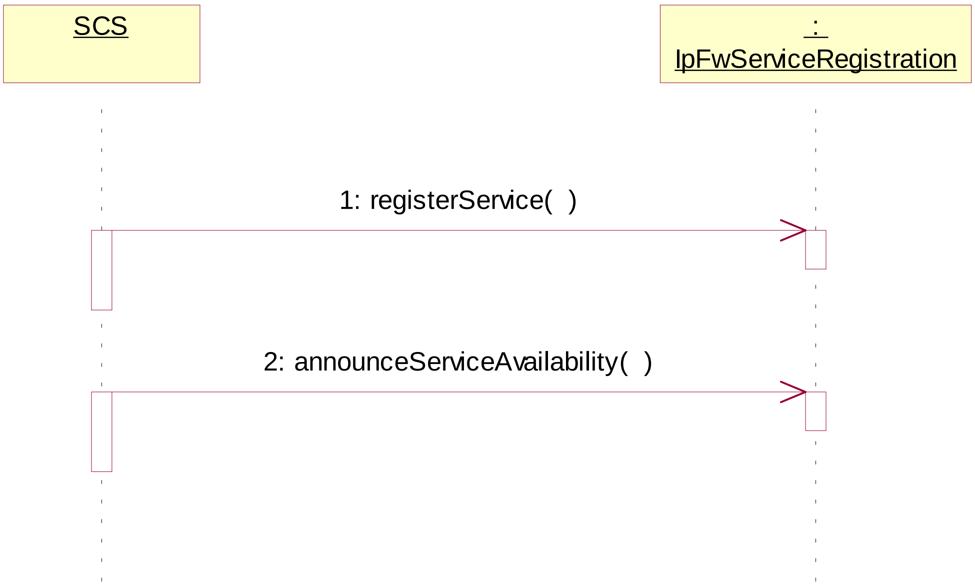
10.2 Service Discovery Sequence Diagrams

No Sequence Diagrams exist for Service Discovery

~~10.1~~10.3 Service Registration Sequence Diagrams

~~10.1.1~~10.3.1 New SCF Registration

The following figure shows the process of registering a new Service Capability Feature in the Framework. Service Registration is a two step process:



1: Registration: first step - register service

The purpose of this first step in the process of registration is to agree, within the network, on a name to call, internally, a newly installed SCF version. It is necessary because the OSA Framework and SCF in the same network may come from different vendors. The goal is to make an association between the new SCF version, as characterized by a list of properties, and an identifier called serviceID.

This service ID will be the name used in that network (that is, between that network's Framework and its SCSs), whenever it is necessary to refer to this newly installed version of SCF (for example for announcing its availability, or for withdrawing it later).

The following input parameters are given from the SCS to the Framework in this first registration step:

- in serviceTypeName

This is a string with the name of the SCF, among a list of standard names (e.g. "P_MPCC").

- in servicePropertyList

This is a list of types TpServiceProperty; each TpServiceProperty is a triplet (ServicePropertyName, ServicePropertyValueList, ServicePropertyMode).

- ServicePropertyName is a string that defines a valid SFC property name (valid SCF property names are listed in the SCF data definition).

- ServicePropertyValueList is a numbered set of types TpServicePropertyValue; TpServicePropertyValue is a string that describes a valid value of a SCF property (valid SCF property values are listed in the SCF data definition).

- ServicePropertyMode is the value of the property modes (e.g. "mandatory", meaning that all properties of this SCF must be given values at service registration time).

The following output parameter results from service registration:

- out serviceID

This is a string, automatically generated by the Framework of this network, based on the following:

- a string that contains a unique number, generated by the Framework;

- a string that identifies the SCF name (e.g. "P_MPCC");

- a concatenation of strings that identify the SCF specialization, if any.

This is the name by which the newly installed version of SCF, described by the list of properties above, is going to be identified internally in this network.

2: Registration: second step - announce service availability

At this point the network's Framework is aware of the existence of a new SCF, and could let applications know - but they would have no way to use it. Installing the SCS logic and assigning a name to it does not make this SCF available. In order to make the SCF available an "entry point", called service factory, is used. The role of the service factory is to control the life cycle of an interface, or set of interfaces, and provide clients with the references that are necessary to invoke the methods offered by these interfaces. The starting point for a client to use an SCF is to obtain an interface reference to a factory of the desired SCF.

A Network Operator, upon completion of the first registration phase, and once it has an identifier to the new SCF version, will instantiate a factory for it that will allow client to use it. Then it will inform the Framework of the value of the interface associated to the new SCF. After the receipt of this information, the Framework makes the new SCF (identified by the pair [serviceID, serviceFactoryRef]) discoverable.

The following input parameters are given from the SCS to the Framework in this second registration step:

- in serviceID

This is the identifier that has been agreed in the network for the new SCF; any interaction related to the SCF needs to include the serviceID, to know which SCF it is.

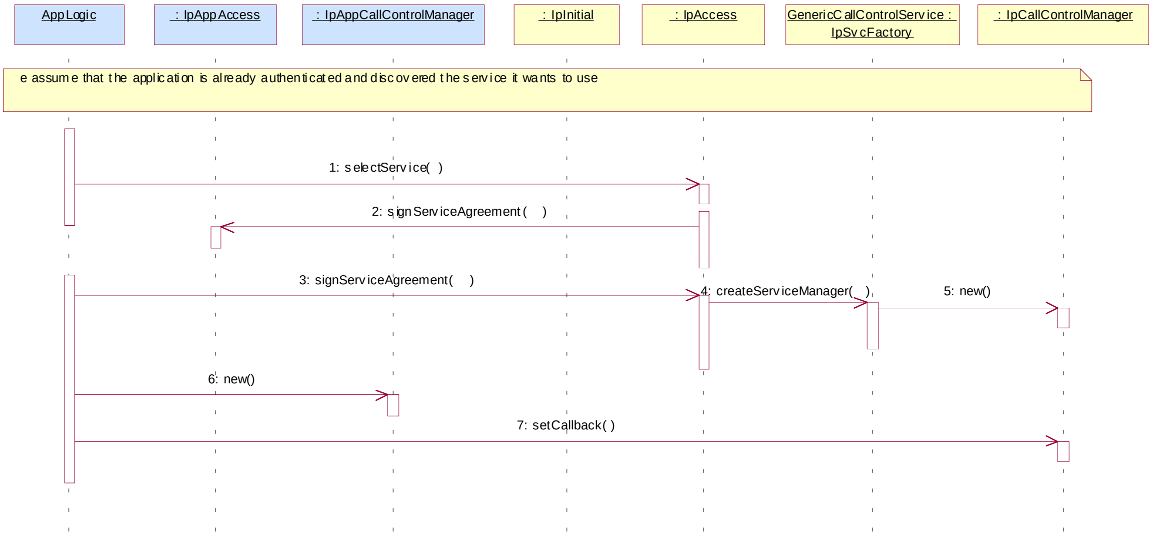
- in serviceFactoryRef

This is the interface reference at which the service factory of the new SCF is available. Note that the Framework will have to invoke the method createServiceManager() in this interface, any time between now and when it accepts the first application requests for discovery, so that it can get the service manager interface necessary for applications as an entry point to any SCF.

Service Factory Sequence Diagrams

~~10.2.1~~10.4.1 Sign Service Agreement

This sequence illustrates how the application can get access to a specified service. It only illustrates the last part: the signing of the service agreement and the corresponding actions towards the service. For more information on accessing the framework, authentication and discovery of services, see the corresponding sections.


AppLogic
: IpAppAccess
: IpAppCallControlManager
: IpInitial
: IpAccess
GenericCallControlService : IpSvcFactory
: IpCallControlManager
e assume that the application is already authenticated and discovered the service it wants to use
1: selectService( )
2: signServiceAgreement( )
3: signServiceAgreement( )
4: createServiceManager( )
5: new()
6: new()
7: setCallback( )

1: The application selects the service, using a serviceID for the generic call control service. The serviceID could have been obtained via the discovery interface. A ServiceToken is returned to the application.

2: The framework signs the service agreement.

3: The client application signs the service agreement. As a result a service manager interface reference (in this case of type IpCallControlManager) is returned to the application.

4: Provided the signature information is correct and all conditions have been fulfilled, the framework will request the service identified by the serviceID to return a service manager interface reference. The service manager is the initial point of contact to the service.

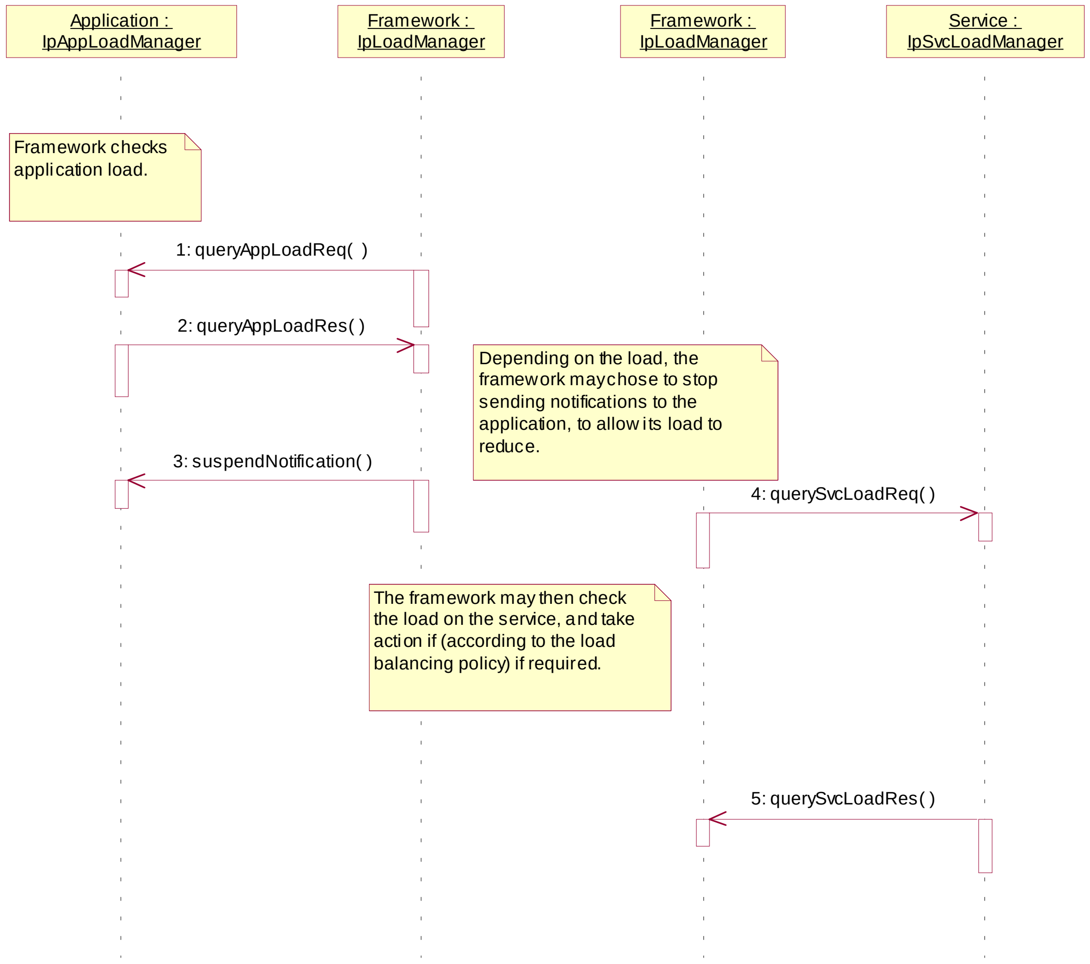
5: The service factory creates a new manager interface instance (a call control manager) for the specified application. It should be noted that this is an implementation detail. The service implementation may use other mechanism to get a service manager interface instance.

6: The application creates a new IpAppCallControlManager interface to be used for callbacks.

7: The Application sets the callback interface to the interface created with the previous message.

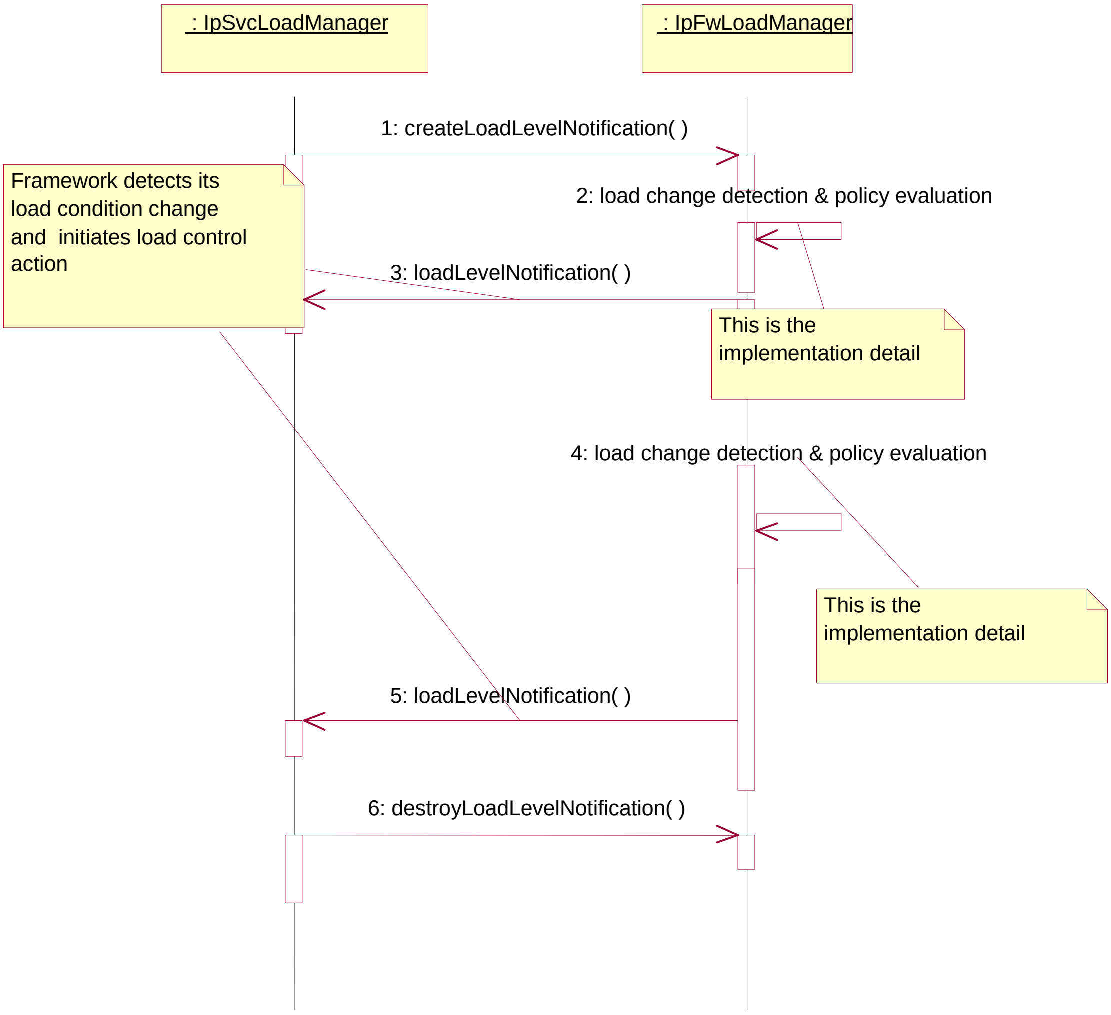
10.5 Integrity Management Sequence Diagrams

10.5.1 Load Management: Client and Service Load Balancing

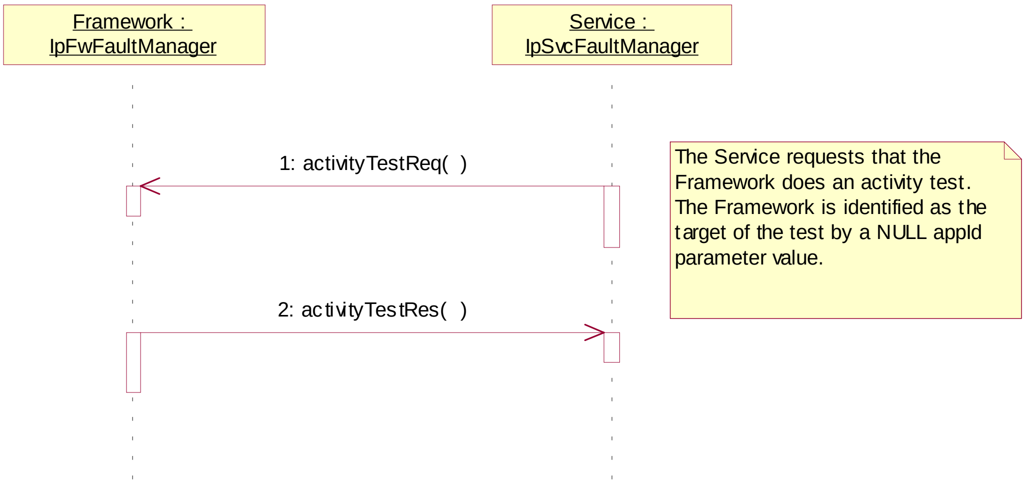

Application : IpAppLoadManager
Framework : IpLoadManager
Framework : IpLoadManager
Service : IpSvcLoadManager
Framework checks application load.
1: queryAppLoadReq( )
2: queryAppLoadRes( )
Depending on the load, the framework may chose to stop sending notifications to the application, to allow its load to reduce.
3: suspendNotification( )
4: querySvcLoadReq( )
The framework may then check the load on the service, and take action if (according to the load balancing policy) if required.
5: querySvcLoadRes( )

10.5.2 Load Management: Service callback registration and load control

This sequence diagram shows how a service creates a load level notification request for itself and the framework invokes load management function based on policy.


: IpSvcLoadManager
: IpFwLoadManager
1: createLoadLevelNotification( )
2: load change detection & policy evaluation
3: loadLevelNotification( )
Framework detects its load condition change and initiates load control action
This is the implementation detail
4: load change detection & policy evaluation
This is the implementation detail
5: loadLevelNotification( )
6: destroyLoadLevelNotification( )

~~10.5.2~~10.5.3 Fault Management: Service requests Framework activity test

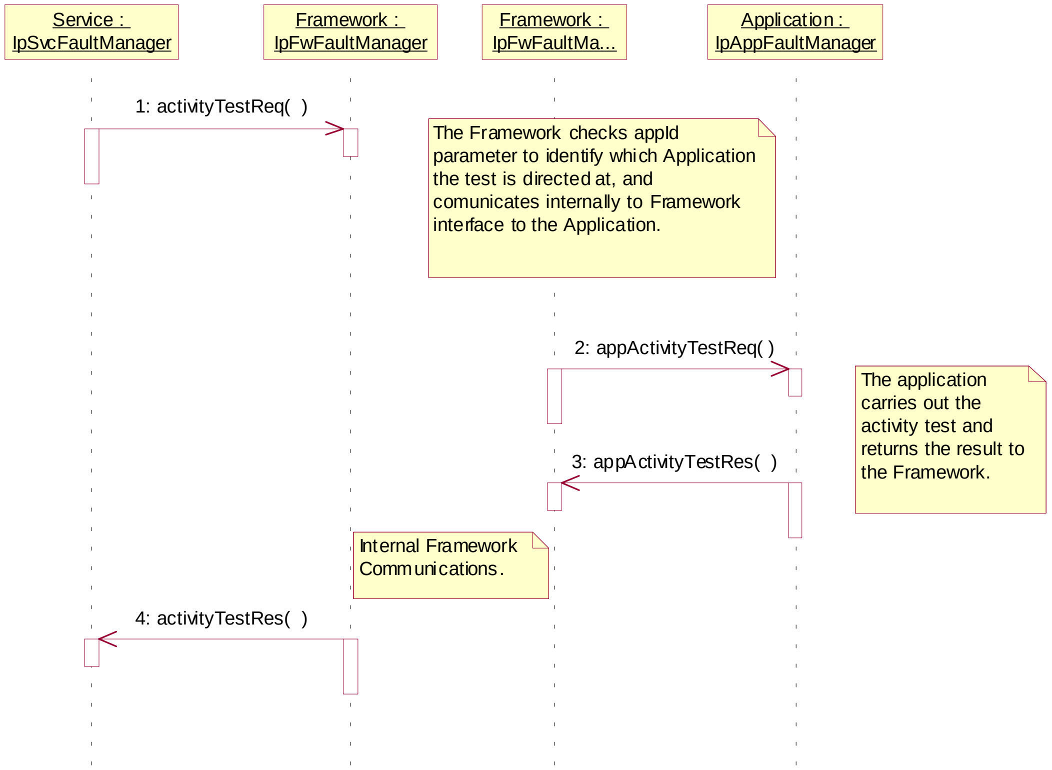


Framework : IpFwFaultManager
Service : IpSvcFaultManager
1: activityTestReq( )
2: activityTestRes( )
The Service requests that the Framework does an activity test. The Framework is identified as the target of the test by a NULL appId parameter value.

1: The service asks the framework to carry out its activity test. The service denotes that it requires the activity test done for the framework, rather than an application, by supplying a NULL value for the appID parameter.

2: The framework carries out the test and returns the result to the service.

~~10.5.3~~10.5.4 Fault Management: Service requests Application activity test


Service : IpSvcFaultManager
Framework : IpFwFaultManager
Framework : IpFwFaultMa...
Application : IpAppFaultManager
1: activityTestReq( )
The Framework checks appId parameter to identify which Application the test is directed at, and comunicates internally to Framework interface to the Application.
2: appActivityTestReq( )
The application carries out the activity test and returns the result to the Framework.
3: appActivityTestRes( )
Internal Framework Communications.
4: activityTestRes( )

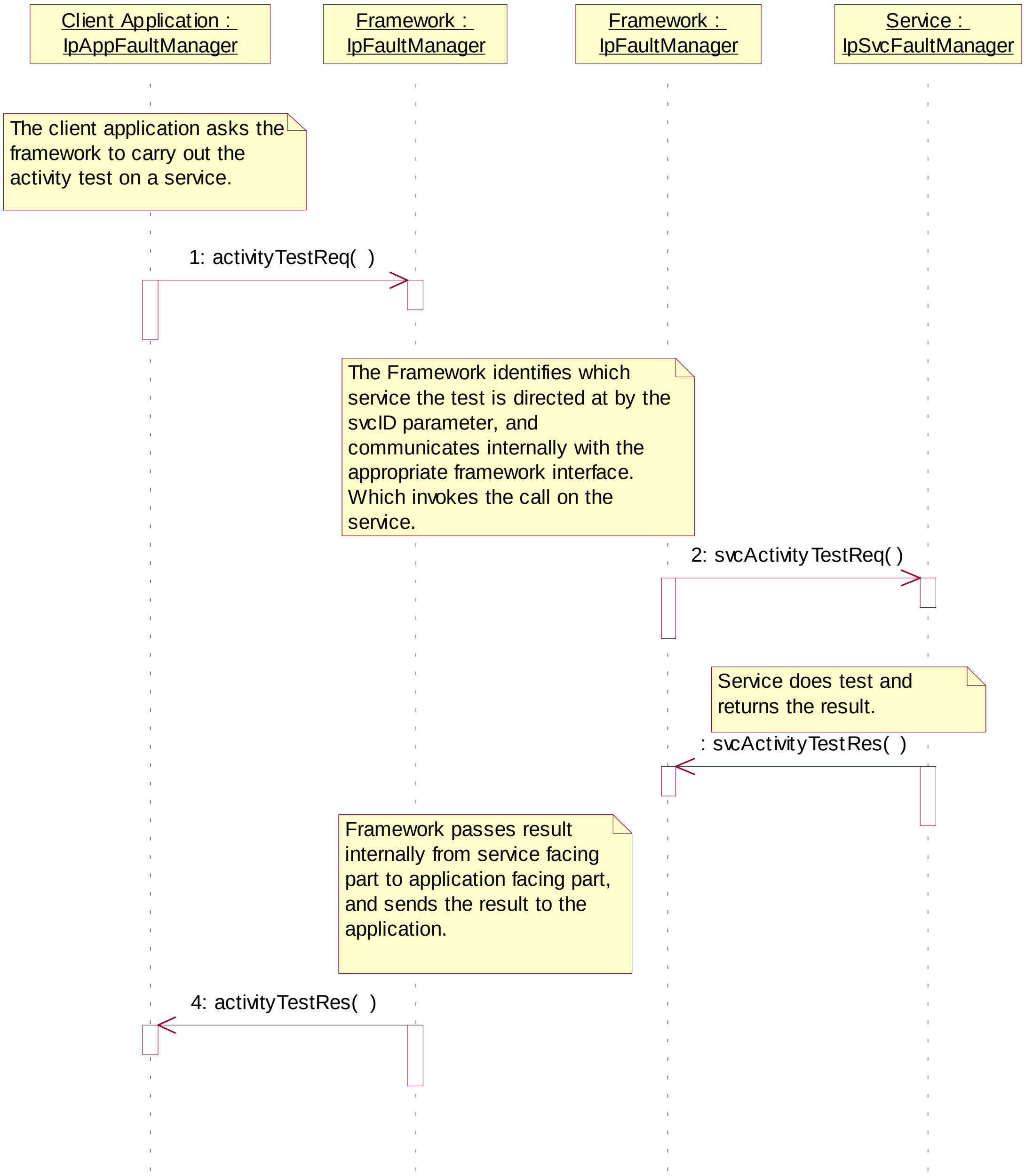
1: The service asks the framework to invoke an activity test on a client application, the application is identified by the appId parameter.

2: The framework asks the application to do the activity test. It is assumed that there is internal communication between the service facing part of the framework (i.e IpFwFaultManager interface) and the part that faces the client application.

3: The application does the activity test and returns the result to the framework.

4: The framework internally passes the result from its application facing interface (IpFaultManager) to its service facing side, and sends the result to the service.

~~10.5.4~~10.5.5 Fault Management: Application requests Service activity test


Client Application : IpAppFaultManager
Framework : IpFaultManager
Framework : IpFaultManager
Service : IpSvcFaultManager
The client application asks the framework to carry out the activity test on a service.
1: activityTestReq( )
The Framework identifies which service the test is directed at by the svcID parameter, and communicates internally with the appropriate framework interface. Which invokes the call on the service.
2: svcActivityTestReq( )
Service does test and returns the result.
: svcActivityTestRes( )
Framework passes result internally from service facing part to application facing part, and sends the result to the application.
4: activityTestRes( )

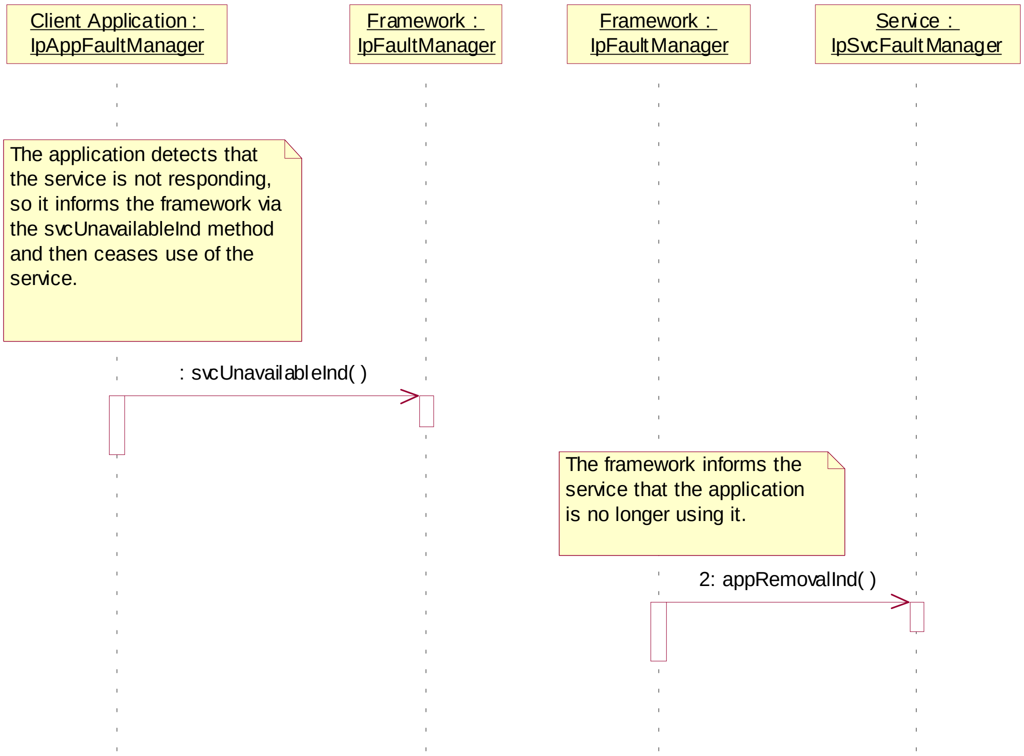
1: The client application asks the framework to invoke an activity test on a service, the service is identified by the svcId parameter.

2: The framework asks the service to do the activity test. It is assumed that there is internal communication between the application facing part of the framework (i.e IpFaultManager interface) and the part that faces the service.

3: The service does the activity test and returns the result to the framework.

4: The framework internally passes the result from its service facing interface (IpFwFaultManager) to its application facing side, and sends the result to the client application.

~~10.5.5~~10.5.6 Fault Management: Application detects service is unavailable



Client Application : IpAppFaultManager
Framework : IpFaultManager
Framework : IpFaultManager
Service : IpSvcFaultManager
The application detects that the service is not responding, so it informs the framework via the svcUnavailableInd method and then ceases use of the service.
: svcUnavailableInd( )
The framework informs the service that the application is no longer using it.
2: appRemovalInd( )

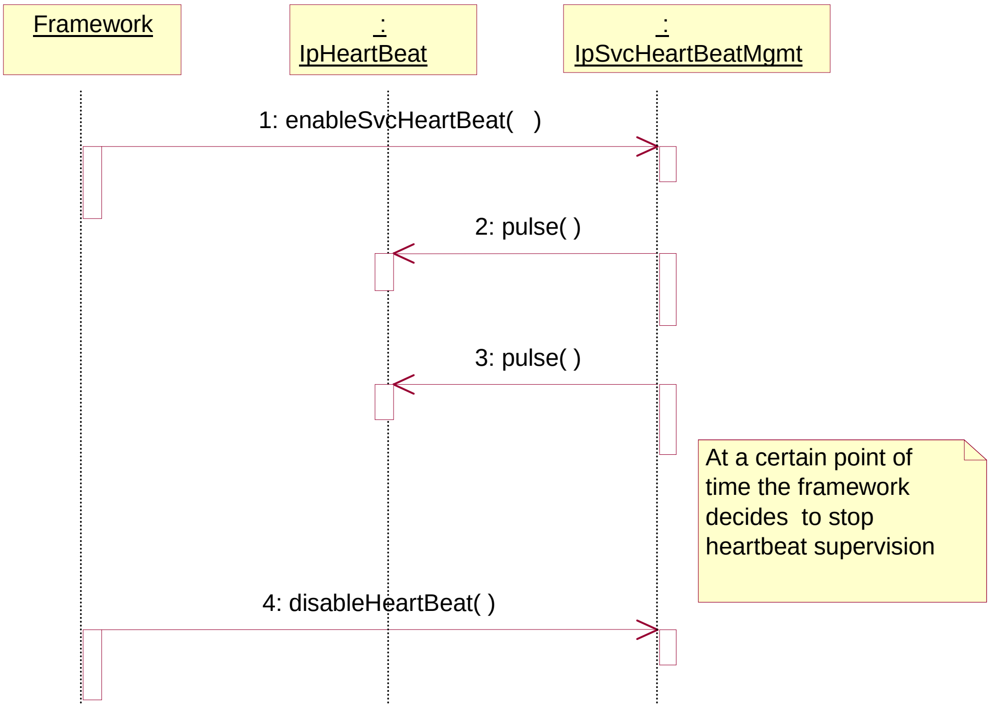
1: The client application detects that the service instance is currently not available, i.e. the service instance is not responding to the client application in the normal way, so it informs the framework and takes action to stop using this service instance and change to a different one (via the usual mechanisms, such as discovery, selectService etc.). The client application should not need to re-authenticate in order to discover and use an alternative service instance.

2: The framework informs the service instance that the client application was unable to get a response from it and has ceased to be one of its users. The framework and service instance must carry out the appropriate updates to remove the client application as one of the users of this service instance. The service or framework may then decide to carry out an activity test to see whether there is a general problem with the service instance that requires further action.

10.6 Event Notification Sequence Diagrams

No Sequence Diagrams exist for Event Notification

10.7 Heartbeat Management: Start/perform/end heartbeat supervision of the service



In this sequence diagram, the framework has decided that it wishes to monitor the service, and has therefore requested the service to commence sending its heartbeat. The service responds by sending its heartbeat at the specified interval. The framework then decides that it is satisfied with the service’s health and disables the heartbeat mechanism. If the heartbeat was not received from the service within the specified interval, the framework can decide that the service has failed the heartbeat and can then perform some recovery action.

Framework-to-Service Class Diagrams

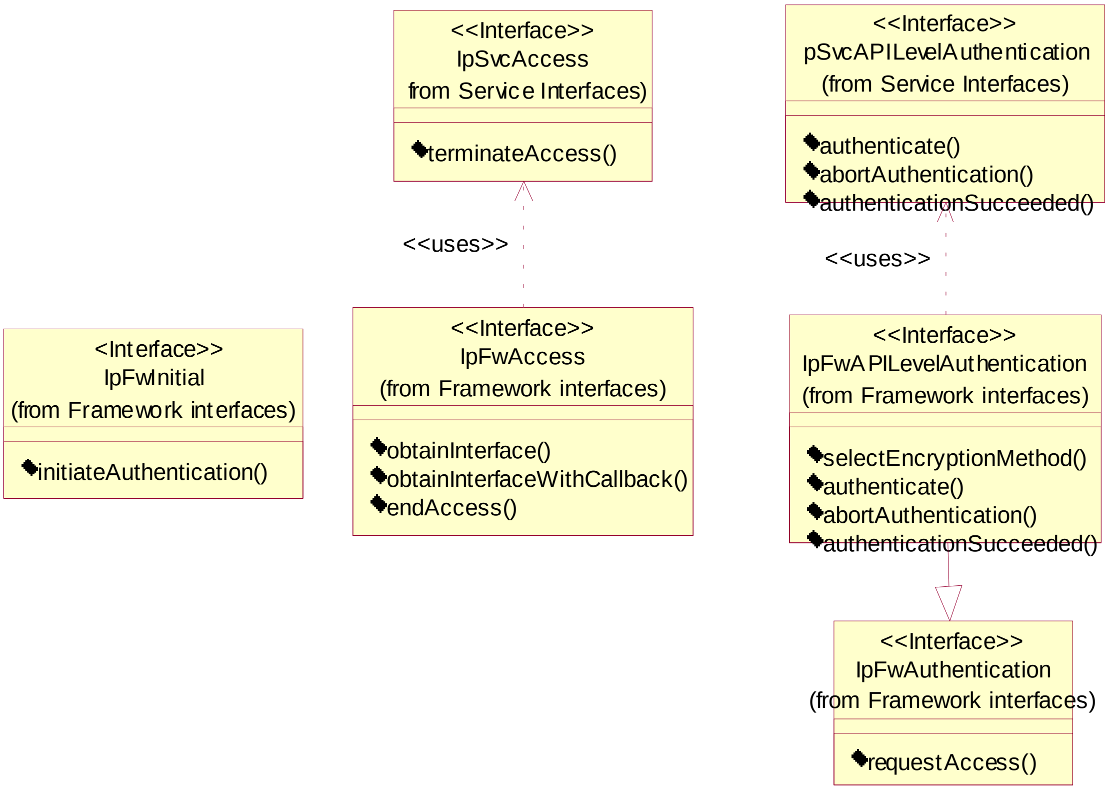


Figure: Trust and Security Management Package Overview

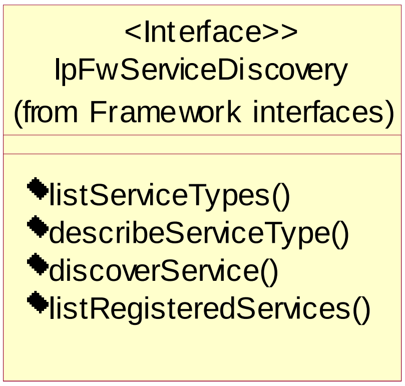


Figure: Service Discovery Package Overview

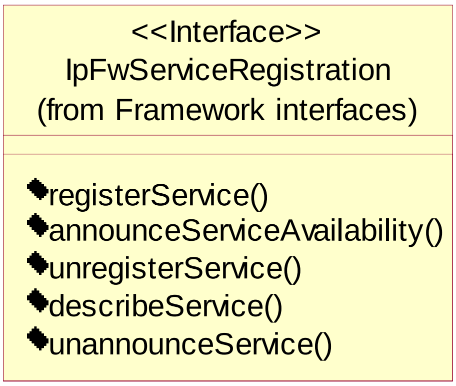


Figure: Service Registration Package Overview

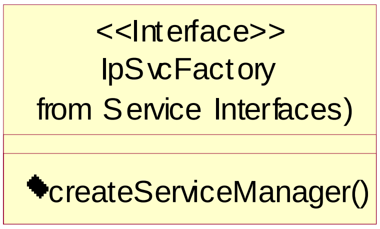


Figure: Service Factory Package Overview

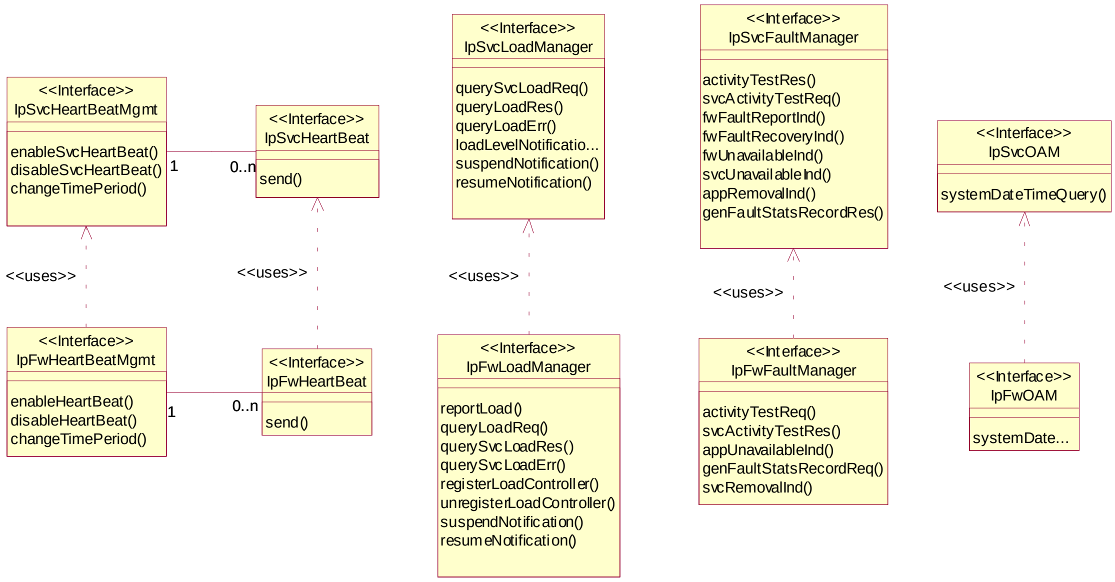


Figure: Integrity Management Package Overview

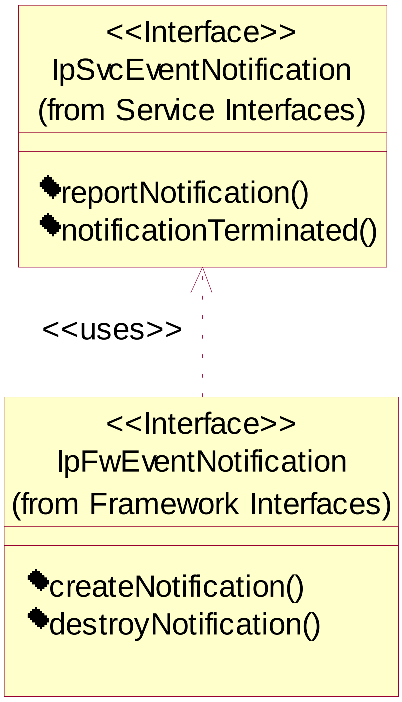


Figure: Event Notification Package Overview

12 Framework-to-Service Interface Classes

12.1 Trust and Security Management Interface Classes

12.1.1 Interface Class IpFwInitial

Inherits from: IpInterface.

The service entity gains a reference to the IpFwInitial interface for the Framework provider that it wishes to access. This may be gained through a URL, a stringified object reference, etc. At this stage, the service entity has no guarantee that this is a reference to the Framework provider. The service entity uses this interface to initiate the authentication process with the Framework provider. The IpFwInitial interface supports the initiateAuthentication operation to allow the authentication process to take place. This operation must be the first invoked by the service entity. Invocations of other operations will fail until authentication has been successfully completed.

<<Interface>> IpFwInitial
initiateAuthentication (svcDomain : in TpAuthDomain, authType : in TpAuthType) : TpAuthDomain

Method
initiateAuthentication()

The service entity uses this method to initiate the authentication process.

Returns <fwDomain> : This provides the service entity with a framework identifier, and a reference to call the authentication interface of the framework.

```
        structure TpAuthDomain {
                domainID:       TpDomainID;
                authInterface:  IpInterface;
                };
```

The domainID parameter is an identifier for the framework (i.e. TpFwID). It is used to identify the framework to the service entity.The authInterface parameter is a reference to the authentication interface of the framework. The type of this interface is defined by the authType parameter. The service entity uses this interface to authenticate with the framework.

Parameters

svcDomain : in TpAuthDomain

This identifies the service entity to the framework, and provides a reference to the entity's authentication interface.

```
      structure TpAuthDomain {
              domainID:TpDomainID;
              authInterface: IpInterface;
              };
```

The domainID parameter is an identifier either for an existing registered service (i.e. TpServiceID) or for a service supplier (i.e. TpServiceSupplierID). It is used to identify the service (supplier) to the framework, (see authenticate() on IpFwAPILevelAuthentication). If the framework does not recognise the domainID, the framework throws the P_INVALID_DOMAIN_ID exception. The authInterface parameter is a reference to call the authentication interface of the service (supplier). The type of this interface is defined by the authType parameter. If the interface reference is not of the correct type, the framework throws the P_INVALID_INTERFACE_TYPE exception.

authType : in TpAuthType

This identifies the type of authentication mechanism requested by the service entity. It provides the opportunity to use an alternative to the OSA Authentication interface, e.g. an implementation specific authentication mechanism like CORBA Security, using the FwAuthentication interface, or Operator specific Authentication interfaces. OSA Authentication is the default authentication mechanism (P_OSA_AUTHENTICATION). If P_OSA_AUTHENTICATION is selected, then the svcDomain and fwDomain authInterface parameters are references to interfaces of type IpSvc/FwAPILevelAuthentication. If P_AUTHENTICATION is selected, the authInterface parameters are refereces to interfaces of type IpSvc/FwAuthentication which is used when an underlying distibution technology authentication mechanism is used.

Returns

TpAuthDomain

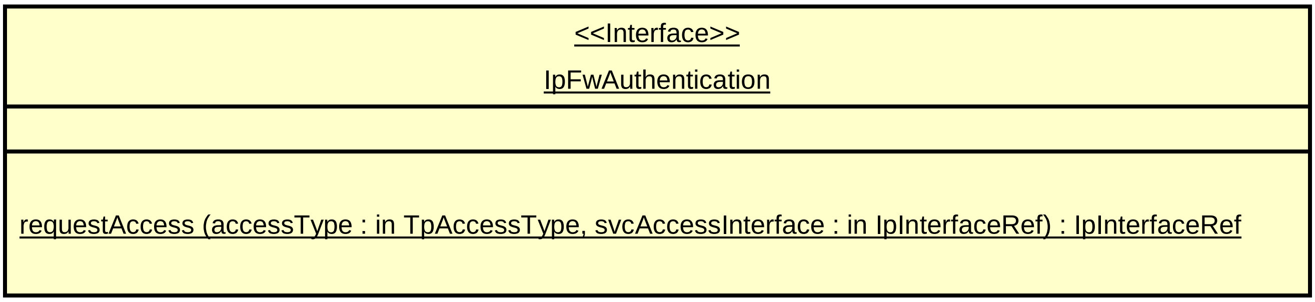
Raises

TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE

12.1.2 Interface Class IpFwAuthentication

Inherits from: IpInterface.

The Authentication Framework interface is used by client to request access to other interfaces supported by the Framework. The mutual authentication process should in this case be done with some underlying distribution technology authentication mechanism, e.g. CORBA Security.



<<Interface>>

IpFwAuthentication

requestAccess (accessType : in TpAccessType, svcAccessInterface : in IpInterfaceRef) : IpInterfaceRef

Method

requestAccess()

Once service entity and framework are authenticated, the service entity invokes the requestAccess operation on the IpFwAuthentication or IpFwAPILevelAuthentication interfaces. This allows the service entity to request the type of access it requires. If it requests P_OSA_ACCESS, then a reference to the IpFwAccess interface is returned. (Operators can define their own access interfaces to satisfy service requirements for different types of access.)

If this method is called before the service entity and framework have successfully completed the authentication process, then the request fails, and the P_ACCESS_DENIED exception is thrown.

Returns <fwAccessinterface> : This provides the reference for the service entity to call the access interface of the framework.

Parameters

`accessType : in TpAccessType`

This identifies the type of access interface requested by the service entity. If the framework does not provide the type of access identified by accessType, then the P_INVALID_ACCESS_TYPE exception is thrown.

`svcAccessInterface : in IpInterfaceRef`

This provides the reference for the framework to call the access interface of the service entity. If the interface reference is not of the correct type, the framework throws the P_INVALID_INTERFACE_TYPE exception.

Returns

`IpInterfaceRef`

Raises

`TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_ACCESS_TYPE, P_INVALID_INTERFACE_TYPE`

12.1.3 Interface Class IpFwAPILevelAuthentication

Inherits from: IpFwAuthentication.

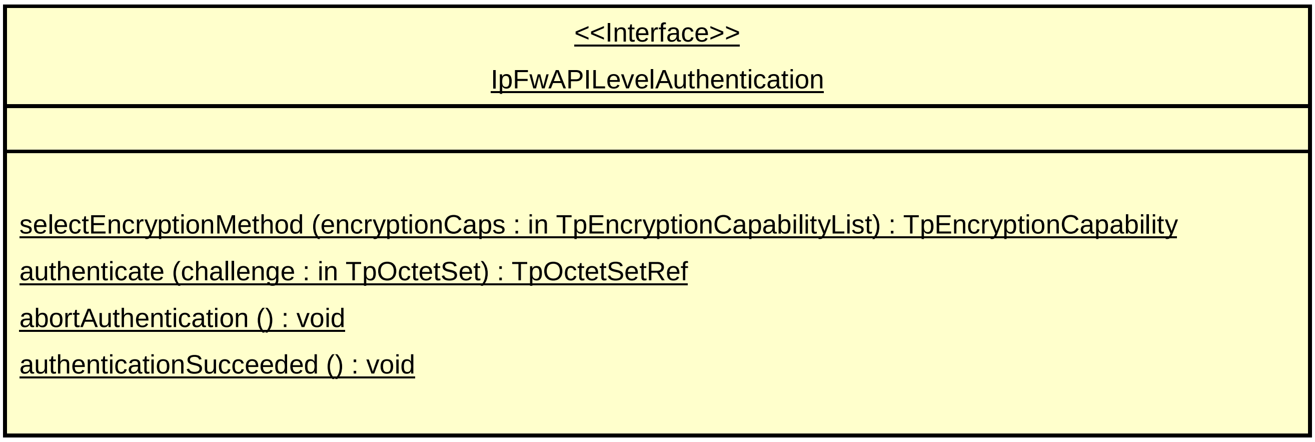
Once the service entity has made initial contact with the provider, authentication of the service entity and Framework provider may be required. The API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. In some cases strong authentication may need to be enforced by the framework provider to prevent misuse of resources. In addition it may be necessary to define the minimum encryption key length that can be used to ensure a high degree of confidentiality. The service entity must authenticate with the framework before it will be able to use any of the other interfaces supported by the framework. Invocations on other interfaces will fail until authentication has been successfully completed

1. The service entity calls initiateAuthentication on the provider's IpFwInitial interface. This allows the service entity to specify the type of authentication process. This authentication process may be specific to the Framework provider, or to the implementation technology used. The initiateAuthentication operation can be used to designate the specific process, (e.g. CORBA security could be used in a CORBA-based implementation of OSA). OSA defines a generic authentication interface (IpFwAPILevelAuthentication), which can be used to perform the authentication process. The initiateAuthentication operation allows the service entity to pass a reference to its IpSvcAPILevelAuthentication interface to the Framework, and receive a reference to the IpFwAPILevelAuthentication interface supported by the framework, in return.

2. The service entity invokes the selectEncryptionMethod on the framework's IpFwAPILevelAuthentication interface. This includes the encryption capabilities of the service entity. The framework then chooses an encryption method based on the encryption capabilities of the service entity and the framework. If the service entity is capable of handling more than one encryption method, then the framework chooses one option, the prescribedMethod. In some instances, the encryption capability of the service entity may not fulfil the demands of the framework, in which case, the authentication will fail.

3. The service entity and framework interact to authenticate each other. For an authentication type of P_OSA_ACCESS, this procedure may consist of a number of messages e.g. a challenge/ response protocol. This authentication protocol is performed using the authenticate operation on the IpFwAPILevelAuthentication interface. P_OSA_ACCESS is based on CHAP, which is primarily a one-way protocol. Mutual authentication is achieved by the framework invoking the authenticate method on the service entity's APILevelAuthentication interface.

NOTE: At any point during the access session, either side can request re-authentication. Re-authentication does not have to be mutual.



<<Interface>>

IpFwAPILevelAuthentication

selectEncryptionMethod (encryptionCaps : in TpEncryptionCapabilityList) : TpEncryptionCapability

authenticate (challenge : in TpOctetSet) : TpOctetSetRef

abortAuthentication () : void

authenticationSucceeded () : void

Method

selectEncryptionMethod()

The service entity uses this method to initiate the authentication process. The framework returns its preferred mechanism. This should be within the capability of the service entity. If a mechanism that is both acceptable to the framework and within the capability of the service entity cannot be found, then the throws the P_NO_ACCEPTABLE ENCRYPTION_CAPABILITY exception. Once the framework has returned its preferred mechanism, it will wait for a predefined unit of time before invoking the service entity's authenticate() method (the wait is to ensure that the service entity can initialise any resources necessary to use the prescribed encryption method)

Returns <prescribedmethod> : This is the mechanism preferred by the framework for the encryption process. If the service entity does not understand the value of the prescribedMethod returned by the framework, it is considered a catastrophic error and the service entity must abort the authentication process.

Parameters

encryptionCaps : in TpEncryptionCapabilityList

This is the means by which the encryption mechanisms supported by the service entity are conveyed to the framework.

Returns

TpEncryptionCapability

Raises

TpCommonExceptions, P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY

Method

authenticate()

The service entity uses this method to authenticate the framework. The challenge will be encrypted using the mechanism prescribed by selectEncryptionMethod. The framework must respond with the correct responses to the challenges presented by the service entity. The serviceID received in the initiateAuthentication() can be used by the framework to reference the correct public key for the service entity (the key management system is currently outside of the scope of the OSA API specification). The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. The invocation of this method may be interleaved with authenticate() calls by the framework on the service entity's APILevelAuthentication interface.

Returns <response> : This is the response of the framework to the challenge of the service entity in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by the selectEncryptionMethod() method.

Parameters

challenge : in TpOctetSet

The challenge presented by the service entity to be responded to by the framework. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

Returns

TpString

Raises

TpCommonExceptions

Method

abortAuthentication()

The service entity uses this method to abort the authentication process. This method is invoked if the service entity no longer wishes to continue the authentication process, (unless the application responded incorrectly to a challenge in which case no further communication with the application should occur.) If this method has been invoked, calls to the requestAccess operation on IpFwAPILevelAuthentication will result in the P_ACCESS_DENIED exception will be thrown, until the service entity has been properly authenticated.

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method

authenticationSucceeded()

The service entity uses this method to inform the Framework of the success of the authentication attempt.

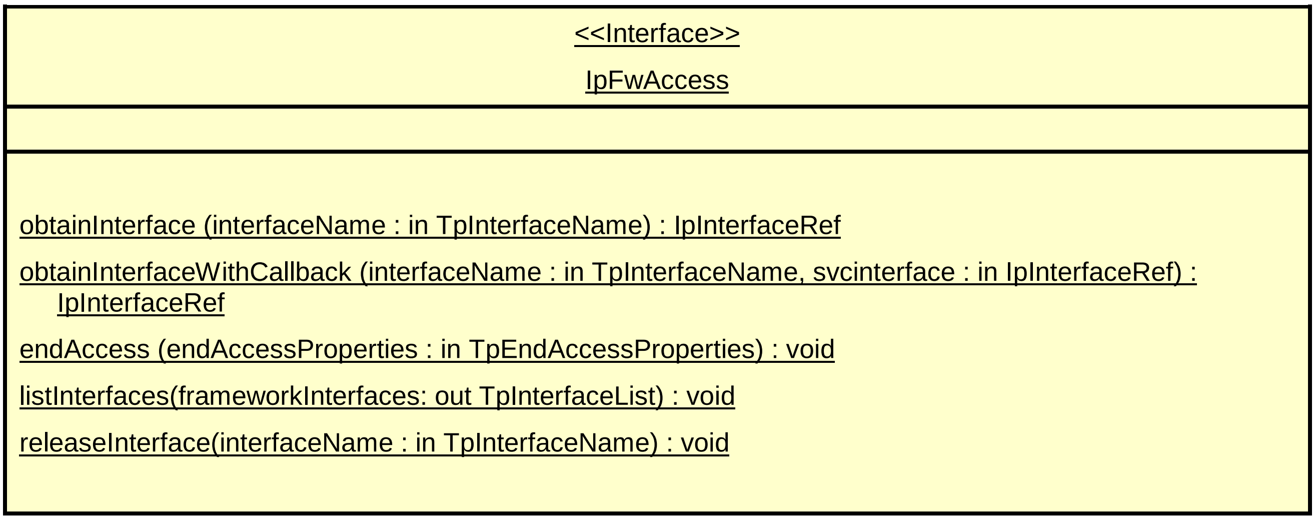
Parameters

No Parameters were identified for this method

12.1.4 Interface Class IpFwAccess

Inherits from: IpInterface.

Once the service entity has authenticated with the framework provider, the service entity can gain access to other framework interfaces. After authentication, the service entity can gain access to the framework's functions, by invoking the requestAccess method on the IpFwAPILevelAuthentication or IpFwAuthentication interfaces. This allows the service entity to request the type of access they require. If they request P_OSA_ACCESS, then a reference to the IpFwAccess interface is returned. (Operators can define their own access interfaces to satisfy service entity requirements for different types of access.) The service entity must also provide the framework with a reference to a 'callback' interface to allow the framework to initiate interactions during the access session. If the service entity has requested P_OSA_ACCESS, then they must provide a reference to a IpSvcAccess interface to the framework. The IpFwAccess interface allows the service entity to gain references to other interfaces offered by the framework. References to these framework interfaces are gained by invoking the obtainInterface, or obtainInterfaceWithCallback operations. The latter is used when a callback interface is supplied to the framework. For example, a service registration interface reference is returned when invoking obtainInterface with "registration" as the interface name. The endAccess operation is used to end the service entity's session with the framework. After it is invoked, the service entity will no longer be authenticated with the framework. The service entity will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail. The IpSvcAccess interface is offered by the service entity to the framework to allow the framework to initiate interactions during the access session. It can be used to terminate the access session and request that the service entity re-authenticate.



<<Interface>>

IpFwAccess

obtainInterface (interfaceName : in TpInterfaceName) : IpInterfaceRef

obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, svcinterface : in IpInterfaceRef) : IpInterfaceRef

endAccess (endAccessProperties : in TpEndAccessProperties) : void

listInterfaces(frameworkInterfaces: out TpInterfaceList) : void

releaseInterface(interfaceName : in TpInterfaceName) : void

Method
obtainInterface()

This method is used to obtain other framework interfaces. The service entity uses this method to obtain interface references to other framework interfaces. (The obtainInterfacesWithCallback method should be used if the service entity is required to supply a callback interface to the framework.)

Returns <fwInterface> : This is the reference to the interface requested.

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface for which a reference is requested. If the interfaceName is invalid, the framework throws the P_INVALID_INTERFACE_NAME exception.

Returns

IpInterfaceRef

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME

Method

obtainInterfaceWithCallback()

This method is used to obtain other framework interfaces. The service entity uses this method to obtain interface references to other framework interfaces, when it is required to supply a callback interface to the framework. (The obtainInterface method should be used when no callback interface needs to be supplied.)

Returns <fwInterface> : This is the reference to the interface requested.

Parameters

interfaceName : in TpInterfaceName

The name of the framework interface for which a reference is requested. If the interfaceName is invalid, the framework throws the P_INVALID_INTERFACE_NAME exception.

svcinterface : in IpInterfaceRef

This is the reference to the service entity interface, which is used for callbacks. If the interface reference is not of the correct type, the framework throws the P_INVALID_INTERFACE_TYPE exception.

Returns

IpInterfaceRef

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME, P_INVALID_INTERFACE_TYPE

Method

endAccess()

The service entity uses this method to end its access session with the framework. After it is invoked, the service entity will no longer be authenticated with the framework. The service entity will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail.

Parameters

endAccessProperties : in TpEndAccessProperties

This is a list of properties that can be used to tell the framework the actions to perform when ending the access session (e.g. existing application sessions may be stopped, or left running). If a property is not recognised by the framework, the P_INVALID_PROPERTY exception is thrown.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_PROPERTY

Method

listInterfaces()

The service entity uses this method to obtain the names of all interfaces supported by the framework. It can then obtain the interfaces it wants to use using either obtainInterface() or obtainInterfaceWithCallback().

Parameters

frameworkInterfaces : out TpInterfaceNameList

The frameworkInterfaces parameter contains a list of interfaces that the framework makes available.

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method

releaseInterface()

The service entity uses this method to release a framework interface that was obtained during this access session.

Parameters

interfaceName : in TpInterfaceName

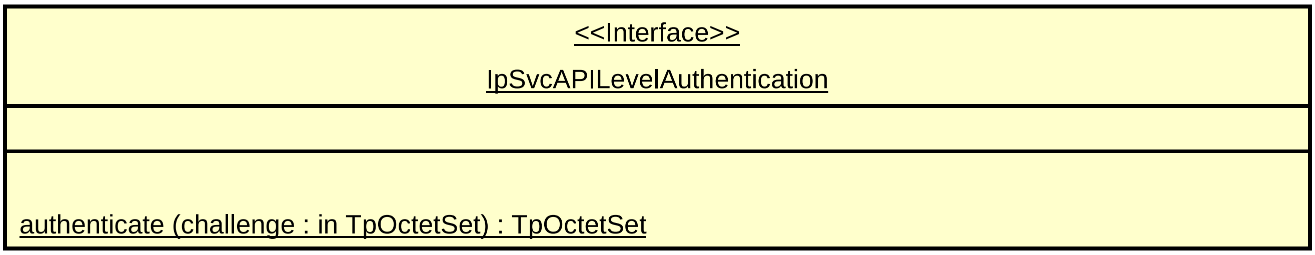
This is the name of the framework interface which is being released. If the interfaceName is invalid, the framework throws the P_INVALID_INTERFACE_NAME exception. If the interface has not been given to the service entity during this access session, then the P_TASK_REFUSED exception will be thrown.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME

12.1.5 Interface Class IpSvcAPILevelAuthentication

Inherits from: IpInterface.


<<Interface>>
IpSvcAPILevelAuthentication
authenticate (challenge : in TpOctetSet) : TpOctetSet

abortAuthentication () : void

authenticationSucceeded () : void

Method

authenticate()

The framework uses this method to authenticate the service entity. The challenge will be encrypted using the mechanism prescribed in selectEncryptionMethod. The service entity must respond with the correct responses to the challenges presented by the framework. The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. The invocation of this method may be interleaved with authenticate() calls by the service entity on the IpFWAPILevelAuthentication interface.

Returns <response> : This is the response of the service entity to the challenge of the framework in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Parameters

challenge : in TpString

The challenge presented by the framework to be responded to by the service entity. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

Returns

TpString

Raises

TpCommonExceptions

Method

abortAuthentication()

The framework uses this method to abort the authentication process. This method is invoked if the framework wishes to abort the authentication process, (unless the service entity responded incorrectly to a challenge in which case no further communication with the service entity should occur.) If this method has been invoked, calls to the requestAccess operation on IpFwAPILevelAuthentication will result in the P_ACCESS_DENIED exception being thrown until the service entity has been properly authenticated.

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

Method

authenticationSucceeded()

The Framework uses this method to inform the service entity of the success of the authentication attempt.

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

12.1.6 Interface Class IpSvcAccess

Inherits from: IpInterface.

<<Interface>> IpSvcAccess
terminateAccess (terminationText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : in TpString) : void

Method

terminateAccess()

This method is used to end the service entity's access session with the framework. The service entity must re-authenticate if it wishes to continue its association with the framework. The service entity will not be able to use the references to any of the framework interfaces gained during the access session. Any method invocations associated with these interfaces will fail. If at any point the framework's level of confidence in the identity of the service entity becomes too low, perhaps due to re-authentication failing, the framework should terminate all outstanding service agreements for that entity and should take steps to terminate the entity's access session WITHOUT invoking terminateAccess() on the service entity. This follows a generally accepted security model where the framework has decided that it can no longer trust the service entity and will therefore sever ALL contact with it.

Parameters

terminationText : in TpString

This is the termination text that describes the reason for the termination of the access session.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the service entity, the P_INVALID_SIGNING_ALGORITHM exception is thrown.

digitalSignature : in TpString

This is a signed version of a hash of the termination text. The framework uses this to confirm its identity to the service entity. The service entity can check that the framework has signed the terminationText. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception is thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNING_ALGORITHM, P_INVALID_SIGNATURE

~~12.1~~12.2 Service Registration Interface Classes

Before a service can be brokered (discovered, subscribed, accessed, etc.) by an enterprise, it has to be registered with the Framework. Services are registered against a particular service type. Therefore service types are created first, and then services corresponding to those types are accepted from the Service Suppliers for registration in the framework. The framework maintains a repository of service types and registered services.

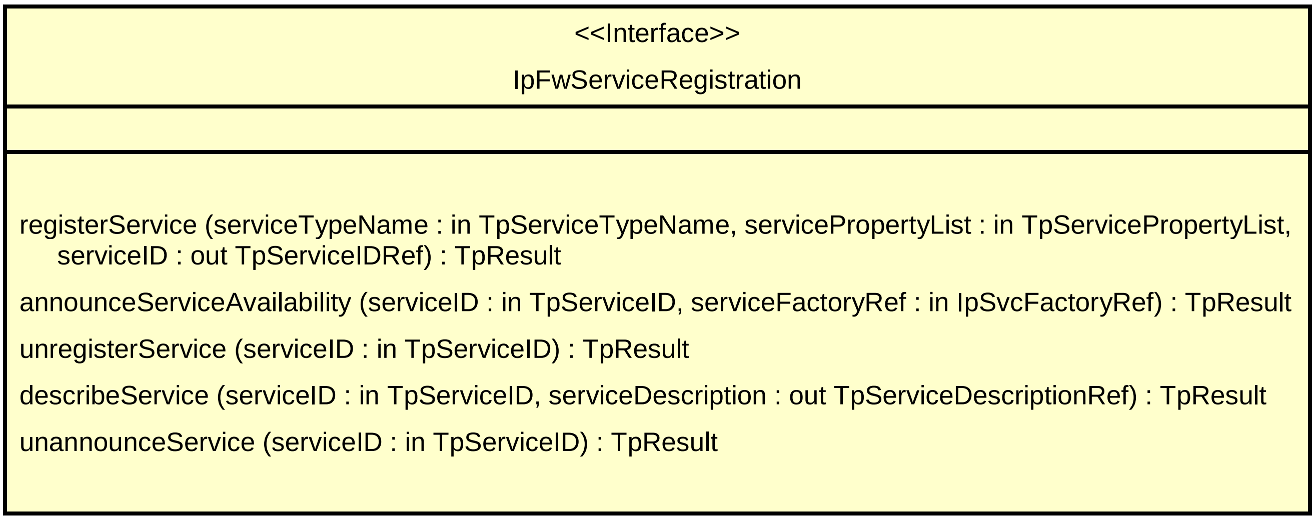
In order to register a new service in the framework, the service supplier must select a service type and the "property values" for the service. The service discovery functionality described in the previous section enables the service supplier to obtain a list of all the service types supported by the framework and their associated sets of service property values.

The Framework service registration-related interfaces are invoked by third party service supplier's administrative applications. They are described below. Note that these methods cannot be invoked until the authentication methods have been invoked successfully.

~~12.1.1~~12.2.1 Interface Class IpFwServiceRegistration

Inherits from: IpInterface.

The Service Registration interface provides the methods used for the registration of network SCFs at the framework.



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

Method
registerService()

The registerService() operation is the means by which a service is registered in the Framework, for subsequent discovery by the enterprise applications . A service-ID is returned to the service supplier when a service is registered in the Framework. The service-ID is the handle with which the service supplier can identify the registered service when needed (e.g. for withdrawing it). The service-ID is only meaningful in the context of the Framework that generated it.

Parameters

serviceTypeName : in TpServiceTypeName

The "serviceTypeName" parameter identifies the service type and a set of named property types that may be used in further describing this service (i.e., it restricts what is acceptable in the servicePropertyList parameter). If the string representation of the "type" does not obey the rules for identifiers, then an P_ILLEGAL_SERVICE_TYPE exception is raised. If the "type" is correct syntactically but the Framework is able to unambiguously determine that it is not a recognised service type, then a P_UNKNOWN_SERVICE_TYPE exception is raised.

servicePropertyList : in TpServicePropertyList

The "servicePropertyList" parameter is a list of property name and property value pairs. They describe the service being registered. This description typically covers behavioral, non-functional and non-computational aspects of the service. Service properties are marked "mandatory" or "readonly". These property mode attributes have the following semantics:

a. mandatory - a service associated with this service type must provide an appropriate value for this property when registering.

b. readonly - this modifier indicates that the property is optional, but that once given a value, subsequently it may not be modified.

Specifying both modifiers indicates that a value must be provided and that subsequently it may not be modified. An example of such properties are those which form part of a service agreement and hence cannot be modified by service suppliers during the life time of service.

If the type of any of the property values is not the same as the declared type (declared in the service type), then a P_PROPERTY_TYPE_MISMATCH exception is raised. If an attempt is made to assign a dynamic property value to a readonly property, then the P_READONLY_DYNAMIC_PROPERTY exception is raised. If the "servicePropertyList" parameter omits any property declared in the service type with a mode of mandatory, then a P_MISSING_MANDATORY_PROPERTY exception is raised. If two or more properties with the same property name are included in this parameter, the P_DUPLICATE_PROPERTY_NAME exception is raised.

serviceID : out TpServiceIDRef

This is the unique handle that is returned as a result of the successful completion of this operation. The Service Supplier can identify the registered service when attempting to access it via other operations such as unregisterService(), etc. Enterprise client applications are also returned this service-ID when attempting to discover a service of this type.

Raises

TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_PROPERTY_TYPE_MISMATCH,P_DUPLICATE_PROPERTY_NAME,
P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE,P_MISSING_MANDATORY_PROPERTY

Method

announceServiceAvailability()

The registerService() method described previously does not make the service discoverable. The announceServiceAvailability() method is invoked after the service is authenticated and its service factory is instantiated at a particular interface. This method informs the framework of the availability of "service factory" of the previously registered service, identified by its service ID, at a specific interface. After the receipt of this method, the framework makes the corresponding service discoverable.

There exists a "service manager"instance per service instance. Each service implements the IpSvcFactory interface. The IpSvcFactory interface supports a method called the createServiceManager(application: in TpClientAppID, serviceManager: out IpServiceRefRef). When the service agreement is signed for some serviceID (using signServiceAgreement()), the framework calls the createServiceManager() for this service, gets a serviceManager and returns this to the client application.

Parameters

`serviceID : in TpServiceID`

The service ID of the service that is being announced. If the string representation of the "serviceID" does not obey the rules for service identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then an P_UNKNOWN_SERVICE_ID exception is raised.

`serviceFactoryRef : in IpSvcFactoryRef`

The interface reference at which the service factory of the previously registered service is available.

Raises

`TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_INVALID_INTERFACE_TYPE`

Method

unregisterService()

The unregisterService() operation is used by the service suppliers to remove a registered service from the Framework. The service is identified by the "service-ID" which was originally returned by the Framework in response to the registerService() operation. The service must be in the SCF Registered state. All instances of the service will be deleted.

Parameters

`serviceID : in TpServiceID`

The service to be withdrawn is identified by the "serviceID" parameter which was originally returned by the registerService() operation. If the string representation of the "serviceID" does not obey the rules for service identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then an P_UNKNOWN_SERVICE_ID exception is raised.

Raises

`TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID`

Method

describeService()

The describeService() operation returns the information about a service that is registered in the framework. It comprises, the "type" of the service , and the "properties" that describe this service. The service is identified by the "service-ID" parameter which was originally returned by the registerService() operation.

The SCS may register various versions of the same SCF, each with a different description (more or less restrictive, for example), and each getting a different serviceID assigned.

Parameters

`serviceID : in TpServiceID`

The service to be described is identified by the "serviceID" parameter which was originally returned by the registerService() operation. If the string representation of the "serviceID" does not obey the rules for object identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then a P_UNKNOWN_SERVICE_ID exception is raised.

serviceDescription : out TpServiceDescriptionRef

This consists of the information about an offered service that is held by the Framework. It comprises the "type" of the service , and the properties that describe this service.

Raises

TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID

Method

unannounceService()

This method results in the service no longer being discoverable by applications. It is, however, still registered and the service ID is still associated with it. Applications currently using the service can continue to use the service but no new applications should be able to start using the service. Also, all unused service tokens relating to the service will be expired. This will prevent anyone who has already performed a selectService() but not yet performed the signServiceAgreement() from being able to obtain a new instance of the service.

Parameters

serviceID : in TpServiceID

The service ID of the service that is being unannounced. If the string representation of the "serviceID" does not obey the rules for service identifiers, then an P_ILLEGAL_SERVICE_ID exception is raised. If the "serviceID" is legal but there is no service offer within the Framework with that ID, then an P_UNKNOWN_SERVICE_ID exception is raised.

Raises

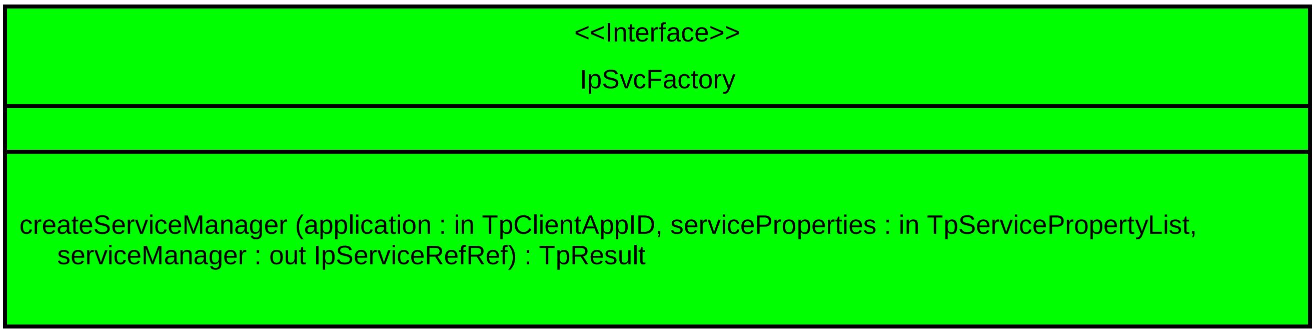
TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID

~~12.2~~12.3 Service Factory Interface Classes

The IpSvcFactory interface allows the framework to get access to a service manager interface of a service. It is used during the signServiceAgreement, in order to return a service manager interface reference to the application. Each service has a service manager interface that is the initial point of contact for the service. E.g., the generic call control service uses the IpCallControlManager interface.

~~12.2.1~~12.3.1 Interface Class IpSvcFactory

Inherits from: IpInterface.



<<Interface>>

IpSvcFactory

createServiceManager (application : in TpClientAppID, serviceProperties : in TpServicePropertyList, serviceManager : out IpServiceRefRef) : TpResult

Method
createServiceManager()

This method returns a new service manager interface reference for the specified application. The service instance will be configured for the client application using the properties agreed in the service level agreement.

Parameters

application : in TpClientAppID

Specifies the application for which the service manager interface is requested.

serviceProperties : in TpServicePropertyList

Specifies the service properties and their values that are to be used to configure the service instance. These properties form a part of the service level agreement. An example of these properties is a list of methods that the client application is allowed to invoke on the service interfaces.

serviceManager : out IpServiceRefRef

Specifies the service manager interface reference for the specified application ID.

Raises

TpCommonExceptions,P_INVALID_PROPERTY

12.4 Service Discovery Interface Classes

This API complements the Service Registration functionality described in another section.

Before a service can be registered in the framework, the service supplier must know what "types" of services the Framework supports and what service "properties" are applicable to each service type. The "listServiceType()" method returns a list of all "service types" that are currently supported by the framework and the "describeServiceType()" method returns a description of each service type. The description of service type includes the "service-specific properties" that are applicable to each service type. Then the service supplier can retrieve a specific set of registered services that both belong to a given type and possess a specific set of "property values", by using the "discoverService()" method.

Additionally the service supplier can retrieve a list of all registered services, without regard to type or property values, by using the "listRegisteredServices()" method. However the scope of the list will depend upon the framework implementation; e.g. a service supplier may only be permitted to retrieve a list of services that the service supplier has previously registered.

12.4.1 Interface Class IpFwServiceDiscovery

Inherits from: IpInterface.

<<Interface>> IpFwServiceDiscovery

listServiceTypes () : TpServiceTypeNameList

describeServiceType (name : in TpServiceTypeName) : TpServiceTypeDescription

discoverService (serviceTypeName : in TpServiceTypeName, desiredPropertyList : in TpServicePropertyList, max : in TpInt32) : TpServiceList

listRegisteredServices () : TpServiceList

Method

listServiceTypes()

This operation returns the names of all service types that are in the repository. The details of the service types can then be obtained using the describeServiceType() method.

Returns <listTypes> : The names of the requested service types.

Parameters

No Parameters were identified for this method

Returns

TpServiceTypeNameList

Raises

TpCommonExceptions

Method

describeServiceType()

This operation lets the caller obtain the details for a particular service type.

Returns <serviceTypeDescription> : The description of the specified service type. The description provides information about: the service properties associated with this service type: i.e. a list of service property {name, mode and type} tuples, the names of the super types of this service type, and whether the service type is currently enabled or disabled.

Parameters

name : in TpServiceTypeName

The name of the service type to be described. If the "name" is malformed, then the P_ILLEGAL_SERVICE_TYPE exception is raised. If the "name" does not exist in the repository, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

Returns

TpServiceTypeDescription

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE

Method

discoverService()

The discoverService operation is the means by which the service supplier can retrieve a specific set of registered services that both belong to a given type and possess a specific set of "property values". The service supplier passes in a list of desired service properties to describe the service it is looking for, in the form of attribute/value pairs for the service properties. The service supplier also specifies the maximum number of matched responses it is willing to accept. The framework must not return more matches than the specified maximum, but it is up to the discretion of the Framework implementation to choose to return less than the specified maximum. The discoverService() operation returns a serviceID/Property pair list for those services that match the desired service property list that the service supplier provided.

Returns <serviceList> : This parameter gives a list of matching services. Each service is characterised by its service ID and a list of service property {name, mode and value list} tuples associated with the service.

Parameters

serviceTypeName : in TpServiceTypeName

The name of the required service type. If the string representation of the "type" does not obey the rules for service type identifiers, then the P_ILLEGAL_SERVICE_TYPE exception is raised. If the "type" is correct syntactically but is not recognised as a service type within the Framework, then the P_UNKNOWN_SERVICE_TYPE exception is raised. The framework may return a service of a subtype of the "type" requested. A service sub-type can be described by the properties of its supertypes.

desiredPropertyList : in TpServicePropertyList

The "desiredPropertyList"parameter is a list of service property {name, mode and value list} tuples that the required services should satisfy. These properties deal with the non-functional and non-computational aspects of the desired service. The property values in the desired property list must be logically interpreted as "minimum", "maximum", etc. by the framework (due to the absence of a Boolean constraint expression for the specification of the service criterion). It is suggested that, at the time of service registration, each property value be specified as an appropriate range of values, so that desired property values can specify an "enclosing" range of values to help in the selection of desired services.

max : in TpInt32

The "max" parameter states the maximum number of services that are to be returned in the "serviceList" result.

Returns

TpServiceList

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE, P_INVALID_PROPERTY

Method

`listRegisteredServices()`

Returns a list of services so far registered in the framework.

Returns <serviceList> : The "serviceList" parameter returns a list of registered services. Each service is characterised by its service ID and a list of service property {name, mode and value list} tuples associated with the service.

Parameters

No Parameters were identified for this method

Returns

`TpServiceList`

Raises

`TpCommonExceptions`

12.5 Integrity Management Interface Classes

12.5.1 Interface Class IpFwFaultManager

Inherits from: IpInterface.

This interface is used by the service instance to inform the framework of events which affect the integrity of the API, and request fault management status information from the framework. The fault manager operations do not exchange callback interfaces as it is assumed that the service instance has supplied its Fault Management callback interface at the time it obtains the Framework's Fault Management interface, by use of the obtainInterfaceWithCallback operation on the IpFwAccess interface.

<<Interface>> IpFwFaultManager
activityTestReq (activityTestID : in TpActivityTestID, testSubject : in TpSubjectType) : void svcActivityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void svcActivityTestErr (activityTestID : in TpActivityTestID) : void appUnavailableInd () : void genFaultStatsRecordReq (timePeriod : in TpTimeInterval, recordSubject : in TpSubjectType) : void svcUnavailableInd (reason : in TpSvcUnavailReason) : void

Method

activityTestReq()

The service instance invokes this method to test that the framework or the client application is operational. On receipt of this request, the framework must carry out a test on itself or on the application, to check that it is operating correctly. The framework reports the test result by invoking the activityTestRes method on the IpSvcFaultManager interface.

Parameters

activityTestID : in TpActivityTestID

The identifier provided by the service instance to correlate the response (when it arrives) with this request.

testSubject : in TpSubjectType

Identifies the subject for testing. (framework or client application).

Raises

TpCommonExceptions

Method

svcActivityTestRes()

The service instance uses this method to return the result of a framework-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the framework to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

svcActivityTestErr()

The service instance uses this method to indicate that an error occurred during a framework-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the framework to correlate this response (when it arrives) with the original request.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

appUnavailableInd()

This method is used by the service instance to inform the framework that the client application is not responding. On receipt of this indication, the framework must act to inform the client application that it should cease use of this service instance.

Parameters

No parameters were identified for this method.

Raises

TpCommonExceptions

Method

genFaultStatsRecordReq()

This method is used by the service instance to solicit fault statistics from the framework. On receipt of this request, the framework must produce a fault statistics record, for the framework or for the application during the specified time interval, which is returned to the service instance using the genFaultStatsRecordRes operation on the IpSvcFaultManager interface.

Parameters

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. A null value leaves this to the discretion of the framework.

testSubject : in TpSubjectType

Specifies the subject to be included in the general fault statistics record. (framework or application).

Raises

TpCommonExceptions

Method

svcUnavailableInd()

This method is used by the service instance to inform the framework that it is about to become unavailable for use. The framework should inform the client application that is currently using this service instance that it is unavailable for use (via the svcUnavailableInd method on the IpAppFaultManager interface).

Parameters

reason : in TpSvcUnavailReason

Identifies the reason for the service instance's unavailability.

Raises

TpCommonExceptions

12.5.2 Interface Class IpSvcFaultManager

Inherits from: IpInterface.

This interface is used to inform the service instance of events that affect the integrity of the Framework, Service or Client Application. The Framework will invoke methods on the Fault Management Service Interface that is specified when the service instance obtains the Fault Management Framework interface: i.e. by use of the obtainInterfaceWithCallback operation on the IpFwAccess interface

<<Interface>> IpSvcFaultManager
activityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void activityTestErr (activityTestID : in TpActivityTestID) : void svcActivityTestReq (activityTestID : in TpActivityTestID) : void fwFaultReportInd (fault : in TpInterfaceFault) : void fwFaultRecoveryInd (fault : in TpInterfaceFaultRef) : void fwUnavailableInd (reason : in TpFwUnavailReason) : void svcUnavailableInd () : void appUnavailableInd () : void genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord, recordSubject : in TpSubjectType) : void genFaultStatsRecordErr (faultStatisticsError : in TpFaultStatisticsError, recordSubject : in TpSubjectType) : void

Method
activityTestRes()

The framework uses this method to return the result of a service-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the service to correlate this response (when it arrives) with the original request.

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method
activityTestErr()

The framework uses this method to indicate that an error occurred during a service-requested activity test.

Parameters

activityTestID : in TpActivityTestID

Used by the service instance to correlate this response (when it arrives) with the original request.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

svcActivityTestReq()

The framework invokes this method to test that the service instance is operational. On receipt of this request, the service instance must carry out a test on itself, to check that it is operating correctly. The service instance reports the test result by invoking the svcActivityTestRes method on the IpFwFaultManager interface.

Parameters

activityTestID : in TpActivityTestID

The identifier provided by the framework to correlate the response (when it arrives) with this request.

Raises

TpCommonExceptions

Method

fwFaultReportInd()

The framework invokes this method to notify the service instance of a failure within the framework. The service instance must not continue to use the framework until it has recovered (as indicated by a fwFaultRecoveryInd).

Parameters

fault : in TpInterfaceFault

Specifies the fault that has been detected by the framework.

Raises

TpCommonExceptions

Method

fwFaultRecoveryInd()

The framework invokes this method to notify the service instance that a previously reported fault has been rectified. The service instance may then resume using the framework.

Parameters

`fault : in TpInterfaceFaultRef`

Specifies the fault from which the framework has recovered.

Raises

`TpCommonExceptions`

Method

`fwUnavailableInd()`

The framework invokes this method to inform the service instance that it is no longer available.

Parameters

`reason : in TpFwUnavailReason`

Identifies the reason why the framework is no longer available

Raises

`TpCommonExceptions`

Method

`svcUnavailableInd()`

The framework invokes this method to inform the service instance that the client application has reported that it can no longer use the service instance (either due to a failure in the client application or in the service instance itself). The service instance should assume that the client application is leaving the service session and should act accordingly to terminate the session from its own end too.

Parameters

No parameters were identified for this method.

Raises

`TpCommonExceptions`

Method

`appUnavailableInd()`

The framework invokes this method to inform the service instance that the client application is ceasing its current use of the service. This may be a result of the application reporting a failure. Alternatively, the framework may have detected that the application has failed: e.g. non-response from an activity test, failure to return heartbeats.

Parameters

None identified for this method.

Raises

TpCommonExceptions

Method

genFaultStatsRecordRes()

This method is used by the framework to provide fault statistics to a service in response to a genFaultStatsRecordReq method invocation on the IpFwFaultManager interface.

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

recordSubject : in TpSubjectType

Specifies the entity (framework or applications) whose fault statistics record has been provided.

Raises

TpCommonExceptions

Method

genFaultStatsRecordErr()

This method is used by the framework to indicate an error fulfilling the request to provide fault statistics, in response to a genFaultStatsRecordReq method invocation on the IpFwFaultManager interface.

Parameters

faultStatisticsError : in TpFaultStatsError

The fault statistics error.

recordSubject : in TpSubjectType

Specifies the entity (framework or application) whose fault statistics record was requested.

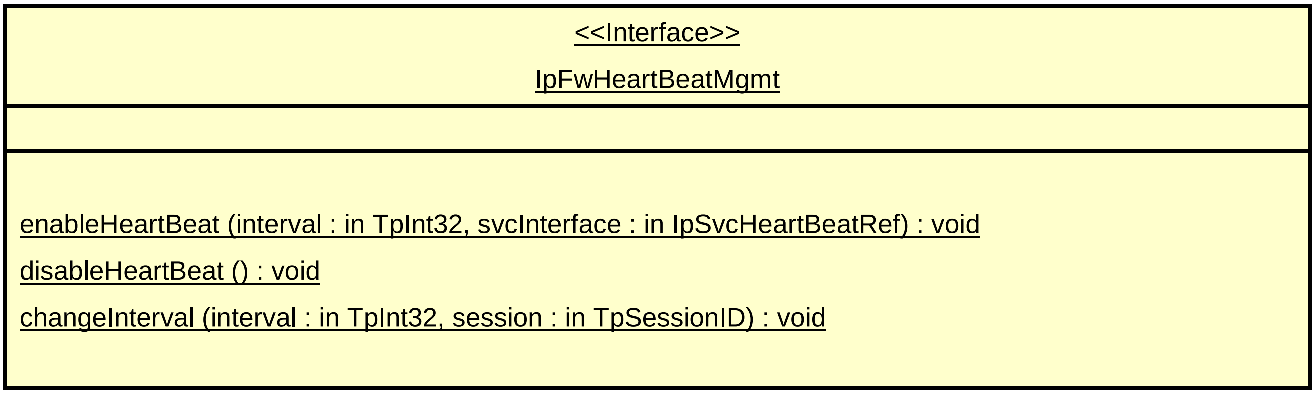
Raises

TpCommonExceptions

12.5.3 Interface Class IpFwHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the framework by a service instance.



<<Interface>>

IpFwHeartBeatMgmt

enableHeartBeat (interval : in TpInt32, svcInterface : in IpSvcHeartBeatRef) : void

disableHeartBeat () : void

changeInterval (interval : in TpInt32, session : in TpSessionID) : void

Method

enableHeartBeat()

With this method, the service instance instructs the framework to begin sending its heartbeat to the specified interface at the specified interval.

Parameters

interval : in TpInt32

The duration in milliseconds between the heartbeats.

svcInterface : in IpSvcHeartBeatRef

This parameter refers to the callback interface the heartbeat is calling.

Raises

TpCommonExceptions, P_INVALID_INTERFACE_TYPE

Method

disableHeartBeat()

Instructs the framework to cease the sending of its heartbeat.

Parameters

None identified.

Raises

TpCommonExceptions

Method

changeInterval()

Allows the administrative change of the heartbeat interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

Raises

TpCommonExceptions

12.5.4 Interface Class IpFwHeartBeat

Inherits from: IpInterface.

The service side framework heartbeat interface is used by the service instance to send the framework its heartbeat.

<<Interface>> IpFwHeartBeat
pulse () : void

Method

pulse()

The service instance uses this method to send its heartbeat to the framework. The framework will be expecting a pulse at the end of every interval specified in the parameter to the IpSvcHeartBeatMgmt.enableSvcHeartbeat() method. If the pulse() is not received within the specified interval, then the service instance can be deemed to have failed the heartbeat.

Parameters

None.

Raises

TpCommonExceptions

12.5.5 Interface Class IpSvcHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the service instance by the framework.

<<Interface>> IpSvcHeartBeatMgmt

enableSvcHeartBeat (interval : in TpInt32, fwInterface : in IpFwHeartBeatRef) : void

disableSvcHeartBeat () : void

changeInterval (interval : in TpInt32, session : in TpSessionID) : void

Method

`enableSvcHeartBeat()`

With this method, the framework instructs the service instance to begin sending its heartbeat to the specified interface at the specified interval..

Parameters

`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

`fwInterface : in IpFwHeartBeatRef`

This parameter refers to the callback interface the heartbeat is calling.

Raises

`TpCommonExceptions, P_INVALID_INTERFACE_TYPE`

Method

`disableSvcHeartBeat()`

Instructs the service instance to cease the sending of its heartbeat.

Parameters

None identified.

Raises

`TpCommonExceptions`

Method

`changeInterval()`

Allows the administrative change of the heartbeat interval

Parameters

`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

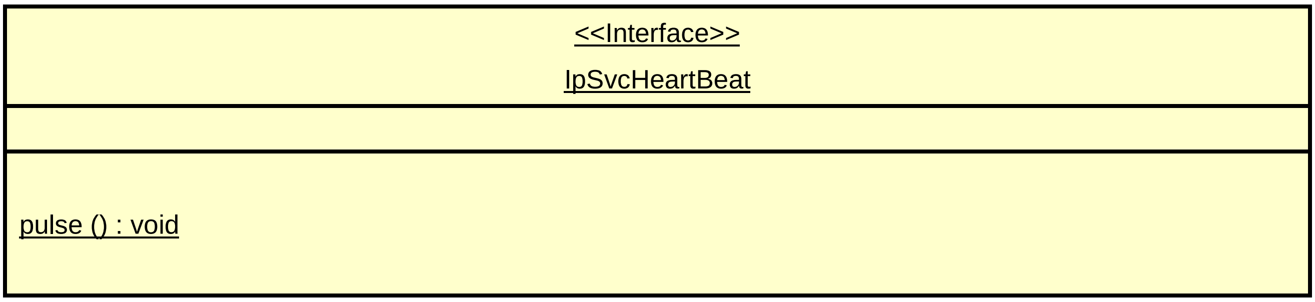
Raises

`TpCommonExceptions`

12.5.6 Interface Class IpSvcHeartBeat

Inherits from: IpInterface.

The service heartbeat interface is used by the framework to send the service instance its heartbeat.



Method

pulse()

The framework uses this method to send its heartbeat to the service instance. The service will be expecting a pulse at the end of every interval specified in the parameter to the IpFwHeartBeatMgmt.enableHeartbeat() method. If the pulse() is not received within the specified interval, then the framework can be deemed to have failed the heartbeat.

Parameters

None identified.

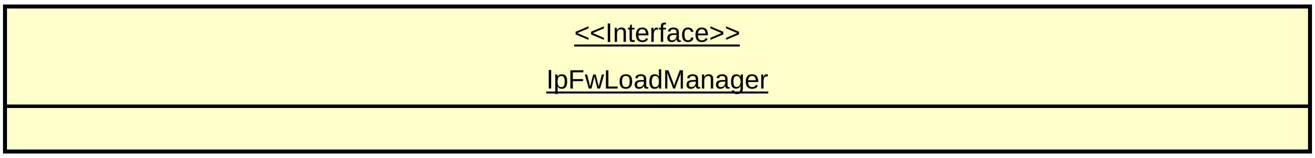
Raises

TpCommonExceptions

12.5.7 Interface Class IpFwLoadManager

Inherits from: IpInterface.

The framework API should allow the load to be distributed across multiple machines and across multiple component processes, according to a load management policy. The separation of the load management mechanism and load management policy ensures the flexibility of the load management services. The load management policy identifies what load management rules the framework should follow for the specific service. It might specify what action the framework should take as the congestion level changes. For example, some real-time critical applications will want to make sure continuous service is maintained, below a given congestion level, at all costs, whereas other services will be satisfied with disconnecting and trying again later if the congestion level rises. Clearly, the load management policy is related to the QoS level to which the application is subscribed. The framework load management function is represented by the IpFwLoadManager interface. To handle responses and reports, the service developer must implement the IpSvcLoadManager interface to provide the callback mechanism.



reportLoad (loadLevel : in TpLoadLevel) : void

queryLoadReq (querySubject : in TpSubjectType, timeInterval : in TpTimeInterval) : void

querySvcLoadRes (loadStatistics : in TpLoadStatisticList) : void

querySvcLoadErr (loadStatisticError : in TpLoadStatisticError) : void

createLoadLevelNotification (notifSubject : in TpSubjectType) : void

destroyLoadLevelNotification (notifSubject : in TpSubjectType: void

suspendNotification (notifSubject : in TpSubjectType) : void

resumeNotification (notifSubject : in TpSubjectType) : void

Method

reportLoad()

The service instance uses this method to report its current load level (0,1, or 2) to the framework: e.g. when the load level on the service instance has changed.

At level 0 load, the service instance is performing within its load specifications (i.e. it is not congested or overloaded). At level 1 load, the service instance is overloaded. At level 2 load, the service instance is severely overloaded.

Parameters

loadLevel : in TpLoadLevel

Specifies the service instance 's load level.

Raises

TpCommonExceptions

Method

queryLoadReq()

The service instance uses this method to request the framework to provide load statistics records for the framework or for the application that uses the service instance.

Parameters

querySubject : in TpSubjectType

Specifies the entity (framework or applications) for which load statistics records should be reported

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistics records should be reported.

Raises

TpCommonExceptions

Method

querySvcLoadRes()

The service instance uses this method to send load statistic records back to the framework that requested the information; i.e. in response to an invocation of the querySvcLoadReq method on the IpSvcLoadManager interface.

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the service-supplied load statistics.

Raises

TpCommonExceptions

Method

querySvcLoadErr()

The service instance uses this method to return an error response to the framework that requested the service instance 's load statistics information, when the service instance is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the querySvcLoadReq method on the IpSvcLoadManager interface.

Parameters

loadStatisticError : in TpLoadStatisticError

Specifies the error code associated with the failed attempt to retrieve the service instance 's load statistics.

Raises

TpCommonExceptions

Method

createLoadLevelNotification()

The service uses this method to register to receive notifications of load level changes associated with the framework or with the application that uses the service instance.

Parameters

notifSubject : in TpSubjectType

Specifies the entity (framework or application) for which load level changes should be reported.

Raises

TpCommonExceptions

Method

destroyLoadLevelNotification()

The service uses this method to unregister for notifications of load level changes associated with the framework or with the application that uses the service instance.

Parameters

notifSubject : in TpSubjectType

Specifies the entity (framework or application) for which load level changes should no longer be reported.

Raises

TpCommonExceptions

Method

suspendNotification()

The service instance uses this method to request the framework to suspend sending it notifications associated with the framework or with the application that uses the service instance; e.g. while the service instance handles a temporary overload condition.

Parameters

notifSubject : in TpSubjectType

Specifies the entity (framework or application) for which the sending of notifications by the framework should be suspended.

Raises

TpCommonExceptions

Method

resumeNotification()

The service instance uses this method to request the framework to resume sending it notifications associated with the framework or with the application that uses the service instance; e.g. after a period of suspension during which the service instance handled a temporary overload condition.

Parameters

notifSubject : in TpSubjectType

Specifies the entity (framework or application) for which the sending of notifications of load level changes by the framework should be resumed.

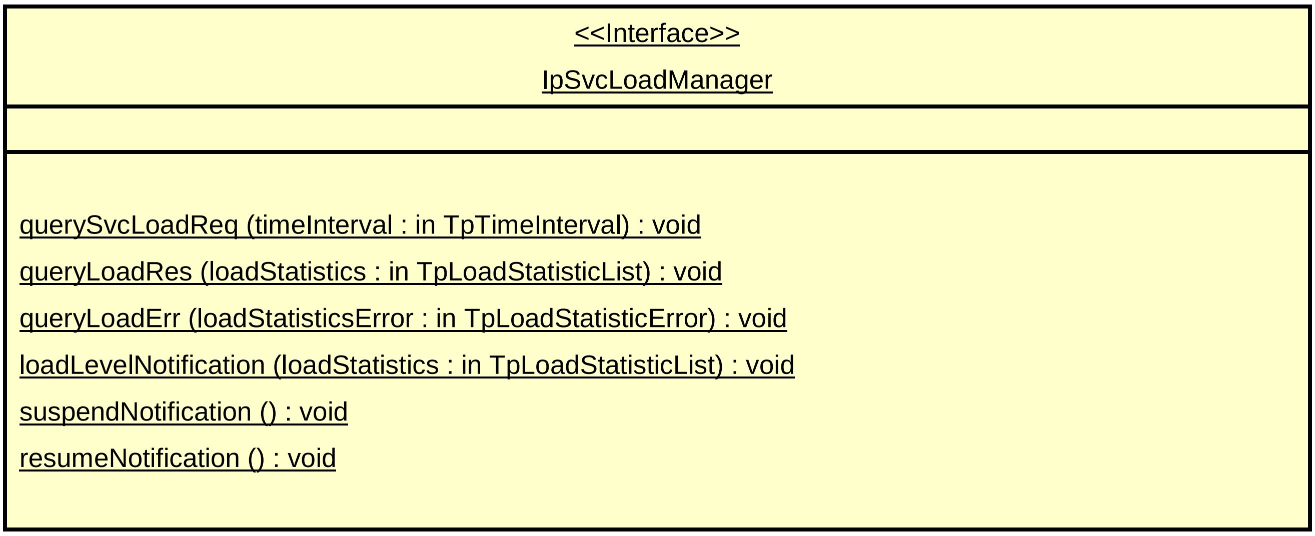
Raises

TpCommonExceptions

12.5.8 Interface Class IpSvcLoadManager

Inherits from: IpInterface.

The service developer supplies the load manager service interface to handle requests, reports and other responses from the framework load manager function. The service instance supplies the identity of its callback interface at the time it obtains the framework's load manager interface, by use of the obtainInterfaceWithCallback() method on the IpFwAccess interface.



<<Interface>>

IpSvcLoadManager

querySvcLoadReq (timeInterval : in TpTimeInterval) : void

queryLoadRes (loadStatistics : in TpLoadStatisticList) : void

queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void

loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void

suspendNotification () : void

resumeNotification () : void

Method

querySvcLoadReq()

The framework uses this method to request the service instance to provide its load statistic records.

Parameters

timeInterval : in TpTimeInterval

Specifies the time interval for which load statistic records should be reported.

Raises

`TpCommonExceptions`

Method

`queryLoadRes()`

The framework uses this method to send load statistic records back to the service instance that requested the information; i.e. in response to an invocation of the queryLoadReq method on the IpFwLoadManager interface.

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the framework-supplied load statistics

Raises

`TpCommonExceptions`

Method

`queryLoadErr()`

The framework uses this method to return an error response to the service-instance that requested the framework's load statistics information, when the framework is unsuccessful in obtaining any load statistic records; i.e. in response to an invocation of the queryLoadReq method on the IpFwLoadManager interface.

Parameters

`loadStatisticsError : in TpLoadStatisticError`

Specifies the error code associated with the failed attempt to retrieve the framework's load statistics.

Raises

`TpCommonExceptions`

Method

`loadLevelNotification()`

Upon detecting load condition change, (e.g. load level changing from 0 to 1, 0 to 2, 1 to 0, for the application~~s~~ or framework which have been registered for load level notifications) this method is invoked on the SCF.

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the framework-supplied load statistics, which include the load level change(s).

Raises

TpCommonExceptions

Method
suspendNotification()

The framework uses this method to request the service instance to suspend sending it any notifications: e.g. while the framework handles a temporary overload condition.

Parameters
No Parameters were identified for this method

Raises

TpCommonExceptions

Method
resumeNotification()

The framework uses this method to request the service instance to resume sending it notifications: e.g. after a period of suspension during which the framework handled a temporary overload condition.

Parameters
No Parameters were identified for this method

Raises

TpCommonExceptions

12.5.9 Interface Class IpFwOAM

Inherits from: IpInterface.

The OAM interface is used to query the system date and time. The service and the framework can synchronise the date and time to a certain extent. Accurate time synchronisation is outside the scope of this API.

<<Interface>> IpFwOAM
systemDateTimeQuery (clientDateAndTime : in TpDateAndTime) : TpDateAndTime

Method

systemDateTimeQuery()

This method is used to query the system date and time. The client (service) passes in its own date and time to the framework. The framework responds with the system date and time.

Returns <systemDateAndTime> : This is the system date and time of the framework.

Parameters

clientDateAndTime : in TpDateAndTime

This is the date and time of the client (service). The P_INVALID_DATE_TIME_FORMAT exception is thrown if the format of the parameter is invalid.

Returns

TpDateAndTime

Raises

TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT

12.5.10 Interface Class IpSvcOAM

Inherits from: IpInterface.

<<Interface>>

IpSvcOAM

systemDateTimeQuery (systemDateAndTime : in TpDateAndTime) : TpDateAndTime

Method

systemDateTimeQuery()

This method is used by the framework to send the system date and time to the service. The service responds with its own date and time.

Returns <clientDateAndTime> : This is the date and time of the client (service).

Parameters

`systemDateAndTime : in TpDateAndTime`

This is the system date and time of the framework. The P_INVALID_DATE_TIME_FORMAT exception is thrown if the format of the parameter is invalid.

Returns

`TpDateAndTime`

Raises

`TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT`

12.6 Event Notification Interface Classes

12.6.1 Interface Class IpFwEventNotification

Inherits from: IpInterface.

The event notification mechanism is used to notify the service of generic events that have occurred.

<<Interface>> IpFwEventNotification
createNotification (eventCriteria : in TpFwEventCriteria) : TpAssignmentID destroyNotification (assignmentID : in TpAssignmentID) : void

Method

`createNotification()`

This method is used to install generic notifications so that events can be sent to the service.

Returns <assignmentID> : Specifies the ID assigned by the framework for this newly installed event notification.

Parameters

`eventCriteria : in TpFwEventCriteria`

Specifies the event specific criteria used by the service to define the event required.

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_INVALID_EVENT_TYPE, P_INVALID_CRITERIA

Method

destroyNotification()

This method is used by the service to delete generic notifications from the framework.

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID given by the framework when the previous createNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will throw the P_INVALID_ASSIGNMENT_ID exception.

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

12.6.2 Interface Class IpSvcEventNotification

Inherits from: IpInterface.

This interface is used by the framework to inform the service of a generic event. The Event Notification Framework will invoke methods on the Event Notification Service Interface that is specified when the Event Notification interface is obtained.

<<Interface>>

IpSvcEventNotification

reportNotification (eventInfo : in TpFwEventInfo, assignmentID : in TpAssignmentID) : void

notificationTerminated () : void

Method

reportNotification()

This method notifies the service of the arrival of a generic event.

Parameters

eventInfo : in TpFwEventInfo

Specifies specific data associated with this event.

assignmentID : in TpAssignmentID

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

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

Method

notificationTerminated()

This method indicates to the service that all generic event notifications have been terminated (for example, due to faults detected).

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

13 Framework-to-Service State Transition Diagrams

13.1 Trust and Security Management State Transition Diagrams

There are no State Transition Diagrams defined for Trust and Security Management

~~13.1~~13.2 Service Registration State Transition Diagrams

~~13.1.1~~13.2.1 State Transition Diagrams for IpFwServiceRegistration

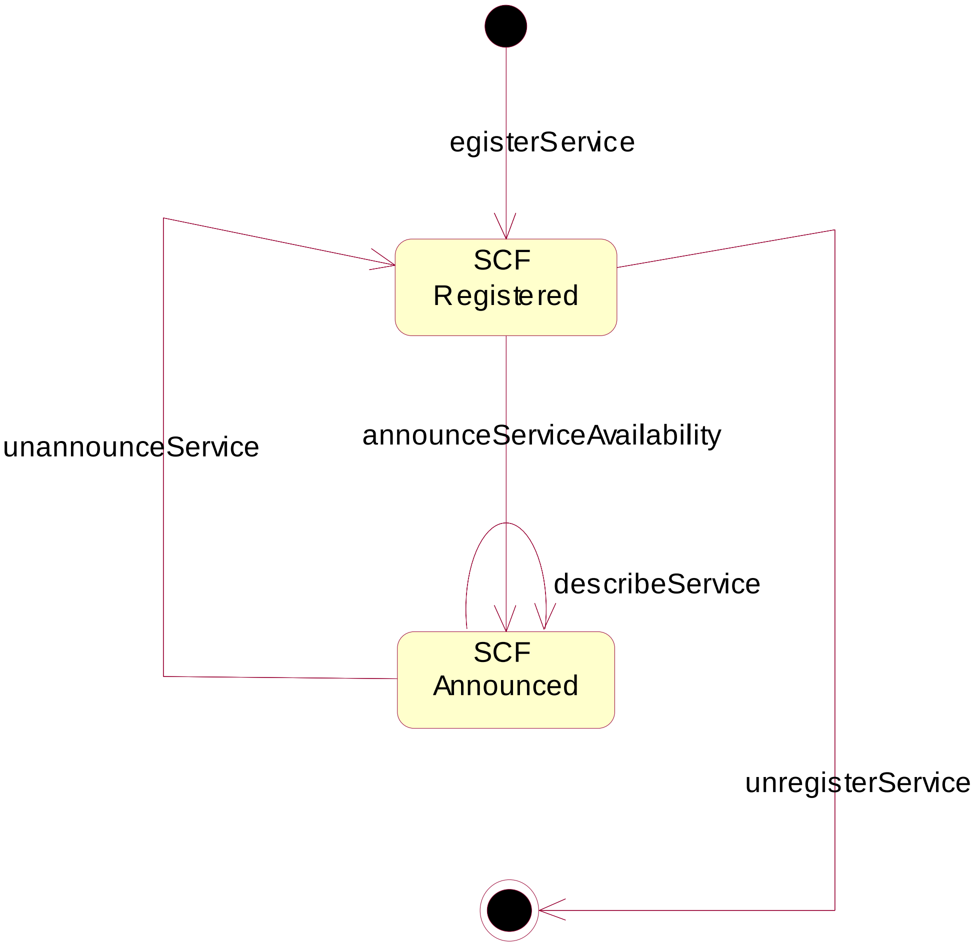


Figure : State Transition Diagram for IpFwServiceRegistration

~~13.1.1.1~~13.2.1.1 SCF Registered State

This is the state entered when a Service Capability Server (SCS) registers its SCF in the Framework, by informing it of the existence of an SCF characterised by a service type and a set of service properties. As a result the Framework associates a service ID to this SCF, that will be used to identify it by both sides.

An SCF may be unregistered, the service ID then being no longer associated with the SCF.

~~13.1.1.2~~13.2.1.2 SCF Announced State

This is the state entered when the existence of the SCF has been announced, thus making it available for discovery by applications. The SCF can be unannounced at any time, taking it back into the SCF Registered state where it is no longer available for discovery.

~~13.2~~13.3 Service Factory State Transition Diagrams

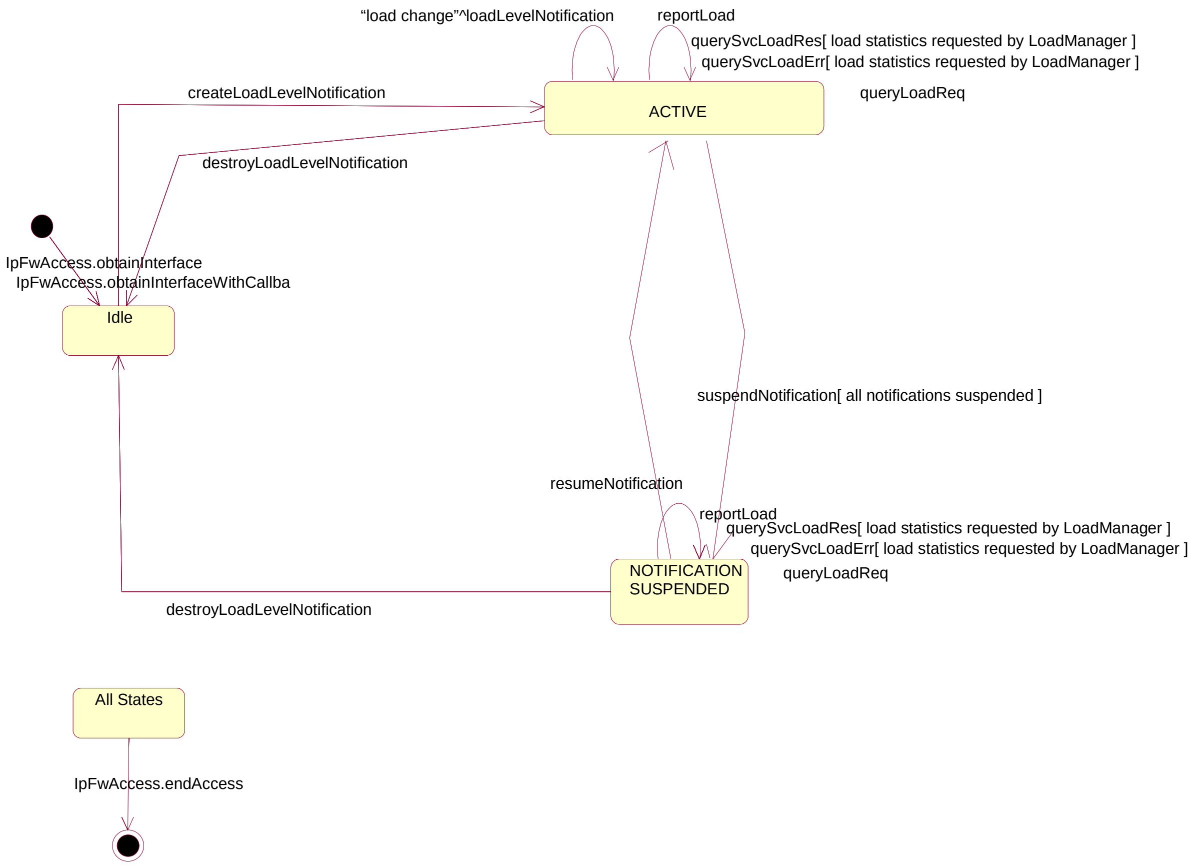
There are no State Transition Diagrams defined for Service Factory

13.4 Service Discovery State Transition Diagrams

There are no State Transition Diagrams defined for Service Discovery

13.5 Integrity Management State Transition Diagrams

13.5.1 State Transition Diagram for IpFwLoadManager


"load change"^loadLevelNotification
reportLoad
querySvcLoadRes[load statistics requested by LoadManager]
querySvcLoadErr[load statistics requested by LoadManager]
createLoadLevelNotification
ACTIVE
queryLoadReq
destroyLoadLevelNotification
IpFwAccess.obtainInterface
IpFwAccess.obtainInterfaceWithCallba
Idle
suspendNotification[all notifications suspended]
resumeNotification
reportLoad
querySvcLoadRes[load statistics requested by LoadManager]
querySvcLoadErr[load statistics requested by LoadManager]
NOTIFICATION SUSPENDED
queryLoadReq
destroyLoadLevelNotification
All States
IpFwAccess.endAccess

IDLE

In this state the service has obtained an interface reference to the IpFwLoadManager from the IpFwAccess interface.

ACTIVE

In this state the service has indicated its interest in notifications by performing a createLoadLevelNotification() invocation on the IpFwLoadManager. The load manager can now request the service to supply load statistics information (by invoking querySvcLoadReq()). Furthermore the LoadManager can request the service to control its load (by invoking loadLevelNotification(), resumeNotification() or suspendNotification() on the service side of interface). In case the service detects a change in load level, it reports this to the LoadManager by calling the method reportLoad().

NOTIFICATION SUSPENDED

Due to, e.g. a temporary load condition, the service has requested the load manager to suspend sending the load level notification information.

13.6 Event Notification State Transition Diagrams

There are no State Transition Diagrams defined for Event Notification

14

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010712**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 017** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

*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:	⌘	Removal of P_SERVICE_ACCESS_TYPE		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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:	⌘	29.198-3 contains a P SERVICE ACCESS TYPE exception that has an identical description to that of P INVALID ACCESS TYPE. No method in the framework is capable of throwing this exception, and the exception wasn't in Parlay 2.1.
Summary of change:	⌘	To remove the P_SERVICE_ACCESS_TYPE exception from 29.198-3.
Consequences if not approved:	⌘	P_SERVICE_ACCESS_TYPE exception is left redundant. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	16	
Other specs affected:	⌘	□ Other core specifications □ Test specifications □ O&M Specifications	⌘
Other comments:	⌘		

Resulting changes

16 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_DUPLICATE_PROPERTY_NAME	A dupilcate property name has been received
P_ILLEGAL_SERVICE_ID	Illegal Service ID
P_ILLEGAL_SERVICE_TYPE	Illegal Service Type
P_INVALID_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_INVALID_ACTIVITY_TEST_ID	ID does not correspond to a valid activity test request
P_INVALID_AGREEMENT_TEXT	Invalid agreement text
P_INVALID_AUTH_CAPABILITY	Invalid authentication capability
P_INVALID_AUTH_TYPE	Invalid type of authentication mechanism
P_INVALID_CLIENT_APP_ID	Invalid Client Application ID
P_INVALID_DOMAIN_ID	Invalid client ID
P_INVALID_ENT_OP_ID	Invalid Enterprise Operator ID
P_INVALID_PROPERTY	The framework does not recognise the property supplied by the client
P_INVALID_SAG_ID	Invalid Subscription Assignment Group ID
P_INVALID_SERVICE_CONTRACT_ID	Invalid Service Contract ID
P_INVALID_SERVICE_ID	Invalid service ID
P_INVALID_SERVICE_PROFILE_ID	Invalid service profile ID
P_INVALID_SERVICE_TOKEN	The service token has not been issued, or it has expired.
P_INVALID_SERVICE_TYPE	Invalid Service Type
P_INVALID_SIGNATURE	Invalid digital signature
P_INVALID_SIGNING_ALGORITHM	Invalid signing algorithm
P_MISSING_MANDATORY_PROPERTY	Mandatory Property Missing
P_NO_ACCEPTABLE_AUTH_CAPABILITY	An authentication mechanism, which is acceptable to the framework, is not supported by the client
P_PROPERTY_TYPE_MISMATCH	Property Type Mismatch
P_SERVICE_ACCESS_DENIED	The client application is not allowed to access this service.
~~P_SERVICE_ACCESS_TYPE~~	~~The framework does not support the type of access interface requested by the client.~~
P_SERVICE_NOT_ENABLED	The service ID does not correspond to a service that has been enabled
P_UNKNOWN_SERVICE_ID	Unknown Sevice ID
P_UNKNOWN_SERVICE_TYPE	Unknown Service Type

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010713**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**018**	⌘	ev	**-**	⌘	Current version:	**4.1.0** ⌘

*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:	⌘	Confusing meaning of prescribedMethod		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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:	⌘	Various sections of the specification convey confusion over the role of authentication methods and authentication capabilities (encryption method)
Summary of change:	⌘	It is proposed to remove references to the authentication capabilities and replace them with references to encryption methods to clarify the roles. The STD for IpAPILevelAuthentication has been reworked. It is also proposed to remove the prescribedMethod parameter from the authenticate() methods.
Consequences if not approved:	⌘	Confusion over the role and types of authentication capabilities will remain in the specification. The TS will be ambiguous and difficult to implement correctly – interworking will be jeopardised. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	4, 6.4.2, 6.4.4, 8.1.5, 9.1.2, 15.3.3, 15.3.4, and 16		
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.

Problem

The concept of authentication capabilities seems to be muddled within the specification. The method which is used to select these capabilities has already been renamed to selectEncryptionMethod(), because that is what the method is actually used for. None, RSA512, RSA1024, DES56 and DES128 are the values for TpAuthCapability. These are methods used for encrypting the challenge and are therefore encryption capabilities, not authentication capabilities. Although RSA can be used as an authentication capability (due to its support for digital signatures) it is not used as such within the framework.

However, there are many places in the specification which state that the prescribedMethod (returned from selectEncryptionMethod) is used to determine how many challenge/response exchanges have to occur. The values for the authentication capabilities do not specify the number of challenge/response exchanges that need to be executed, only the method of encryption for the challenge.

Whether challenge/response exchanges are used for authentication or not is dependent on the method of authentication decided at initiateAuthentication time. The authentication method of P_OSA_AUTHENTICATION is based on CHAP, and can therefore utilise several challenge/response exchanges (until the side initiating the challenge is satisfied that the far side has authenticated correctly).

CHAP authentication is one-way, but mutual authentication (two-way) can be used if the party being authenticated decides to authenticate the other. There is no requirement within CHAP, and therefore should be no requirement within the P_OSA_AUTHENTICATION authentication type, that mutual authentication must be performed.

Proposal

Lucent believes that various sections of the specification need to be updated to remove any confusion over the role of authentication types and authentication capabilities (encryption method). We would also like to remove references to the authentication capabilities and replace them with references to encryption methods to clarify the roles.

The STD for IpAPILevelAuthentication will need to be reworked as it currently shows a failed selectEncryptionMethod resulting in the object moving to the sink state. We don't feel that the object should automatically move to the sink state just because no matching encryption method could be found in the first invocation of selectEncryptionMethod. There is no reference to this in the text for selectEncryptionMethod. Moving back to the IDLE state would allow the application to re-evaluate its encryption capabilities before either trying again or invoking abortAuthentication (it is assumed that there is a guard timer to hold against the application holding the object indefinitely). Besides which, the exception shown for this case is incorrect in the STD.

Any text referencing one-way or two-way authentication will need to be modified as CHAP (which the authentication method used within IpAPILevelAuthentication is based on) is primarily a one-way protocol which allows for mutual authentication.

Text should be added that states that re-authentication can be requested at any time, by either party, and does not have to be mutual authentication.

Lucent would also like to propose the removal of the prescribedMethod parameter from the authenticate() methods. We feel that this parameter is redundant, as each side is already aware of the prescribedMethod. In fact, the presence of this parameter can interfere with the authentication process, as the entity receiving the authenticate() request needs to check that the prescribedMethod passed as a parameter matches the method returned in the selectEncryptionMethod. If it does not match, then an exception has to be thrown.

Note: In removing this parameter, it was found that there is no use of P_INVALID_AUTH_CAPABILITY. Should this be removed?

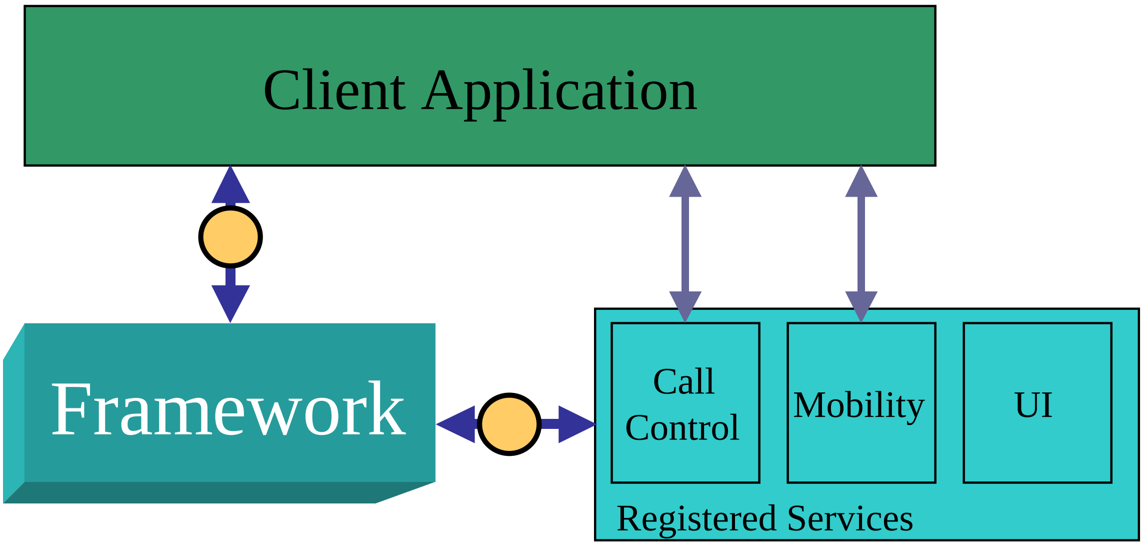
The framework overview section at the start of section 4 mentions that the application MUST authenticate the framework. Lucent feels that mutual authentication should not be enforced by the API, and that it should be the client application's responsibility to decide if it needs to authenticate the framework. As long as the framework has authenticated the application, and therefore trusts it, there seems to be no reason why the framework should deny an application’s requests just because it hasn’t yet tried to authenticate the framework.

Resulting changes

4. Overview of the Framework

This subclause explains which basic mechanisms are executed in the OSA Framework prior to offering and activating applications.

The Framework API contains interfaces between the Application Server and the Framework, and between Network Service Capability Server (SCS) and the Framework (these interfaces are represented by the yellow circles in the diagram below). The description of the Framework in this document separates the interfaces into these two distinct sets: Framework to Application interfaces and Framework to Service interfaces.



Some of the mechanisms are applied only once (e.g. establishment of service agreement), others are applied each time a user subscription is made to an application (e.g. enabling the call attempt event for a new user).

Basic mechanisms between Application and Framework:

- **Authentication:** Once an off-line service agreement exists, the application can access the authentication interface. The authentication model of OSA is a peer-to-peer model, but authentication does not have to be mutual. ~~The application must authenticate the framework and vice versa.~~ The application must be authenticated before it is allowed to use any other OSA interface. It is a policy decision for the application whether it must authenticate the framework or not. It is a policy decision for the framework whether it allows an application to authenticate it before it has completed its authentication of the application.

6.4.2 Initial Access

2: Select Encryption Method

The Application invokes selectEncryptionMethod on the Framework's API Level Authentication interface, identifying the ~~authentication~~ encryption methods it supports. The Framework prescribes the method to be used.

3: Authenticate

4: The application provides an indication if authentication succeeded.

5: The Application and Framework authenticate each other ~~using the prescribed method~~. The sequence diagram illustrates one of a series of one or more invocations of the authenticate method on the Framework's API Level Authentication interface. In each invocation, the Application supplies a challenge and the Framework returns the correct response. Alternatively or additionally the Framework may issue its own challenges to the Application using the authenticate method on the Application's API Level Authentication interface.

6.4.4 API Level Authentication

This sequence diagram illustrates the two-way mechanism by which the client application and the framework mutually authenticate one another.

The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. In some cases strong authentication may need to be enforced by the Framework to prevent misuse of resources. In addition it may be necessary to define the minimum encryption key length that can be used to ensure a high degree of confidentiality.

The application must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

1) The application calls initiateAuthentication on the OSA Framework Initial interface. This allows the application to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. The initiateAuthentication method can be used to specify the specific process, (e.g. CORBA security). OSA defines generic a authentication interface (API Level Authentication), which can be used to perform the authentication process. The initiateAuthentication method allows the application to pass a reference to its own authentication interface to the Framework, and receive a reference to the authentication interface preferred by the client, in return. In this case the API Level Authentication interface.

2) The application invokes the selectEncryptionMethod on the Framework's API Level Authentication interface. This includes the ~~authentication~~ encryption capabilities of the application. The framework then chooses an encryption~~authentication~~ method based on the encryption ~~authentication~~ capabilities of the application and the Framework. If the application is capable of handling more than one encryption~~authentication~~ method, then the Framework chooses one option, defined in the prescribedMethod parameter. In some instances, the encryption~~authentication~~ capability of the application may not fulfil the demands of the Framework, in which case, the authentication will fail.

3) The application and Framework interact to authenticate each other. For an authentication method of P_OSA_ACCESS, ~~Depending on the method prescribed,~~ this procedure ~~may~~ consists of a number of ~~messages e.g. a~~ challenge/ response ~~protocol~~exchanges. This authentication protocol is performed using the authenticate method on the API Level Authentication interface. ~~Depending on the authentication method selected, the protocol may require invocations on the API Level Authentication interface supported by the Framework; or on the application counterpart; or on both.~~P_OSA_ACCESS is based on CHAP, which is primarily a one-way protocol. Mutual authentication is achieved by the framework invoking the authenticate method on the application's APILevelAuthentication interface.

NOTE: At any point during the access session, either side can request re-authentication. Re-authentication does not have to be mutual.

8.1.1 Interface Class IpAppAPILevelAuthentication

Inherits from: IpInterface.

<<Interface>>

IpAppAPILevelAuthentication

authenticate (~~prescribedMethod : in TpAuthCapability,~~ challenge : in TpString, response : out TpStringRef) : TpResult

abortAuthentication () : TpResult

authenticationSucceeded () : TpResult

Method

authenticate()

This method is used by the framework to authenticate the client application. The challenge will be encrypted using the mechanism ~~indicated in prescribedMethod~~prescribed by selectEncryptionMethod. The client application must respond with the correct responses to the challenges presented by the framework. The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. ~~and the order of the exchanges is dependent on the prescribedMethod. (~~ ~~These~~ The invocation of this method may be interleaved with authenticate() calls by the client application on the IpAPILevelAuthentication interface.~~ This is defined by the prescribedMethod.)~~

Parameters

~~prescribedMethod : in TpAuthCapability~~

~~see selectEncryptionMethod() on the IpAPIlLevelAuthentication interface. This parameter contains the agreed method for authentication. If this is not the same value as returned by selectEncryptionMethod(), then an error code (P_INVALID_AUTH_CAPABILITY) is returned.~~

challenge : in TpString

The challenge presented by the framework to be responded to by the client application. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod.

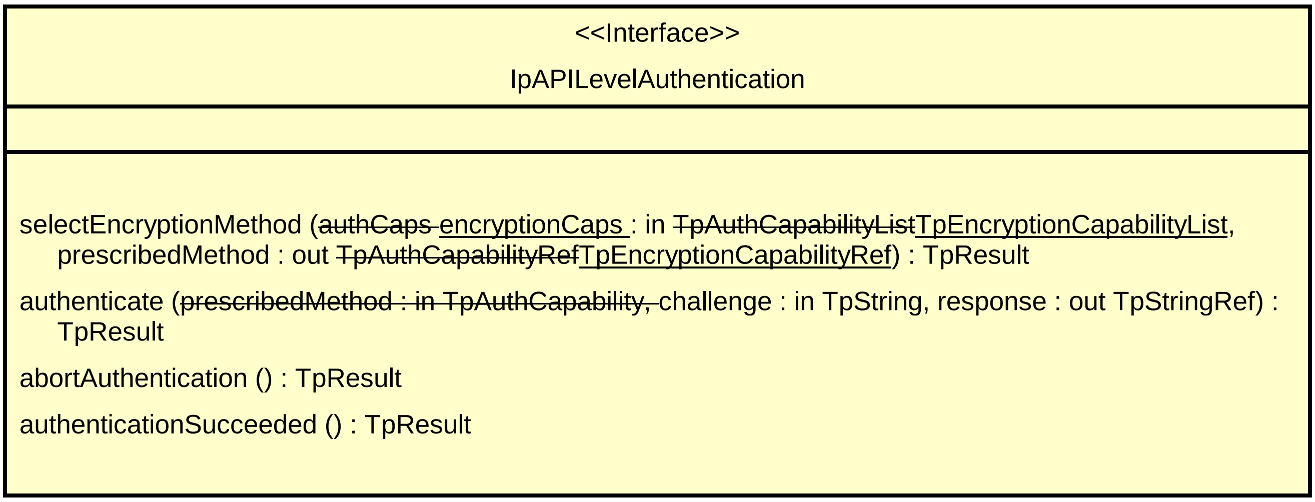
response : out TpStringRef

This is the response of the client application to the challenge of the framework in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

8.1.5 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

The API Level Authentication Framework interface is used by client application to perform its part of the mutual authentication process with the Framework necessary to be allowed to use any of the other interfaces supported by the Framework.



<<Interface>>

IpAPILevelAuthentication

selectEncryptionMethod (~~authCaps~~ encryptionCaps : in ~~TpAuthCapabilityList~~TpEncryptionCapabilityList, prescribedMethod : out ~~TpAuthCapabilityRef~~TpEncryptionCapabilityRef) : TpResult

authenticate (~~prescribedMethod : in TpAuthCapability,~~ challenge : in TpString, response : out TpStringRef) : TpResult

abortAuthentication () : TpResult

authenticationSucceeded () : TpResult

Method

selectEncryptionMethod()

The client application uses this method to initiate the authentication process. The framework returns its preferred mechanism. This should be within capability of the client application. If a mechanism that is acceptable to the framework within the capability of the client application cannot be found, the framework ~~returns an error code (~~ throws the P_NO_ACCEPTABLE_~~AUTH~~ENCRYPTION_CAPABILITY~~).~~ exception.   Once the framework has returned its preferred mechanism, it will wait for a predefined unit of time before invoking the client's authenticate() method (the wait is to ensure that the client can initialise any resources necessary to use the prescribed encryption method)

Parameters

encryption~~auth~~Caps : in ~~TpAuthCapabilityList~~TpEncryptionCapabilityList

This is the means by which the ~~authentication~~ encryption mechanisms supported by the client application are conveyed to the framework.

prescribedMethod : out TpEncryption~~Auth~~CapabilityRef

This is returned by the framework to indicate the mechanism preferred by the framework for the ~~authentication~~ encryption process. If the value of the prescribedMethod returned by the framework is not understood by the client application, it is considered a catastrophic error and the client application must abort.

Raises

TpCommonExceptions,_P_ACCESS_DENIED, P_NO_ACCEPTABLE_ENCRYPTION~~AUTH~~_CAPABILITY

Method

authenticate()

This method is used by the client application to authenticate the framework. The challenge will be encrypted using the mechanism ~~indicated in prescribedMethod~~prescribed by selectEncryptionMethod. The framework must respond with the correct responses to the challenges presented by the client application. The clientAppID received in the initiateAuthentication() can be used by the framework to reference the correct public key for the client application (the key management system is currently outside of the scope of the OSA APIs). The number of exchanges is dependent on the policies of each side. The whole authentication process is deemed successful when the authenticationSucceeded method is invoked. The invocation of this method may be interleaved with authenticate() calls by the framework on the client's APILevelAuthentication interface.
~~The number of exchanges and the order of the exchanges is dependent on the prescribedMethod.~~

Parameters

~~**prescribedMethod : in TpAuthCapability**~~

~~see selectEncryptionMethod(). This parameter contains the method that the framework has specified as acceptable for authentication. If this is not the same value as returned by selectEncryptionMethod(), then the framework returns an error code (P_INVALID_AUTH_CAPABILITY).~~

challenge : in TpString

The challenge presented by the client application to be responded to by the framework. The challenge mechanism used will be in accordance with the IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

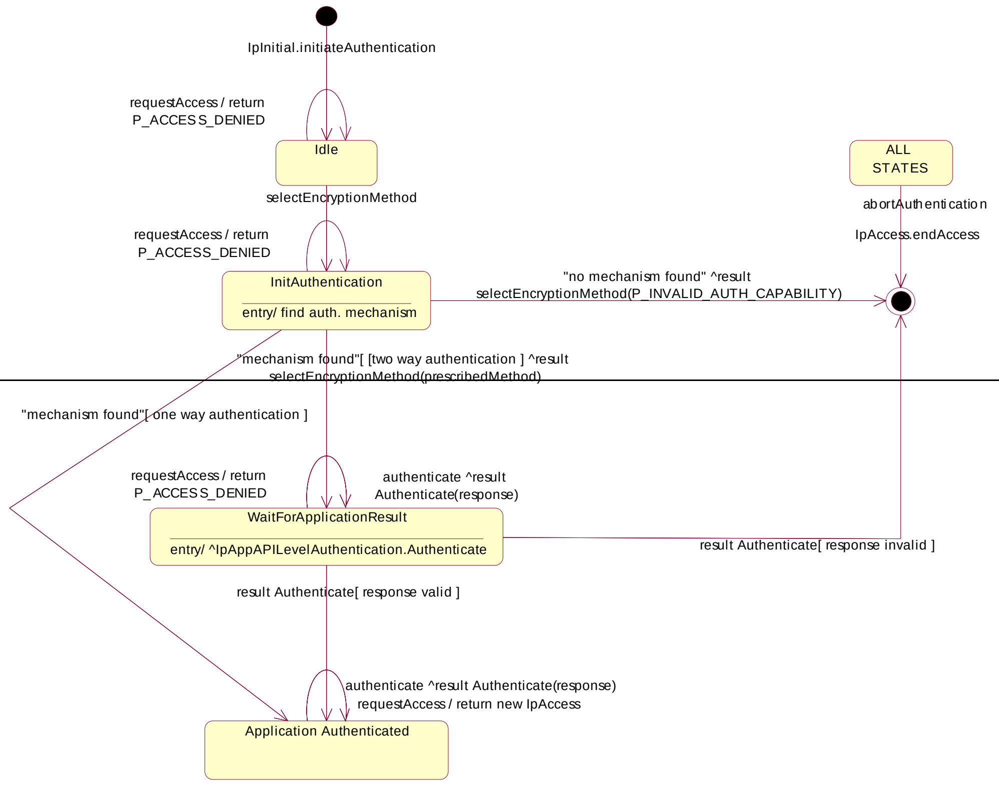
response : out TpStringRef

This is the response of the framework to the challenge of the client application in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectEncryptionMethod().

Raises

TpCommonExceptions,_P_ACCESS_DENIED~~, P_INVALID_AUTH_CAPABILITY~~

9.1.2 State Transition Diagrams for IpAPILevelAuthentication


IpInitial.initiateAuthentication
requestAccess / return P_ACCESS_DENIED
Idle
selectEncryptionMethod
ALL STATES
abortAuthentication
IpAccess.endAccess
requestAccess / return P_ACCESS_DENIED
InitAuthentication
entry/ find auth. mechanism
"no mechanism found" ^result selectEncryptionMethod(P_INVALID_AUTH_CAPABILITY)
"mechanism found"[[two way authentication] ^result selectEncryptionMethod(prescribedMethod)
"mechanism found"[one way authentication]
requestAccess / return P_ACCESS_DENIED
authenticate ^result Authenticate(response)
WaitForApplicationResult
entry/ ^IpAppAPILevelAuthentication.Authenticate
result Authenticate[response invalid]
result Authenticate[response valid]
authenticate ^result Authenticate(response)
requestAccess / return new IpAccess
Application Authenticated

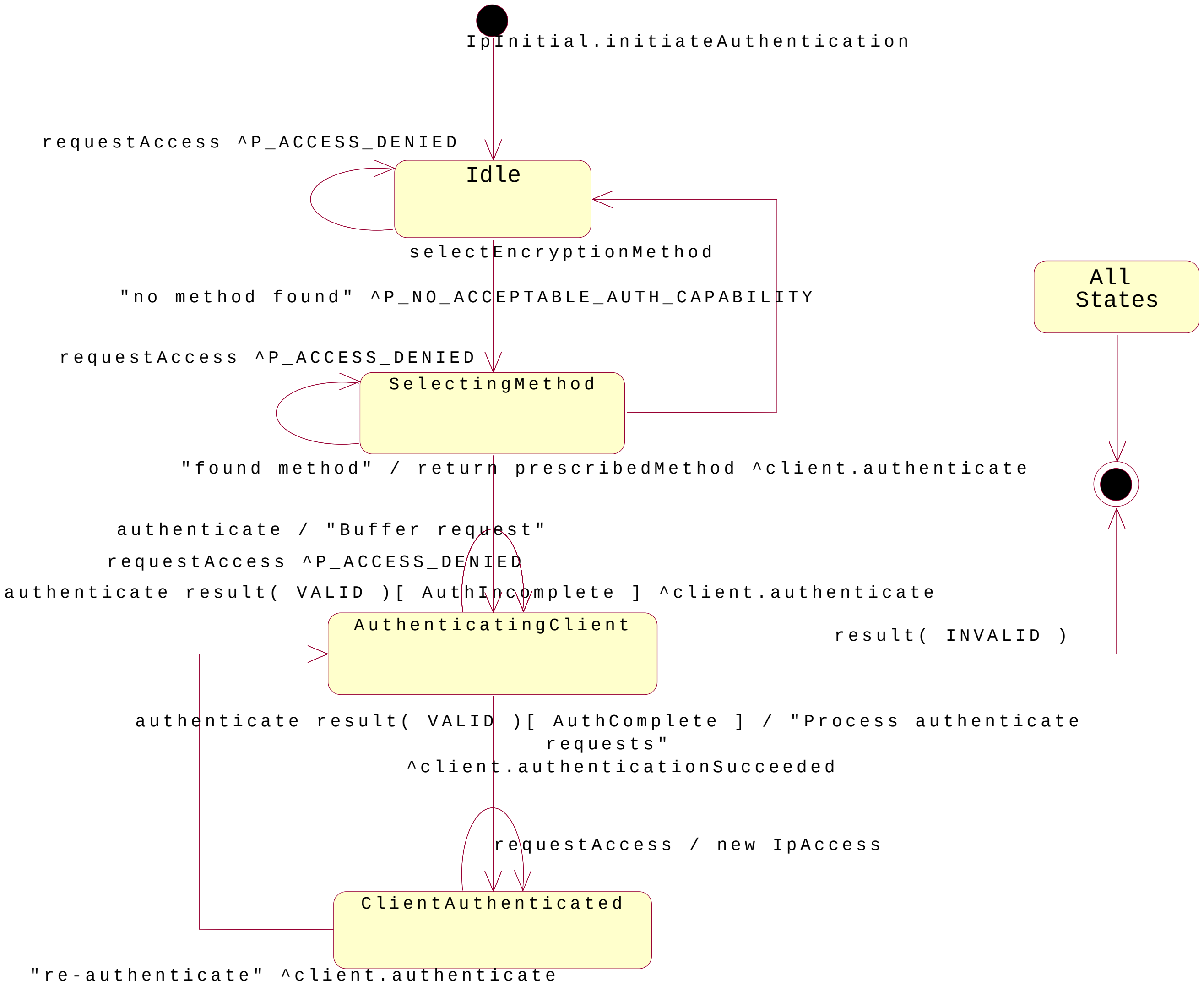

IpInitial.initiateAuthentication
requestAccess ^P_ACCESS_DENIED
Idle
selectEncryptionMethod
"no method found" ^P_NO_ACCEPTABLE_AUTH_CAPABILITY
All States
requestAccess ^P_ACCESS_DENIED
SelectingMethod
"found method" / return prescribedMethod ^client.authenticate
authenticate / "Buffer request"
requestAccess ^P_ACCESS_DENIED
authenticate result(VALID)[AuthIncomplete] ^client.authenticate
AuthenticatingClient
result(INVALID)
authenticate result(VALID)[AuthComplete] / "Process authenticate requests"
^client.authenticationSucceeded
requestAccess / new IpAccess
ClientAuthenticated
"re-authenticate" ^client.authenticate

Figure : State Transition Diagram for IpAPILevelAuthentication

9.1.2.1 Idle State

When the ~~application~~ client has ~~requested~~ invoked the IpInitial~~ interface for~~ initiateAuthentication method, an object implementing the IpAPILevelAuthentication interface is created. The ~~application~~ client now has to provide its ~~authentication~~ encryption capabilities by invoking ~~the~~ SelectEncryptionMethod~~ method~~.

9.1.2.2 ~~InitAuthentication~~ SelectingMethod State

In this state the Framework selects the preferred ~~authentication~~ encryption mechanism within the capability of the ~~application~~client. It is a policy of the framework (perhaps agreed off-line with the enterprise operator) whether the client has to be ~~When a proper mechanism is found, the Framework can decide that the application doesn't have to be authenticated (one way authentication) or that the application has to be authenticated~~authenticated or not. In case no mechanism can be found the ~~error code P_INVALID_AUTH~~P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY exception is ~~returned~~ thrown and the Authentication object ~~is destroyed~~moves back to the IDLE state~~. This implies that the application has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface~~ The client can now revisit its list of supported capabilities to identify whether it is complete. If it has no more encryption capabilities to use, then it must invoke abortAuthentication.~~.~~

9.1.2.3 ~~WaitForApplicationResult~~ AuthenticatingClient State

When entering this state, the Framework requests the ~~application~~ client to authenticate itself by invoking the Authenticate method on the ~~application~~client. In case the ~~application~~ client requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework ~~provides the correct response to the challenge of the application~~will either buffer the requests and respond when the client has been authenticated, or respond immediately, depending on policy. When the Framework has processed the ~~responds~~ response ~~to~~ from the Authenticate request on the client, the response is analysed. If ~~and in case~~ the response is valid but the authentication process is not yet complete, then another Authenticate request is sent to the client. If the response is valid and the authentication process has been completed, then a transition to the state ~~Application~~ ClientAuthenticated is made, the client is informed of its success by invoking authenticationSucceeded, then the framework begins to process any buffered authenticate requests. In case the response is not valid, the Authentication object is destroyed. This implicates that the ~~application~~ client has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface.

9.1.2.4 ~~Application~~ ClientAuthenticated State

In this state the ~~application~~ client is considered authenticated and is now allowed to request access to the IpAccess interface. In case the ~~application~~ client requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework provides the correct response to the challenge~~ of the application~~. If the framework decides to re-authenticate the client, then the authenticate request is sent to the client and a transition back to the AuthenticatingClient state occurs.

~~19.3.3 TpAuthCapability~~15.3.3 TpEncryptionCapability

This data type is identical to a TpString, and is defined as a string of characters that identify the ~~authentication~~ encryption capabilities that could be supported by the ~~OSA~~framework. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". Capabilities may be concatenated, using commas (,) as the separation character. The following values are defined .

String Value	Description
NULL	An empty (NULL) string indicates no client capabilities.
P_DES_56	A simple transfer of secret information that is shared between the client application and the framework with protection against interception on the link provided by the DES algorithm with a 56bit shared secret key
P_DES_128	A simple transfer of secret information that is shared between the client entity and the framework with protection against interception on the link provided by the DES algorithm with a 128bit shared secret key
P_RSA_512	A public-key cryptography system providing authentication without prior exchange of secrets using 512 bit keys
P_RSA_1024	A public-key cryptography system providing authentication without prior exchange of secrets using 1024bit keys

~~19.3.4 TpAuthCapabilityList~~ 15.3.4 TpEncryptionCapabilityList

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

16 Exception Classes

The following are the list of exception classes which are used in this interface of the API.

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_DUPLICATE_PROPERTY_NAME	A dupilcate property name has been received
P_ILLEGAL_SERVICE_ID	Illegal Service ID
P_ILLEGAL_SERVICE_TYPE	Illegal Service Type
P_INVALID_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_INVALID_ACTIVITY_TEST_ID	ID does not correspond to a valid activity test request
P_INVALID_AGREEMENT_TEXT	Invalid agreement text
P_INVALID_~~AUTH~~ENCRYPTION_CAPABILITY	Invalid ~~authentication~~ encryption capability
P_INVALID_AUTH_TYPE	Invalid type of authentication mechanism
P_INVALID_CLIENT_APP_ID	Invalid Client Application ID
P_INVALID_DOMAIN_ID	Invalid client ID
P_INVALID_ENT_OP_ID	Invalid Enterprise Operator ID

Name	Description
P_INVALID_PROPERTY	The framework does not recognise the property supplied by the client
P_INVALID_SAG_ID	Invalid Subscription Assignment Group ID
P_INVALID_SERVICE_CONTRACT_ID	Invalid Service Contract ID
P_INVALID_SERVICE_ID	Invalid service ID
P_INVALID_SERVICE_PROFILE_ID	Invalid service profile ID
P_INVALID_SERVICE_TOKEN	The service token has not been issued, or it has expired.
P_INVALID_SERVICE_TYPE	Invalid Service Type
P_INVALID_SIGNATURE	Invalid digital signature
P_INVALID_SIGNING_ALGORITHM	Invalid signing algorithm
P_MISSING_MANDATORY_PROPERTY	Mandatory Property Missing
P_NO_ACCEPTABLE_~~AUTH~~ENCRYPTION_CAPABILITY	An ~~authentication~~ encryption mechanism, which is acceptable to the framework, is not supported by the client
P_PROPERTY_TYPE_MISMATCH	Property Type Mismatch
P_SERVICE_ACCESS_DENIED	The client application is not allowed to access this service.
P_SERVICE_ACCESS_TYPE	The framework does not support the type of access interface requested by the client.
P_SERVICE_NOT_ENABLED	The service ID does not correspond to a service that has been enabled
P_UNKNOWN_SERVICE_ID	Unknown Sevice ID
P_UNKNOWN_SERVICE_TYPE	Unknown Service Type

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010714**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**019**	⌘	ev	**-**	⌘ Current version:	**4.1.0** ⌘

*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:	⌘	A client should only have one instance of a given service		
Source:	⌘	CN5		
Work item code:	⌘	OSA1	***Date:*** ⌘	30/08/2001
Category:	⌘	**F**	***Release:*** ⌘	REL-4

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 specification currently does not state that an application can not have more than one instance of the same service running at the same time. This can bring about a problem when the framework attempts to communicate with the application. For example, method IpAppFaultManager.svcUnavailableInd() takes as its argument a serviceID. When the Framework invokes this method, there is no way to determine which instance of that ID the invocation is meant to deal with.
Summary of change:	⌘	Some text should be placed in the description of selectService, which states that an exception will be thrown if the client application attempts to select a service when it already has a live instance of that service.
Consequences if not approved:	⌘	The specification will be ambiguous and incorrect with regards to the existence of a 1-2-1 relationship between an application and a service instance. The TS will be ambiguous and difficult to implement correctly – interworking will be jeopardised. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected:	⌘	8.1.6		
Other specs affected:	⌘	□ Other core specifications □ Test specifications □ O&M Specifications	⌘	
Other comments:	⌘			

Problem

In the IpAppFaultManager interface, there is a svcUnavailableInd() method. This method is invoked when the framework needs to inform the client that it can no longer use a service (perhaps due to a heartbeat failure). The parameter to this method is a single serviceID.

However, there is currently nothing in the specification that states that an application cannot have more than one instance of the same service. If the application has two instances of call control, and it receives this svcUnavailableInd telling it that its call control is going down, how does it know which instance is no longer available?

Its perfectly conceivable that the other call control instance is still running perfectly, so Lucent doesn't believe that this method can be said to apply to ALL instances of the service that were allocated to this application. To do so would be dictating the implementation style of the service and its factory, which is not within the remit of the framework.

As a more general note, Lucent cannot see the value for an application to obtain two identical instances of the same service. One argument is that the application might wish to have two instances of the same service so that it can perform load balancing between them. However, load balancing, in this case, should be performed by the service instance transparently to the application.

Proposal

Lucent believes that some text should be placed in the description of selectService which states that an exception will be thrown if the client application attempts to select a service when it already has a live instance of that service. This would prevent the application from having two instances of the same service and would address the problem above.

Resulting changes

8.1.6 Interface Class IpAccess

Method

selectService()

This method is used by the client application to identify the service that the client application wishes to use. If the client application is not allowed to access the service, then ~~an error code (~~ the P_SERVICE_ACCESS_DENIED~~) is returned~~ exception is thrown. The P_SERVICE_ACCESS_DENIED exception is also thrown if the client attempts to select a service for which it has already signed a service agreement for, and therefore obtained an instance of.

Parameters

serviceID : in TpServiceID

This identifies the service required. If the serviceID is not recognised by the framework, an error code (P_INVALID_SERVICE_ID) is returned.

serviceToken : out TpServiceTokenRef

This is a free format text token returned by the framework, which can be signed as part of a service agreement. This will contain operator specific information relating to the service level agreement. The serviceToken has a limited lifetime. If the lifetime of the serviceToken expires, a method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will automatically expire if the client application or framework invokes the endAccess method on the other's corresponding access interface.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_SERVICE_ACCESS_DENIED

3GPP TSG_CN5 (Open Service Access – OSA) **N5-010715**
Meeting #12, Sophia Antipolis, FRANCE, 16 – 19 July 2001

CR-Form-v4

CHANGE REQUEST

⌘ **29.198-03 CR 020** ⌘ ev **-** ⌘ Current version: **4.1.0** ⌘

*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: ⌘	Some methods on the IpApp interfaces should throw exceptions		
Source: ⌘	CN5		
Work item code: ⌘	OSA1	***Date:*** ⌘	30/08/2001
Category: ⌘	**F**	***Release:*** ⌘	REL-4
	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: ⌘	A decision was taken by CN5 that the methods on IpApp* interfaces should not be able to throw exceptions. For the framework, the situation is different from most as there is a dialogue that occurs between the framework and its user. This dialogue can include mutual authentication and the signing of an agreement to use a service. Client applications are not able to throw exceptions in response to errors occurring during a dialogue between themselves and the Framework.
Summary of change: ⌘	CN5 proposes that methods that form a part of the dialogue referenced above, methods that are not called in direct response to a method invocation on the framework, form a separate group of methods that should be allowed to throw exceptions.
Consequences if not approved: ⌘	Client applications are not able to throw exceptions in response to errors occurring during a dialogue between themselves and the Framework. Failure to adopt this CR would result in divergence between the 3GPP R4 specification and the ETSI/Parlay specifications.

Clauses affected: ⌘	8.1.2	
Other specs affected: ⌘	□ Other core specifications □ Test specifications □ O&M Specifications	⌘
Other comments: ⌘		

Problem

A decision was taken during the recent restructuring of the exception handling in 29.198 that the methods on the IpApp* interfaces should not be capable of throwing exceptions. However, for the framework, the situation is different from most. For the framework, there is a dialogue that occurs between the framework and its user. This dialogue can include mutual authentication and the signing of an agreement to use a service.

Proposal

Lucent feels that methods that form a part of the dialogue referenced above, methods that are not called in direct response to a method invocation on the framework, form a separate group of methods that should be allowed to throw exceptions.

Lucent proposes that this group of methods consists of:
IpAppAccess.signServiceAgreement;
IpAppAccess.terminateServiceAgreement;
IpAppAccess.terminateAccess.

For example, if the FW passes in invalid information (signing algorithm, service token or agreement text) to the IpAppAccess.signServiceAgreement() method, then the method should be able to indicate to the server side that it cannot do anything. We feel that simply returning a null digital signature is not really an acceptable mode of indicating the failure.

Note: If the IpApp and IpSvc interfaces are aligned as per contribution N5-010445, then some of this work might already be done as methods on the IpSvc interfaces are allowed to throw exceptions.

Resulting changes

Note: The method description for IpAppOAM.systemDataAndTimeQuery still states that the method can throw a P_INVALID_DATE_TIME_FORMAT exception. It should be considered by the editor of 29.198-3 whether this method is ACTUALLY capable of throwing this exception

The following method descriptions should be updated to specify the exceptions that may be thrown. These exceptions are the same as those for the corresponding methods on the framework interfaces with the sole difference being that no ACCESS_DENIED exceptions may be thrown by the IpApp* interfaces.

The description of the IpAppAccess interface has been updated as the previous description did not appear to be complete.

8.1.2 Interface Class IpAppAccess

Inherits from: IpInterface.

IpAppAccess interface is offered by the client application to the framework to allow it to initiate interactions during the access session. ~~The Access client application interface is used by the Framework to perform the steps that are necessary in order to allow it to service access.~~

<<Interface>>

IpAppAccess

signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : out TpStringRef) : TpResult

terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpString) : TpResult

terminateAccess (terminationText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : in TpString) : TpResult

Method

signServiceAgreement()

This method is used by the framework to request that the client application sign an agreement on the service. It is called in response to the client application calling the selectService() method on the IpAccess interface of the framework. The framework provides the service agreement text for the client application to sign. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application agrees, it signs the service agreement, returning its digital signature to the framework.

Parameters

serviceToken : in TpServiceToken

This is the token returned by the framework in a call to the selectService() method. This token is used to identify the service instance to which this service agreement corresponds. (If the client application selects many services, it can determine which selected service corresponds to the service agreement by matching the service token.) If the serviceToken is invalid, or not known by the client application, then ~~an error code (~~the P_INVALID_SERVICE_TOKEN exception is thrown~~) is returned~~.

agreementText : in TpString

This is the agreement text that is to be signed by the client application using the private key of the client application. If the agreementText is invalid, then ~~an error code (~~the P_INVALID_AGREEMENT_TEXT exception is thrown~~) is returned~~.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the client application, ~~an error code (~~the P_INVALID_SIGNING_ALGORITHM~~) is returned~~ exception is thrown.

digitalSignature : out TpStringRef

The digitalSignature is the signed version of a hash of the service token and agreement text given by the framework.

Raises

TpCommonExceptions, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM

Method

terminateServiceAgreement()

This method is used by the framework to terminate an agreement for the service.

Parameters

serviceToken : in TpServiceToken

This is the token passed back from the framework in a previous selectService() method call. This token is used to identify the service agreement to be terminated. If the serviceToken is invalid, or unknown to the client application, ~~an error code (~~the P_INVALID_SERVICE_TOKEN~~) is returned~~ exception will be thrown.

terminationText : in TpString

This is the termination text that describes the reason for the termination of the service agreement.

digitalSignature : in TpString

This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the service agreement is terminated, otherwise ~~an error code~~the ~~(~~P_INVALID_SIGNATURE~~) is returned~~ exception is thrown.

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

Method

terminateAccess()

The terminateAccess operation is used by the framework to end the client application's access session.
After terminateAccess() is invoked, the client application will no longer be authenticated with the framework. The client application will not be able to use the references to any of the framework interfaces gained during the access session. Any calls to these interfaces will fail. If at any point the framework's level of confidence in the identity of the client becomes too low, perhaps due to re-authentication failing, the framework should terminate all outstanding service agreements for that client application, and should take steps to terminate the client application's access session WITHOUT invoking terminateAccess() on the client application. This follows a generally accepted security model where the framework has decided that it can no longer trust the application and will therefore sever ALL contact with it.

Parameters

terminationText : in TpString

This is the termination text describes the reason for the termination of the access session.

signingAlgorithm : in TpSigningAlgorithm

This is the algorithm used to compute the digital signature. If the signingAlgorithm is invalid, or unknown to the client application, ~~an error code (~~the P_INVALID_SIGNING_ALGORITHM~~) is returned~~ exception will be thrown.

digitalSignature : in TpString

This is a signed version of a hash of the termination text. The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the access session is terminated, otherwise ~~an error code (~~the P_INVALID_SIGNATURE~~) is returned~~ exception is thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNING_ALGORITHM, P_INVALID_SIGNATURE