3GPP TSG CN Plenary Meeting #19 NP-030019
12- 14 March 2003, Birmingham, UK

Source: CN5 (OSA)

Title: Rel-4 CRs 29.198-03 OSA API Part 3: Framework

Agenda item: 7.10

Document for: APPROVAL

Doc-1st-Level	Spec	CR	Rev	Phase	Subject	Cat	Version-Current	Doc-2nd-Level	Workitem
NP-030019	29.198-03	060	-	Rel-4	Correction of status of methods to interfaces in clause 7.3	F	4.6.0	N5-020895	OSA1
NP-030019	29.198-03	061	-	Rel-4	Correction of status of methods to interfaces in clause 8.3	F	4.6.0	N5-020896	OSA1
NP-030019	29.198-03	062	-	Rel-4	Correction to Initial Access Sequence Diagram	F	4.6.0	N5-021070	OSA1
NP-030019	29.198-03	063	-	Rel-5	Correction to Initial Access Sequence Diagram	A	5.1.0	N5-021071	OSA2
NP-030019	29.198-03	064	-	Rel-4	Enable creation/destruction of load level notifications at the request of Framework	F	4.6.0	N5-021131	OSA1
NP-030019	29.198-03	065	-	Rel-5	Enable creation/destruction of load level notifications at the request of Framework	A	5.1.0	N5-021132	OSA2
NP-030019	29.198-03	066	-	Rel-4	Correction of Sequence for Framework – Service load management	F	4.6.0	N5-021086	OSA1
NP-030019	29.198-03	067	-	Rel-5	Correction of Sequence for Framework – Service load management	A	5.1.0	N5-021087	OSA2
NP-030019	29.198-03	072	-	Rel-4	Correction of status of methods to interfaces in clause 6.3	F	4.6.0	N5-020894	OSA1
NP-030019	29.198-03	073	-	Rel-4	Add Initial Load Notification report for Framework Integrity Management Load Notification model	F	4.6.0	N5-030075	OSA1
NP-030019	29.198-03	074	-	Rel-5	Add Initial Load Notification report for Framework Integrity Management Load Notification model	A	5.1.0	N5-030076	OSA2

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020895**
Meeting #20, Miami/ FL, USA, 23 – 27 September 2002

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-03 CR 060** ⌘ **rev** **-** ⌘ Current version: **4.6.0** ⌘

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

Proposed change affects:	UICC apps⌘	☐	ME	☐	Radio Access Network	☐	Core Network	X

Title:	⌘	Correction of status of methods to interfaces in clause 7.3		
Source:	⌘	N5		
Work item code:	⌘	OSA1	***Date:*** ⌘	27/09/2002
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)* *Rel-6 (Release 6)*

Reason for change:	⌘	There is no requirement in the standard about the necessity to implement all or only some of the methods defined for an interface.
Summary of change:	⌘	Clarify which methods are mandatory and which are optional.
Consequences if not approved:	⌘	Application developers will not know which methods will actually be available.

Clauses affected:	⌘	7.3 Interface Classes				
		Y	N			
Other specs affected:	⌘		X	Other core specifications	⌘	
			X	Test specifications		
			X	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/specs/CR.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.

7.3 Interface Classes

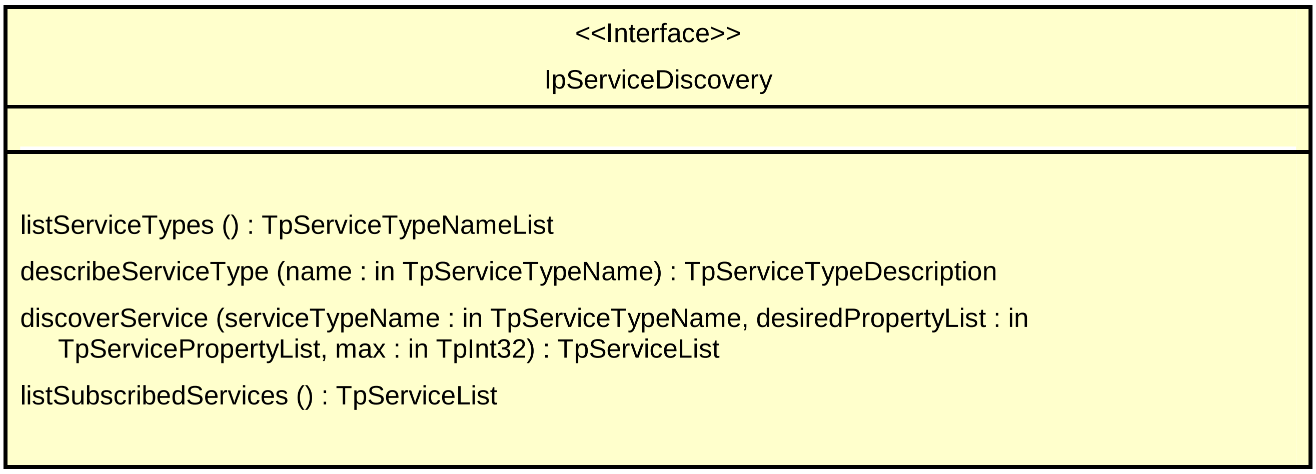
7.3.1 Service Discovery Interface Classes

7.3.1.1 Interface Class IpServiceDiscovery

Inherits from: IpInterface.

The service discovery interface, shown below, consists of four methods. Before a service can be discovered, the enterprise operator (or the client applications) must know what "types" of services are supported by the Framework 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()" 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 enterprise operator (or the client applications) can discover a specific set of registered services that both belong to a given type and possess the desired "property values", by using the "discoverService() method. Once the enterprise operator finds out the desired set of services supported by the framework, it subscribes to (a sub-set of) these services using the Subscription Interfaces. The enterprise operator (or the client applications in its domain) can find out the set of services available to it (i.e., the service that it can use) by invoking "listSubscribedServices()". The service discovery APIs are invoked by the enterprise operators or client applications. They are described below.

This interface shall be implemented by a Framework with as a minimum requirement the listServiceTypes(), describeServiceType() and discoverService() methods.


<<Interface>>

IpServiceDiscovery

listServiceTypes () : TpServiceTypeNameList

describeServiceType (name : in TpServiceTypeName) : TpServiceTypeDescription

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

listSubscribedServices () : 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,P_ACCESS_DENIED`

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 available or unavailable.

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_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE`

Method
`discoverService()`

The discoverService operation is the means by which a client application is able to obtain the service IDs of the services that meet its requirements. The client application 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 client application 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 client application provided. The service properties returned will form a complete view of what the client application will be able to do with the service, as per the service level agreement. If the framework supports 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.

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

Parameters

`serviceTypeName : in TpServiceTypeName`

The "serviceTypeName" parameter conveys the required service type. It is key to the central purpose of "service trading". It is the basis for type safe interactions between the service exporters (via registerService) and service importers (via discoverService). By stating a service type, the importer implies the service type and a domain of discourse for talking about properties of service.

· 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 properties {name and value list} that the discovered set of 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_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE,P_INVALID_PROPERTY`

Method

`listSubscribedServices()`

Returns a list of services so far subscribed by the enterprise operator. The enterprise operator (or the client applications in the enterprise domain) can obtain a list of subscribed services that they are allowed to access.

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

Parameters

No Parameters were identified for this method

Returns

`TpServiceList`

Raises

`TpCommonExceptions,P_ACCESS_DENIED`

7.3.2 Service Agreement Management Interface Classes

7.3.2.1 Interface Class IpAppServiceAgreementManagement

Inherits from: IpInterface.

This interface and the signServiceAgreement() and terminateServiceAgreement() methods shall be implemented by an application.

<<Interface>> IpAppServiceAgreementManagement
signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm) : TpOctetSet terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpOctetSet) : void

Method
signServiceAgreement()

Upon receipt of the initiateSignServiceAgrement() method from the client application, this method is used by the framework to request that the client application sign an agreement on the service. 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.

Returns <digitalSignature> : The digitalSignature is the signed version of a hash of the service token and agreement text given by the framework. If the signature is incorrect the serviceToken will be expired immediately.

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 the P_INVALID_SERVICE_TOKEN exception is thrown.

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 the P_INVALID_AGREEMENT_TEXT exception is thrown.

signingAlgorithm : in TpSigningAlgorithm

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

Returns

`TpOctetSet`

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, the P_INVALID_SERVICE_TOKEN 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 TpOctetSet`

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 the P_INVALID_SIGNATURE exception will be thrown.

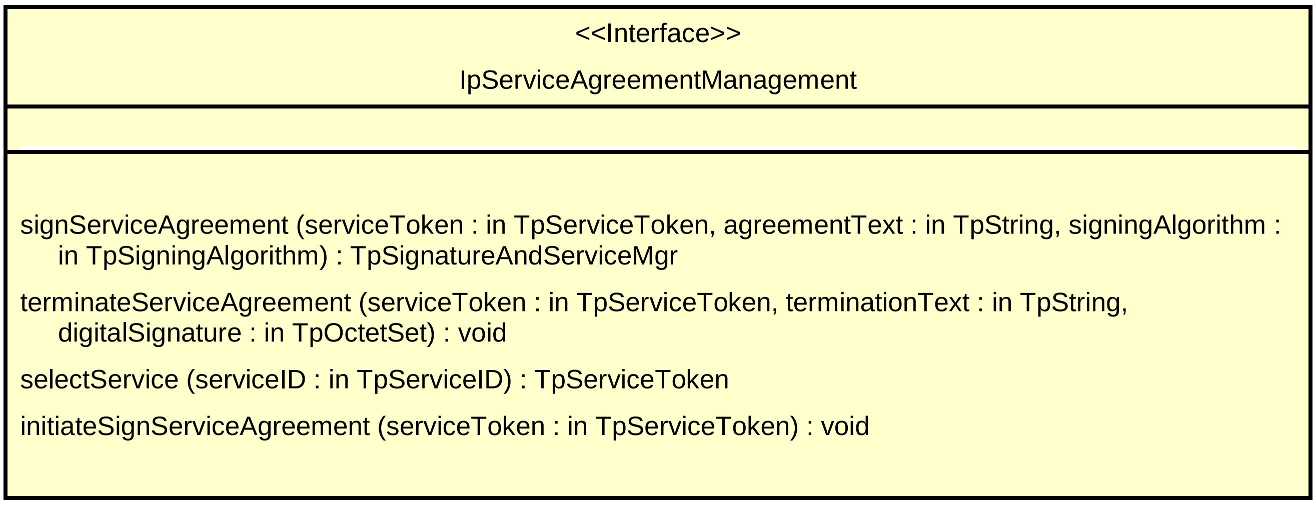
Raises

`TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE`

7.3.2.2 Interface Class IpServiceAgreementManagement

Inherits from: IpInterface.

This interface and ~~all its~~the signServiceAgreement(), terminateServiceAgreement(), selectService() and initiateSignServiceAgreement() methods shall be implemented by a Framework.



<<Interface>>

IpServiceAgreementManagement

signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm) : TpSignatureAndServiceMgr

terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpOctetSet) : void

selectService (serviceID : in TpServiceID) : TpServiceToken

initiateSignServiceAgreement (serviceToken : in TpServiceToken) : void

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.

Returns <signatureAndServiceMgr> : 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:  TpOctetSet;
                    serviceMgrInterface:  IpServiceRef;
                };
```

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.

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.

Returns

`TpSignatureAndServiceMgr`

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 that describes the reason for the termination of the service agreement.

`digitalSignature : in TpOctetSet`

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
`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 the P_SERVICE_ACCESS_DENIED 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. This is because there must be only one service instance per client application.

Returns <serviceToken> : 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.

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.

Returns

`TpServiceToken`

Raises

`TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_SERVICE_ACCESS_DENIED`

Method
`initiateSignServiceAgreement()`

This method is used by the client application to initiate the sign service agreement process. If the client application is not allowed to initiate the sign service agreement process, the exception (P_SERVICE_ACCESS_DENIED) is thrown.

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, the exception (P_INVALID_SERVICE_TOKEN) is thrown.

Raises

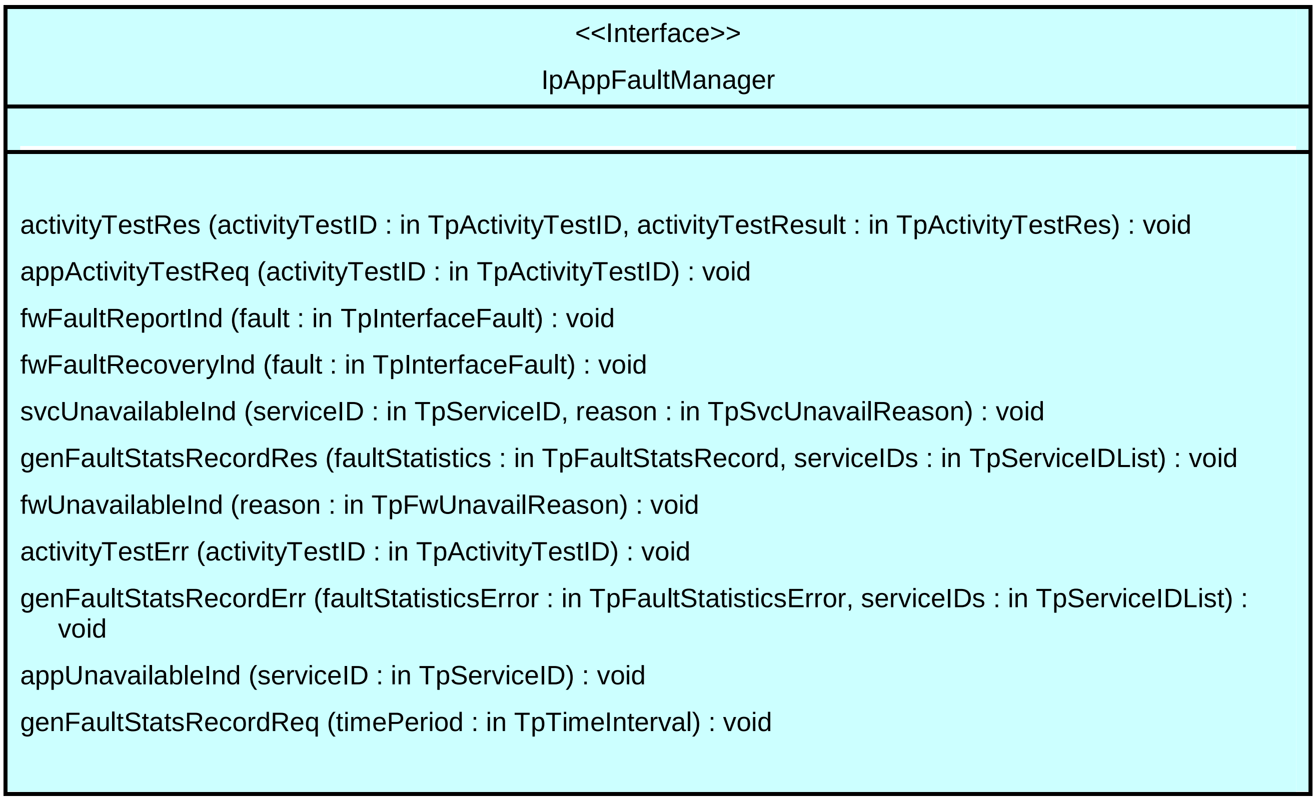
`TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_SERVICE_ACCESS_DENIED`

7.3.3 Integrity Management Interface Classes

7.3.3.1 Interface Class IpAppFaultManager

Inherits from: IpInterface.

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



<<Interface>>

IpAppFaultManager

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

appActivityTestReq (activityTestID : in TpActivityTestID) : void

fwFaultReportInd (fault : in TpInterfaceFault) : void

fwFaultRecoveryInd (fault : in TpInterfaceFault) : void

svcUnavailableInd (serviceID : in TpServiceID, reason : in TpSvcUnavailReason) : void

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

fwUnavailableInd (reason : in TpFwUnavailReason) : void

activityTestErr (activityTestID : in TpActivityTestID) : void

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

appUnavailableInd (serviceID : in TpServiceID) : void

genFaultStatsRecordReq (timePeriod : in TpTimeInterval) : void

Method
activityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Method
appActivityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

Method
`fwFaultReportInd()`

The framework invokes this method to notify the client application of a failure within the framework. The client application 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.

Method
`fwFaultRecoveryInd()`

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

Parameters

`fault : in TpInterfaceFault`

Specifies the fault from which the framework has recovered.

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).

Parameters

`serviceID : in TpServiceID`

Identifies the affected service.

`reason : in TpSvcUnavailReason`

Identifies the reason why the service is no longer available

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 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.

Method
`fwUnavailableInd()`

The framework invokes this method to inform the client application that it is no longer available.

Parameters

`reason : in TpFwUnavailReason`

Identifies the reason why the framework is no longer available

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 TpFaultStatisticsError`

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.

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

`serviceID : in TpServiceID`

Specifies the service for which the indication of unavailability was received.

Method
`genFaultStatsRecordReq()`

This method is used by the framework to solicit fault statistics from the client application, for example when the framework was asked for these statistics by a service instance by using the genFaultStatsRecordReq operation on the IpFwFaultManager interface. On receipt of this request, the client application must produce a fault statistics record, for the application during the specified time interval, which is returned to the framework using the genFaultStatsRecordRes operation on the IpFaultManager interface.

Parameters

`timePeriod : in TpTimeInterval`

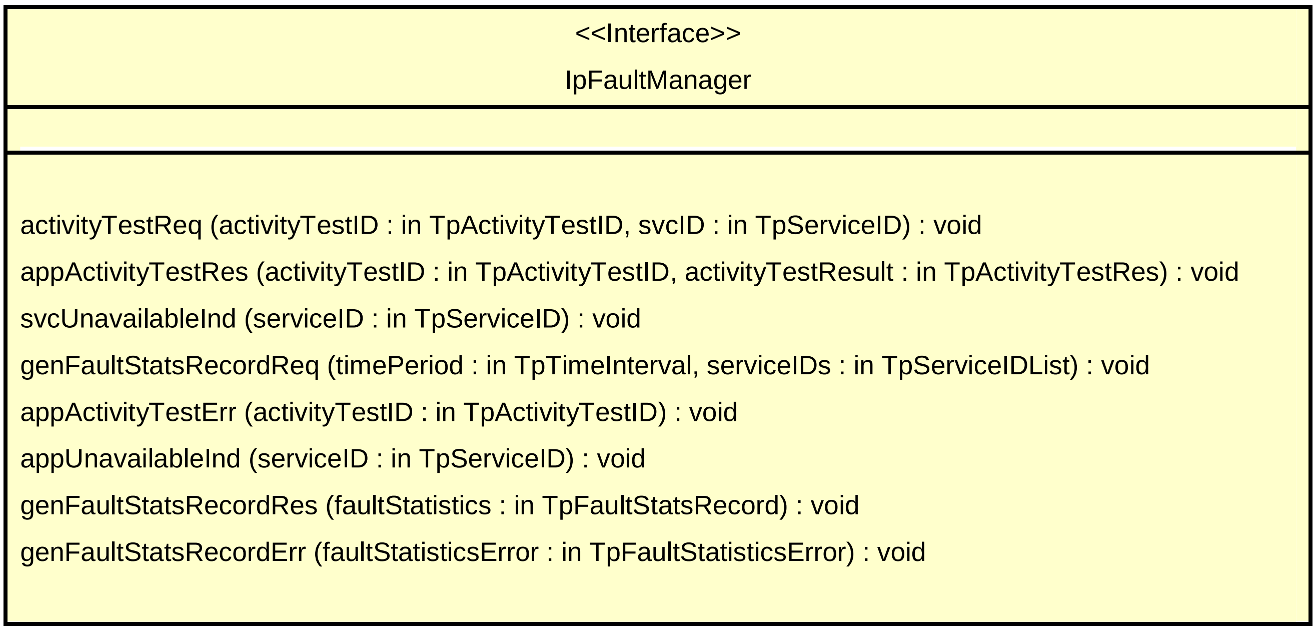
The period over which the fault statistics are to be generated. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the client application.

7.3.3.2 Interface Class IpFaultManager

Inherits from: IpInterface.

This interface is used by the application to inform the framework of events that affect the integrity of the framework and services, and to request information about the integrity of the system. The fault manager operations do not exchange callback interfaces as it is assumed that the client application supplies its Fault Management callback interface at the time it obtains the Framework's Fault Management interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.

If the IpFaultManager interface is implemented by a Framework, at least one of these methods shall be implemented. If the Framework is capable of invoking the IpAppFaultManager.appActivityTestReq() method, it shall implement appActivityTestRes() and appActivityTestErr() in this interface. If the Framework is capable of invoking IpAppFaultManager.genFaultStatsRecordReq(), it shall implement genFaultStatsRecordRes() and genFaultStatsRecordErr() in this interface.



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

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

svcUnavailableInd (serviceID : in TpServiceID) : void

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

appActivityTestErr (activityTestID : in TpActivityTestID) : void

appUnavailableInd (serviceID : in TpServiceID) : void

genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord) : void

genFaultStatsRecordErr (faultStatisticsError : in TpFaultStatisticsError) : void

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. 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.

For security reasons the client application has access to the service ID rather than the service instance ID. However, as there is a one to one relationship between the client application and a service, i.e. there is only one service instance of the specified service per client application, it is the obligation of the framework to determine the service instance ID from the service ID.

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 an empty string.

Raises

`TpCommonExceptions,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE`

Method

`appActivityTestRes()`

The client application 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_SERVICE_ID,P_INVALID_ACTIVITY_TEST_ID`

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. 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 either the framework 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. 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. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the framework.

`serviceIDs : in TpServiceIDList`

Specifies either the framework or services to be included in the general fault statistics record. 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, P_UNAUTHORISED_PARAMETER_VALUE`

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`

Method
`appUnavailableInd()`

This method is used by the application to inform the framework that it is ceasing its use of the service instance. This may 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

Method
genFaultStatsRecordRes()

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

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

Raises

TpCommonExceptions

Method
genFaultStatsRecordErr()

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

Parameters

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

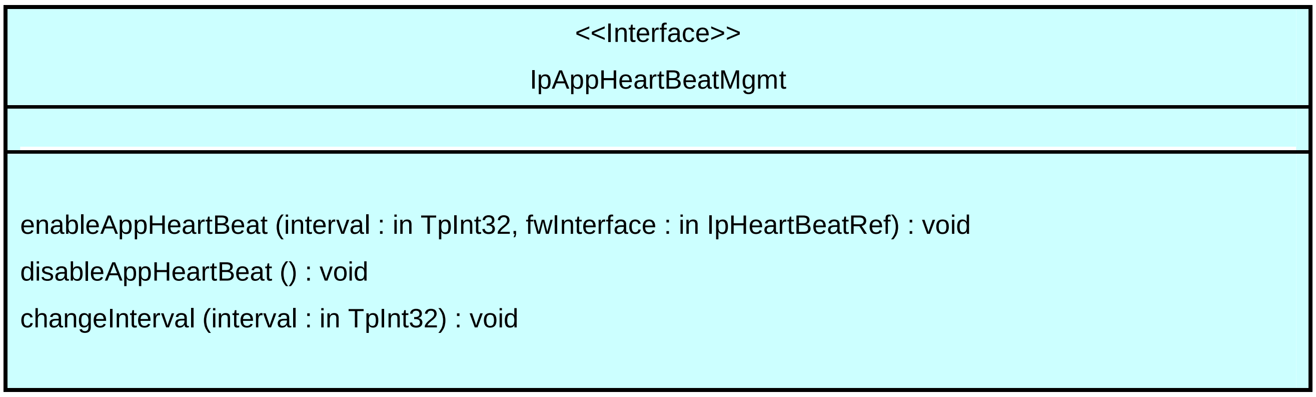
Raises

TpCommonExceptions

7.3.3.3 Interface Class IpAppHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the client application by the framework.



<<Interface>>

IpAppHeartBeatMgmt

enableAppHeartBeat (interval : in TpInt32, fwInterface : in IpHeartBeatRef) : void

disableAppHeartBeat () : void

changeInterval (interval : in TpInt32) : void

Method
`enableAppHeartBeat()`

With this method, the framework instructs the client application 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 IpHeartBeatRef`

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

Method
`disableAppHeartBeat()`

Instructs the client application to cease the sending of its heartbeat.

Parameters
No Parameters were identified for this method

Method
`changeInterval()`

Allows the administrative change of the heartbeat interval.

Parameters

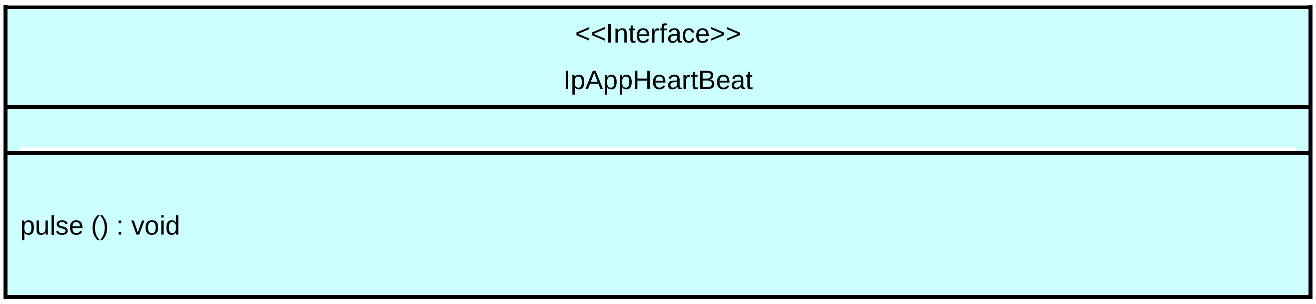
`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

7.3.3.4 Interface Class IpAppHeartBeat

Inherits from: IpInterface.

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



Method
`pulse()`

The framework uses 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.

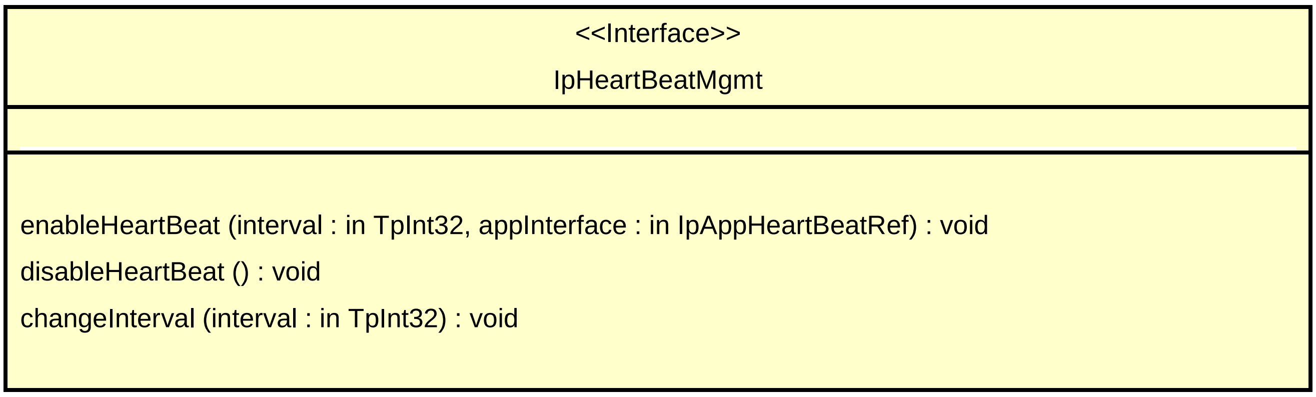
Parameters
No Parameters were identified for this method

7.3.3.5 Interface Class IpHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the framework by a client application.

If the IpHeartBeatMgmt interface is implemented by a Framework, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.



<<Interface>>
IpHeartBeatMgmt
enableHeartBeat (interval : in TpInt32, appInterface : in IpAppHeartBeatRef) : void
disableHeartBeat () : void
changeInterval (interval : in TpInt32) : void

Method
`enableHeartBeat()`

With this method, the client application instructs the framework 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.

`appInterface : in IpAppHeartBeatRef`
This parameter refers to the callback interface the heartbeat is calling.

Raises

`TpCommonExceptions`

Method
`disableHeartBeat()`

Instructs the framework to cease the sending of its heartbeat.

Parameters
No Parameters were identified for this method

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

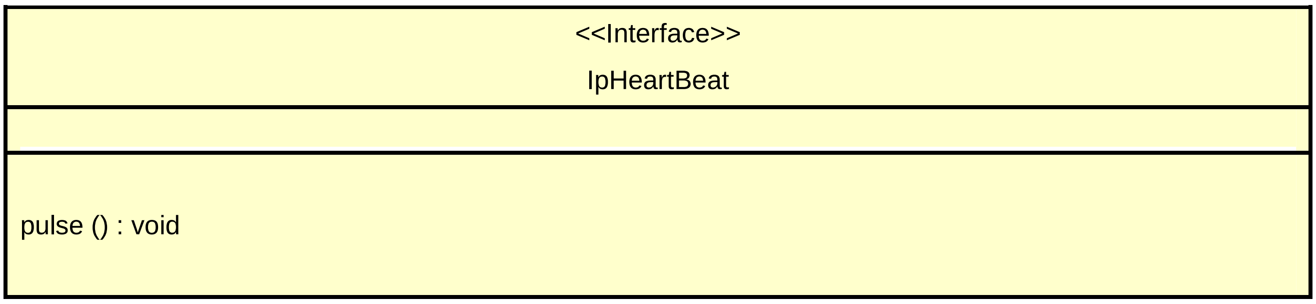
`TpCommonExceptions`

7.3.3.6 Interface Class IpHeartBeat

Inherits from: IpInterface.

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

If a Framework is capable of invoking IpAppHeartBeatMgmt.enableHeartBeat(), it shall implement IpHeartBeat and the pulse() method.



Method
`pulse()`

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

No Parameters were identified for this method

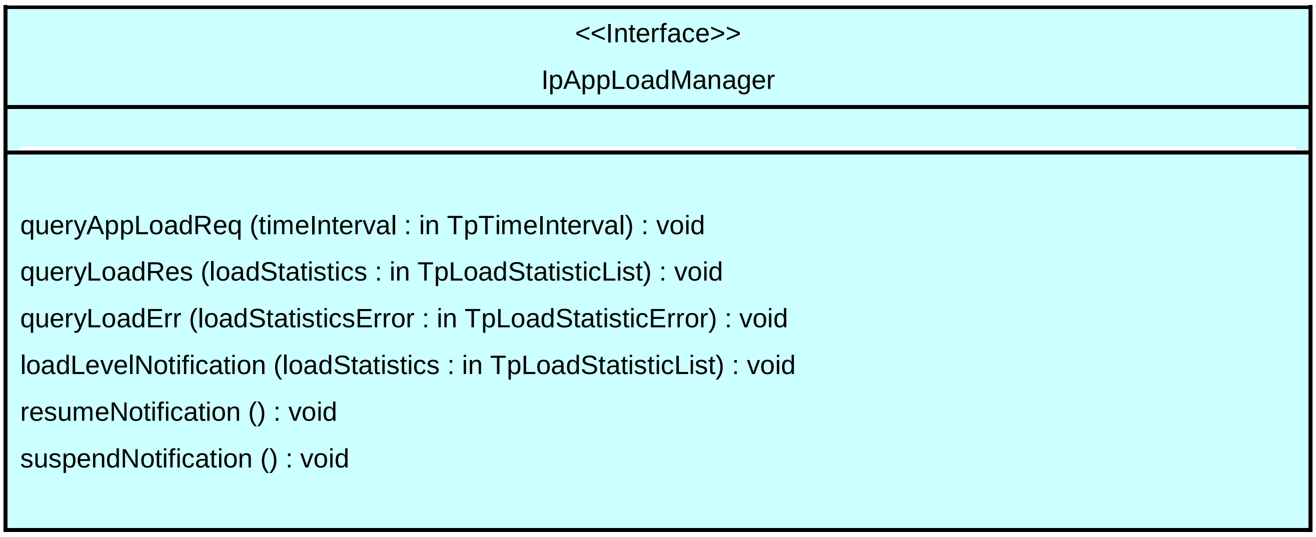
Raises

`TpCommonExceptions`

7.3.3.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

The client application developer supplies the load manager application interface to handle requests, reports and other responses from the framework load manager function. 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() method on the IpAccess interface.


<<Interface>>
IpAppLoadManager
queryAppLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
resumeNotification () : void
suspendNotification () : void

Method

`queryAppLoadReq()`

The framework uses this method to request the application to provide load statistics records for the application.

Parameters

`timeInterval : in TpTimeInterval`

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

Method

`queryLoadRes()`

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

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the framework-supplied load statistics

Method
`queryLoadErr()`

The framework uses this method to return an error response to the application 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 IpLoadManager interface.

Parameters

`loadStatisticsError : in TpLoadStatisticError`

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

Method
`loadLevelNotification()`

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

Parameters

`loadStatistics : in TpLoadStatisticList`

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

Method
`resumeNotification()`

The framework uses this method to request the application 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

Method
`suspendNotification()`

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

Parameters
No Parameters were identified for this method

7.3.3.8 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.

If the IpLoadManager interface is implemented by a Framework, at least one of the methods shall be implemented as a minimum requirement. If load level notifications are supported, the createLoadLevelNotification() and destroyLoadLevelNotification() methods shall be implemented. If suspendNotification() is implemented, then resumeNotification() shall be implemented also. If a Framework is capable of invoking the IpAppLoadManager.queryAppLoadReq() method, then it shall implement queryAppLoadRes() and queryAppLoadErr() methods in this interface.

<<Interface>> IpLoadManager
reportLoad (loadLevel : in TpLoadLevel) : void queryLoadReq (serviceIDs : in TpServiceIDList, timeInterval : in TpTimeInterval) : void queryAppLoadRes (loadStatistics : in TpLoadStatisticList) : void queryAppLoadErr (loadStatisticsError : in TpLoadStatisticError) : void createLoadLevelNotification (serviceIDs : in TpServiceIDList) : void destroyLoadLevelNotification (serviceIDs : in TpServiceIDList) : void resumeNotification (serviceIDs : in TpServiceIDList) : void suspendNotification (serviceIDs : in TpServiceIDList) : void

Method
`reportLoad()`

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

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

Parameters

`loadLevel : in TpLoadLevel`

Specifies the application's load level.

Raises

`TpCommonExceptions`

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.   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, 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, P_UNAUTHORISED_PARAMETER_VALUE`

Method
`queryAppLoadRes()`

The client application 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 queryAppLoadReq method on the IpAppLoadManager interface.

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the application-supplied load statistics.

Raises

`TpCommonExceptions`

Method
`queryAppLoadErr()`

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

Parameters

`loadStatisticsError : in TpLoadStatisticError`

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

Raises

`TpCommonExceptions`

Method
`createLoadLevelNotification()`

The client application uses this method to register to receive notifications of load level changes associated with either the framework or with its instances of the 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 SCFs to be registered for load control. To register for framework load control, the serviceIDs parameter must be an empty list.

Raises

`TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE`

Method
`destroyLoadLevelNotification()`

The client application uses this method to unregister for notifications of load level changes associated with either the framework or with its instances of the 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 level changes should no longer be reported. To unregister for framework load control, the serviceIDs parameter must be an empty list.

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 either the framework 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. 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 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.

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 either the framework or with its instances of the 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 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.

Raises

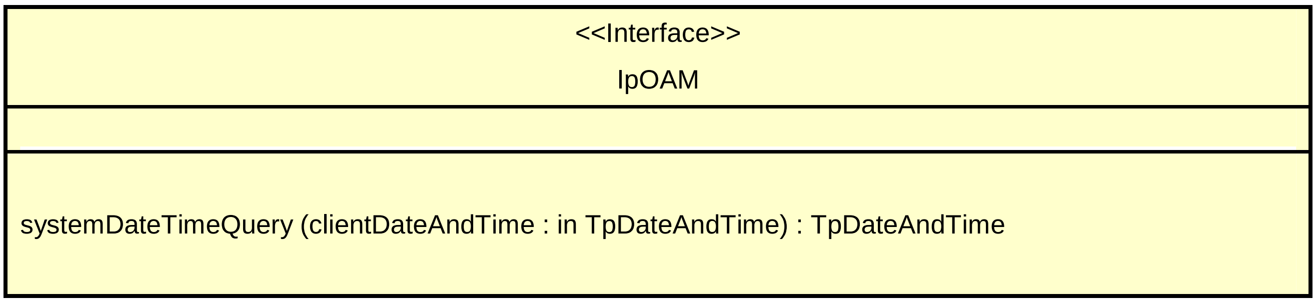
TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

7.3.3.9 Interface Class IpOAM

Inherits from: IpInterface.

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

This interface and ~~its~~the systemDateTimeQuery() method are optional.


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

Method

systemDateTimeQuery()

This method is used to query the system date and time. The client application 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 (application). The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

`TpDateAndTime`

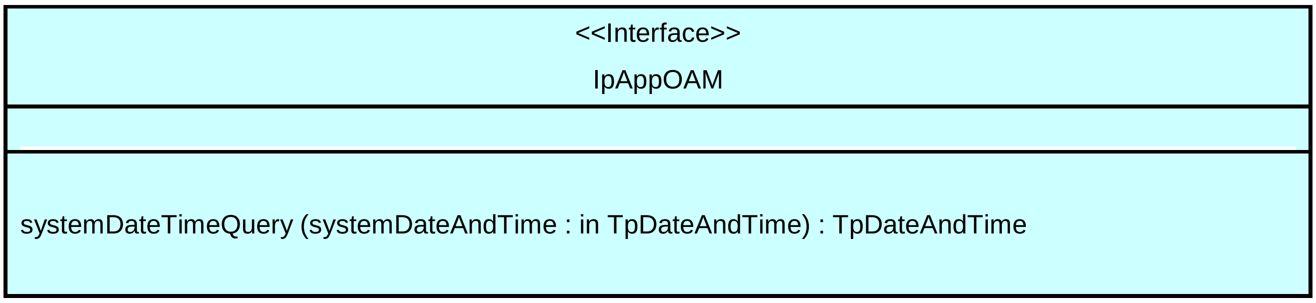
Raises

`TpCommonExceptions,P_INVALID_TIME_AND_DATE_FORMAT`

7.3.3.10 Interface Class IpAppOAM

Inherits from: IpInterface.

The OAM client application interface is used by the Framework to query the application date and time, for synchronisation purposes. This method is invoked by the Framework to interchange the framework and client application date and time.


<<Interface>>
IpAppOAM
systemDateTimeQuery (systemDateAndTime : in TpDateAndTime) : TpDateAndTime

Method

`systemDateTimeQuery()`

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

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

Parameters

`systemDateAndTime : in TpDateAndTime`

This is the system date and time of the framework.

Returns

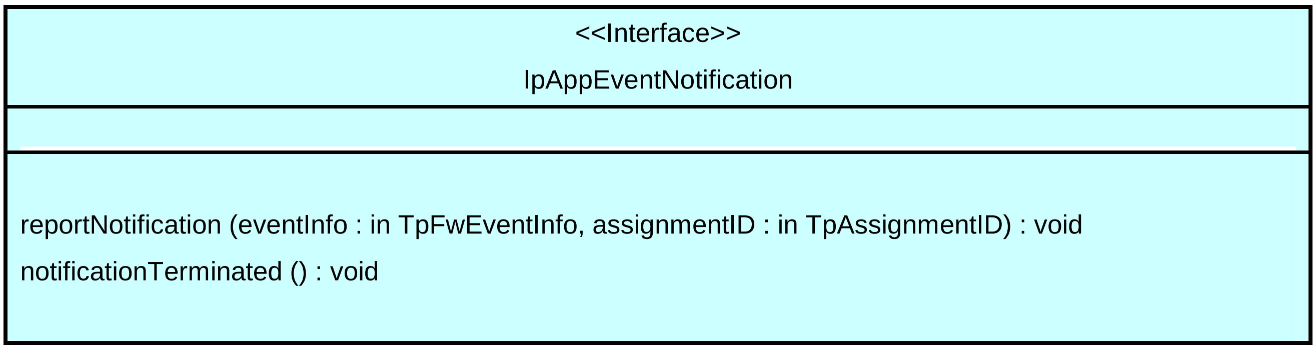
`TpDateAndTime`

7.3.4 Event Notification Interface Classes

7.3.4.1 Interface Class IpAppEventNotification

Inherits from: IpInterface.

This interface is used by the services to inform the application of a generic service-related event. The Event Notification Framework will invoke methods on the Event Notification Application Interface that is specified when the Event Notification interface is obtained.



<<Interface>>
IpAppEventNotification
reportNotification (eventInfo : in TpFwEventInfo, assignmentID : in TpAssignmentID) : void
notificationTerminated () : void

Method
`reportNotification()`

This method notifies the application 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 application can use assignment id to associate events with event specific criteria and to act accordingly.

Method
`notificationTerminated()`

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

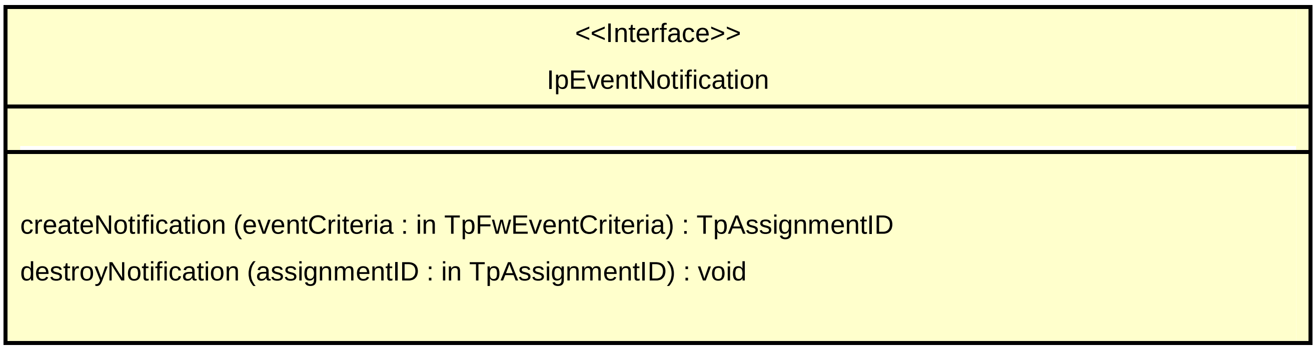
Parameters
No Parameters were identified for this method

7.3.4.2 Interface Class IpEventNotification

Inherits from: IpInterface.

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

If Event Notifications are supported by a Framework, this interface and ~~all its~~ the createNotification() and destroyNotification() methods shall be supported.



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

Method
`createNotification()`

This method is used to enable generic notifications so that events can be sent to the application.

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

Parameters

`eventCriteria : in TpFwEventCriteria`

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

Returns

`TpAssignmentID`

Raises

`TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE`

Method
`destroyNotification()`

This method is used by the application 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 return the error code P_INVALID_ASSIGNMENTID.

Raises

`TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_ASSIGNMENT_ID`

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-030076**
Meeting #22, Bangkok, THAILAND, 27 – 31 January 2003

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**074**	⌘ **rev**	**-**	⌘	Current version:	**5.1.0**	⌘

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

Proposed change affects:	UICC apps⌘ ☐	ME ☐	Radio Access Network ☐	Core Network **X**

Title:	⌘	Add Initial Load Notification report for Framework Integrity Management Load Notification model		
Source:	⌘	N5		
Work item code:	⌘	OSA2	***Date:*** ⌘	14/02/2003
Category:	⌘	**A**	***Release:*** ⌘	*REL-5*
		Use one of the following categories: ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.		*Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *Rel-4 (Release 4)* *Rel-5 (Release 5)* *Rel-6 (Release 6)*

Reason for change:	⌘	When load notifications are enabled or resumed, currently there is no initial load report. Changes in load subsequently result in load reports in which the current load is reported. In the absence of an initial load report it is not possible to determine whether the load condition is improving or worsening.
Summary of change:	⌘	For all methods that enable or resume load notifications modify behaviour to ensure that an initial load report is produced
Consequences if not approved:	⌘	Load Level Notification is incomplete and cannot be used as intended.

Clauses affected:	⌘	7.1.2.1, 7.1.2.5, 7.3.2.7, 7.3.2.8, 7.4.3.1, 8.1.4.1, 8.3.2.7, 8.3.2.8, 8.4.4.1				
		Y	**N**			
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/specs/CR.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.

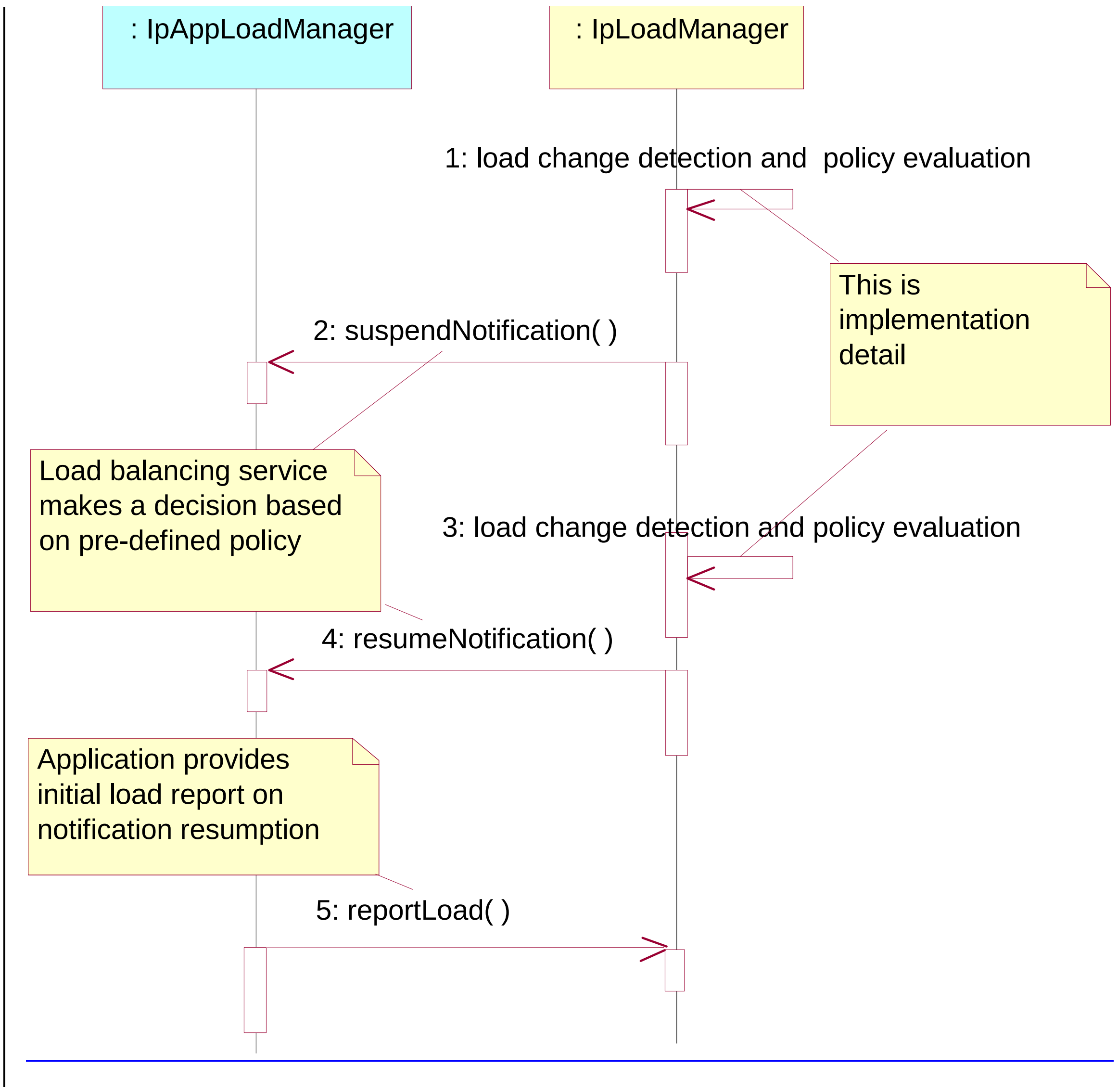
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

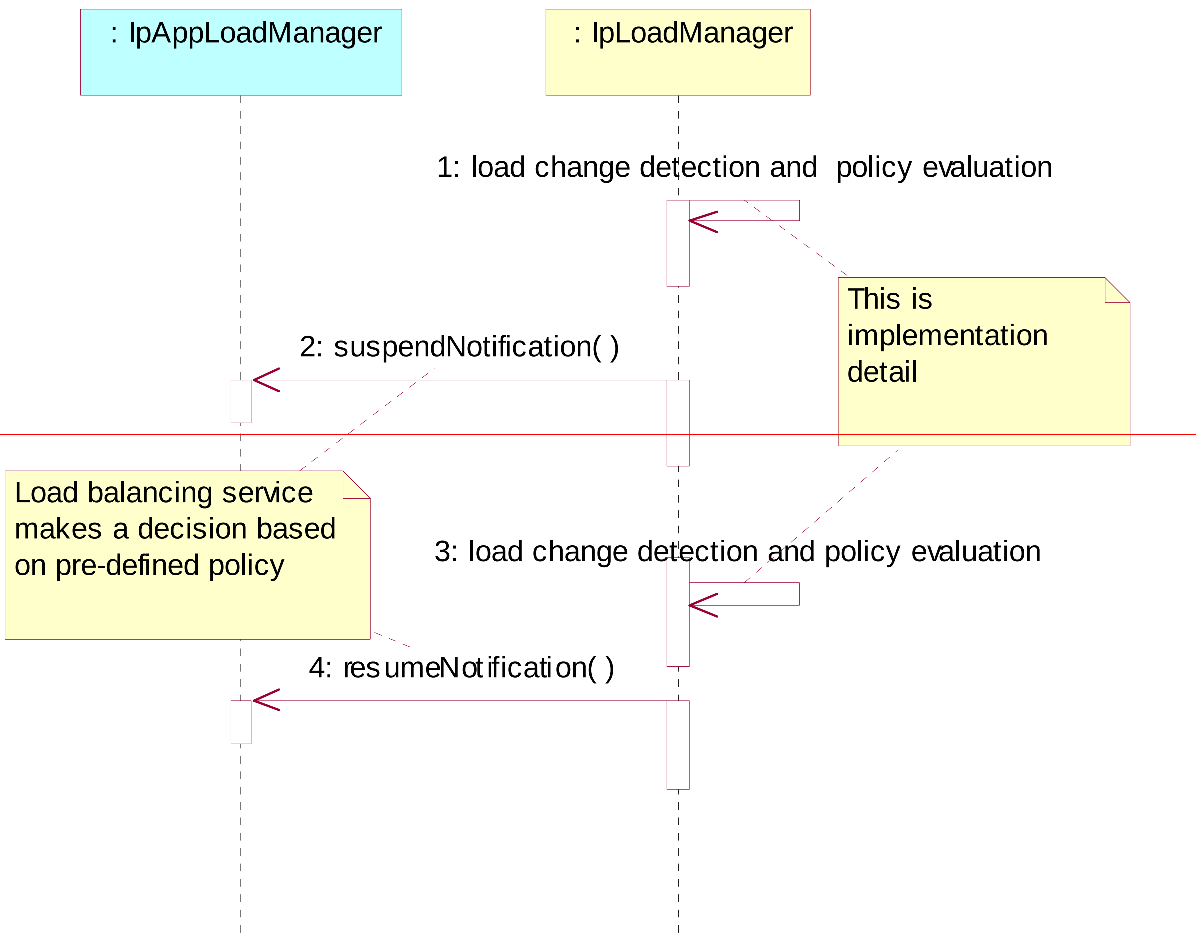
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

************ Start of change # 1 *****************

7.3.2.8 Load Management: Suspend/resume notification from application

This sequence diagram shows the scenario of suspending or resuming notifications from the application based on the evaluation of the load balancing policy as a result of the detection of a change in load level of the framework.


: IpAppLoadManager
: IpLoadManager
1: load change detection and policy evaluation
This is implementation detail
2: suspendNotification( )
Load balancing service makes a decision based on pre-defined policy
3: load change detection and policy evaluation
4: resumeNotification( )
Application provides initial load report on notification resumption
5: reportLoad( )

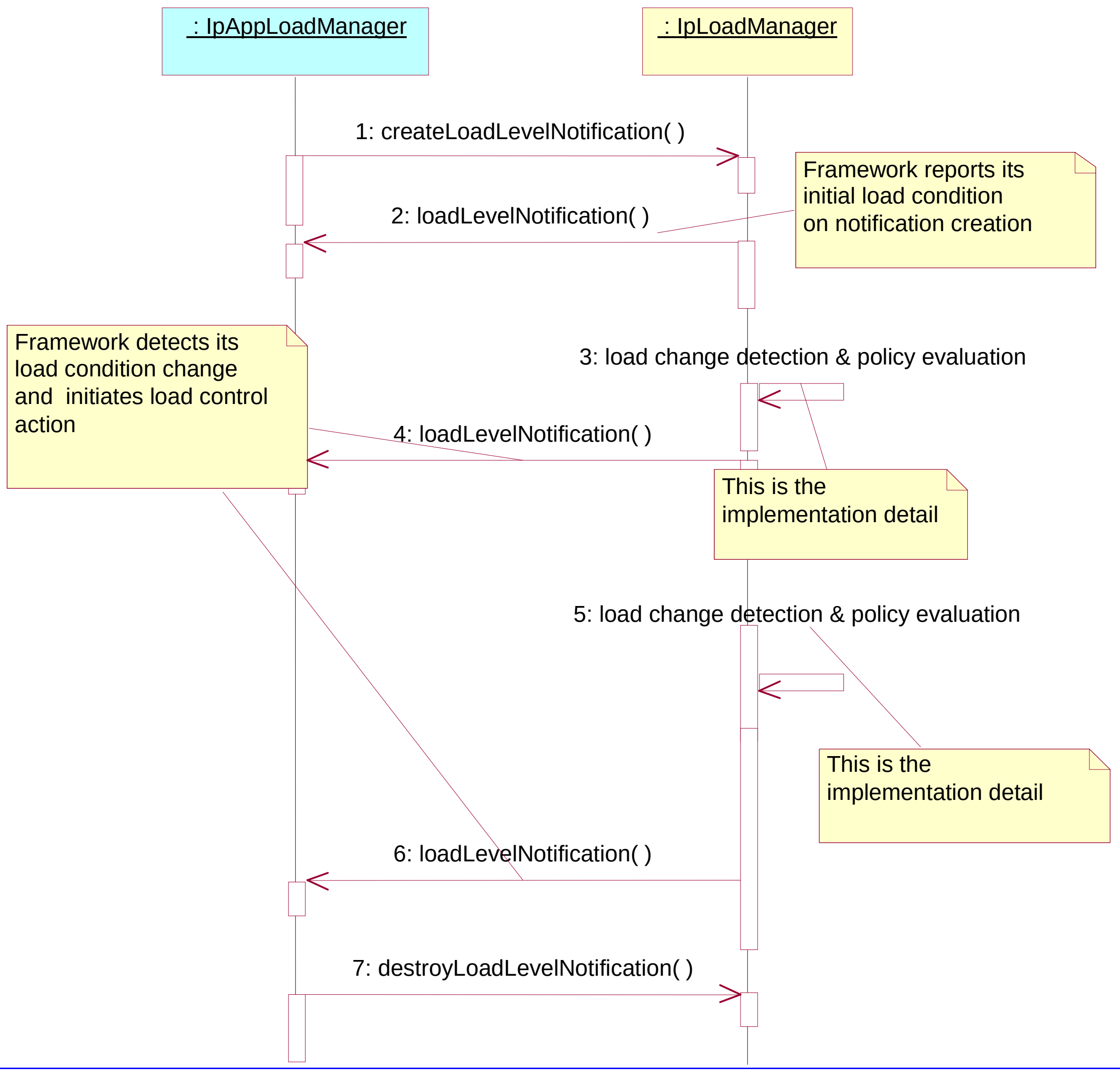

: IpAppLoadManager
: IpLoadManager
1: load change detection and policy evaluation
This is implementation detail
2: suspendNotification( )
Load balancing service makes a decision based on pre-defined policy
3: load change detection and policy evaluation
4: resumeNotification( )

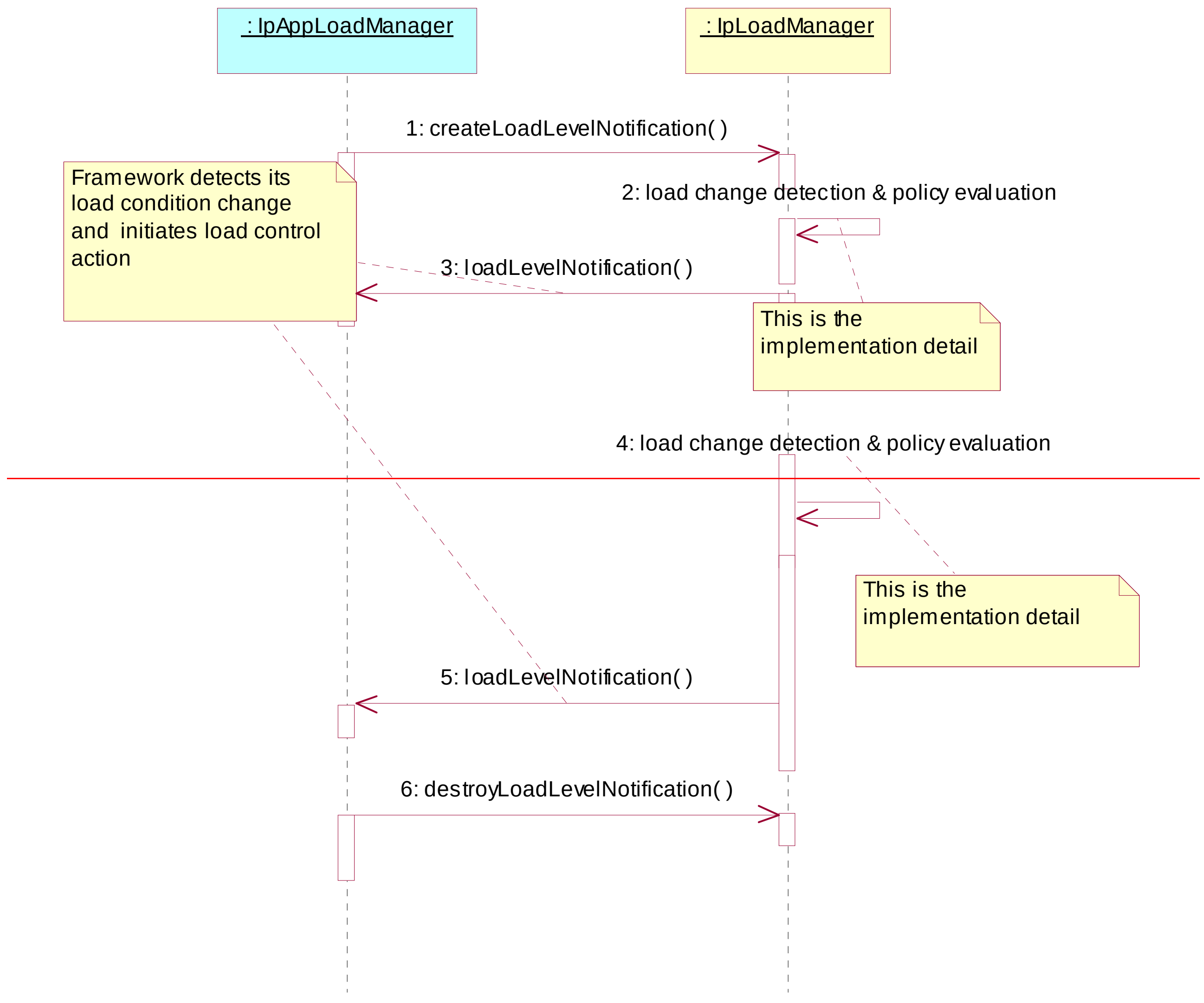
************ End of change # 1 ****************

************ Start of change # 2 ****************

7.1.2.5 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( )
2: loadLevelNotification( )
Framework reports its initial load condition on notification creation
3: load change detection & policy evaluation
Framework detects its load condition change and initiates load control action
4: loadLevelNotification( )
This is the implementation detail
5: load change detection & policy evaluation
This is the implementation detail
6: loadLevelNotification( )
7: destroyLoadLevelNotification( )


: 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( )

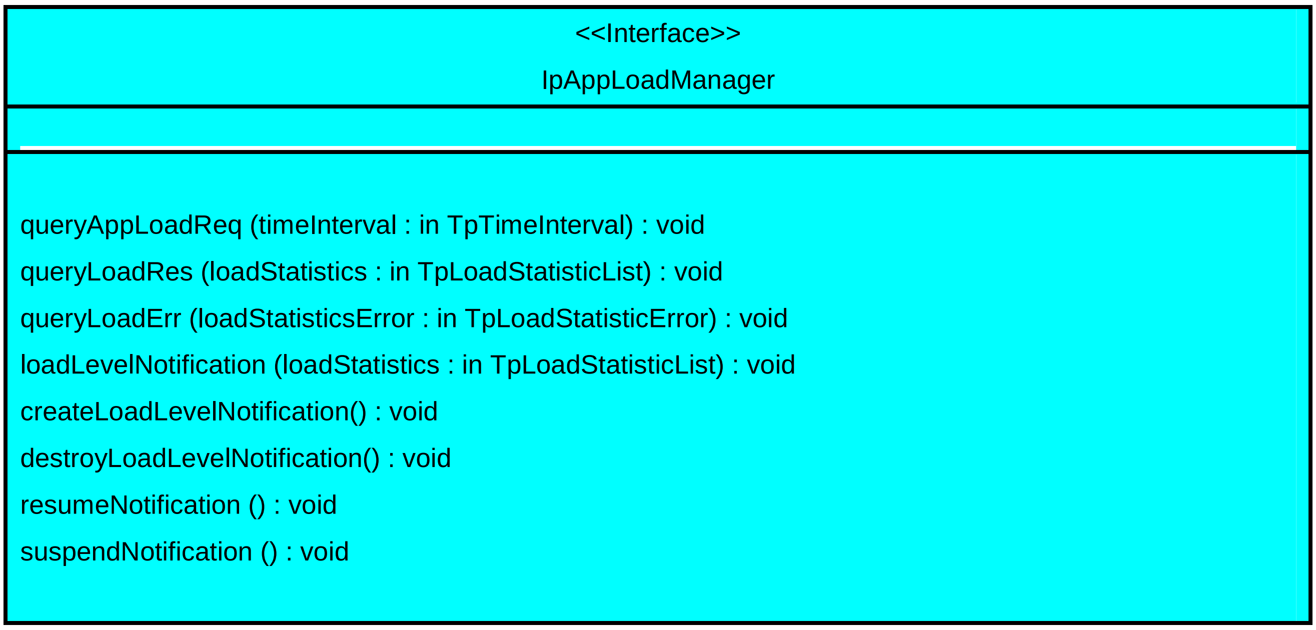
************ End of change # 2 *****************

************ Start of change # 3 *****************

7.3.2.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

The client application developer supplies the load manager application interface to handle requests, reports and other responses from the framework load manager function. 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() method on the IpAccess interface.


<<Interface>>
IpAppLoadManager
queryAppLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
createLoadLevelNotification() : void
destroyLoadLevelNotification() : void
resumeNotification () : void
suspendNotification () : void

7.3.2.7.1 Method queryAppLoadReq()

The framework uses this method to request the application to provide load statistics records for the application.

Parameters

`timeInterval : in TpTimeInterval`

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

7.3.2.7.2 Method queryLoadRes()

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

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the framework-supplied load statistics

7.3.2.7.3 Method queryLoadErr()

The framework uses this method to return an error response to the application 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 IpLoadManager interface.

Parameters

`loadStatisticsError : in TpLoadStatisticError`

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

7.3.2.7.4 Method loadLevelNotification()

Upon detecting load condition change, (e.g. load level changing from 0 to 1, 0 to 2, 1 to 0, for the SCFs or framework which have been registered for load level notifications) this method is invoked on the application. In addition this method shall be invoked on the application in order to provide a notification of current load status, when load notifications are first requested, or resumed after suspension.

Parameters

`loadStatistics : in TpLoadStatisticList`

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

7.3.2.7.5 Method createLoadLevelNotification()

The framework uses this method to register to receive notifications of load level changes associated with the application. Upon receipt of this method the client application shall inform the framework of the current load using the reportLoad method on the corresponding IpLoadManager.

Parameters

No Parameters were identified for this method

7.3.2.7.6 Method destroyLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

7.3.2.7.7 Method resumeNotification()

The framework uses this method to request the application to resume sending it notifications: e.g. after a period of suspension during which the framework handled a temporary overload condition. Upon receipt of this method the client application shall inform the framework of the current load using the reportLoad method on the corresponding IpLoadManager.

Parameters

No Parameters were identified for this method

7.3.2.7.8 Method suspendNotification()

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

Parameters

No Parameters were identified for this method

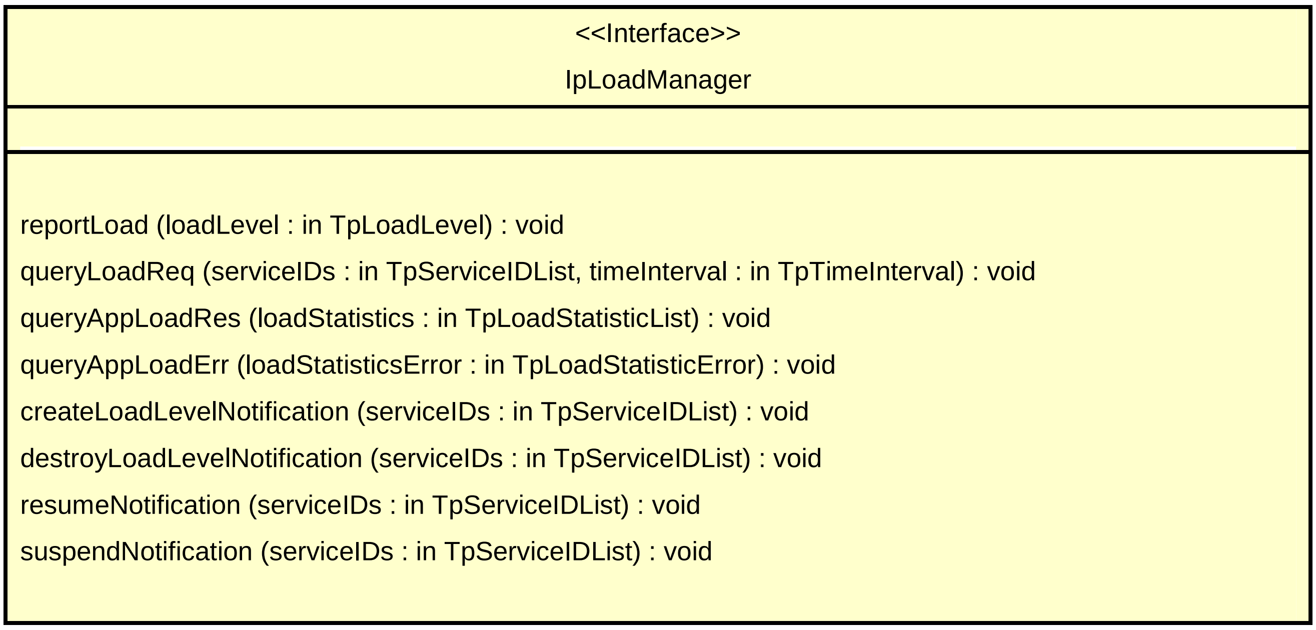
************ End of change # 3 *****************

************ Start of change # 4 *****************

7.3.2.8 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) : void
queryLoadReq (serviceIDs : in TpServiceIDList, timeInterval : in TpTimeInterval) : void
queryAppLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryAppLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
createLoadLevelNotification (serviceIDs : in TpServiceIDList) : void
destroyLoadLevelNotification (serviceIDs : in TpServiceIDList) : void
resumeNotification (serviceIDs : in TpServiceIDList) : void
suspendNotification (serviceIDs : in TpServiceIDList) : void

7.3.2.8.1 Method reportLoad()

The client application uses this method to report its current load level (0,1, or 2) to the framework: e.g. when the load level on the application has changed. In addition this method shall be called by the application in order to report current load status, when load notifications are first requested, or resumed after suspension.

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

Parameters

`loadLevel : in TpLoadLevel`

Specifies the application's load level.

Raises

`TpCommonExceptions`

7.3.2.8.2 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.   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, 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, P_UNAUTHORISED_PARAMETER_VALUE`

7.3.2.8.3 Method queryAppLoadRes()

The client application 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 queryAppLoadReq method on the IpAppLoadManager interface.

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the application-supplied load statistics.

Raises

`TpCommonExceptions`

7.3.2.8.4 Method queryAppLoadErr()

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

Parameters

`loadStatisticsError : in TpLoadStatisticError`

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

Raises

`TpCommonExceptions`

7.3.2.8.5 Method createLoadLevelNotification()

The client application uses this method to register to receive notifications of load level changes associated with either the framework or with its instances of the 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. Upon receipt of this method the framework shall inform the client application of the current framework or service instance load using the loadLevelNotification method on the corresponding IpAppLoadManager.

Parameters

`serviceIDs : in TpServiceIDList`

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

Raises

`TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE`

7.3.2.8.6 Method destroyLoadLevelNotification()

The client application uses this method to unregister for notifications of load level changes associated with either the framework or with its instances of the 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 level changes should no longer be reported. To unregister for framework load control, the serviceIDs parameter must be an empty list.

Raises

`TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE`

7.3.2.8.7 Method resumeNotification()

The client application uses this method to request the framework to resume sending it load management notifications associated with either the framework 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. 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. Upon receipt of this method the framework shall inform the client application of the current framework or service instance load using the loadLevelNotification method on the corresponding IpAppLoadManager.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework 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.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

7.3.2.8.8 Method suspendNotification()

The client application uses this method to request the framework to suspend sending it load management notifications associated with either the framework or with its instances of the 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 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.

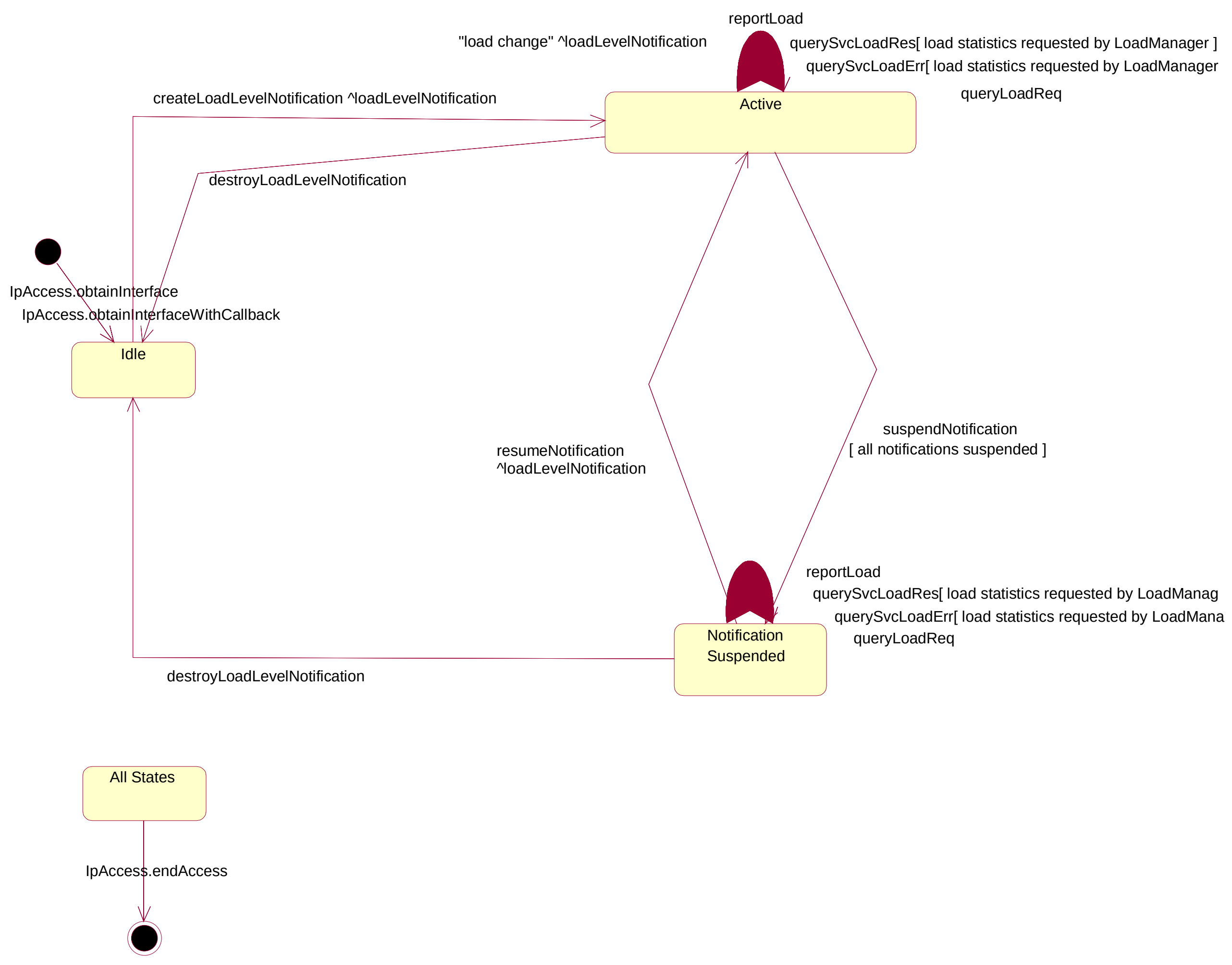
Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

************ End of change # 4 *****************

************ Start of change # 5 *****************

7.4.3.1 State Transition Diagrams for IpLoadManager


reportLoad
"load change" ^loadLevelNotification
querySvcLoadRes[load statistics requested by LoadManager]
querySvcLoadErr[load statistics requested by LoadManager
queryLoadReq
createLoadLevelNotification ^loadLevelNotification
Active
destroyLoadLevelNotification
IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
suspendNotification
[all notifications suspended]
resumeNotification
^loadLevelNotification
reportLoad
querySvcLoadRes[load statistics requested by LoadManag
querySvcLoadErr[load statistics requested by LoadMana
queryLoadReq
Notification
Suspended
destroyLoadLevelNotification
All States
IpAccess.endAccess

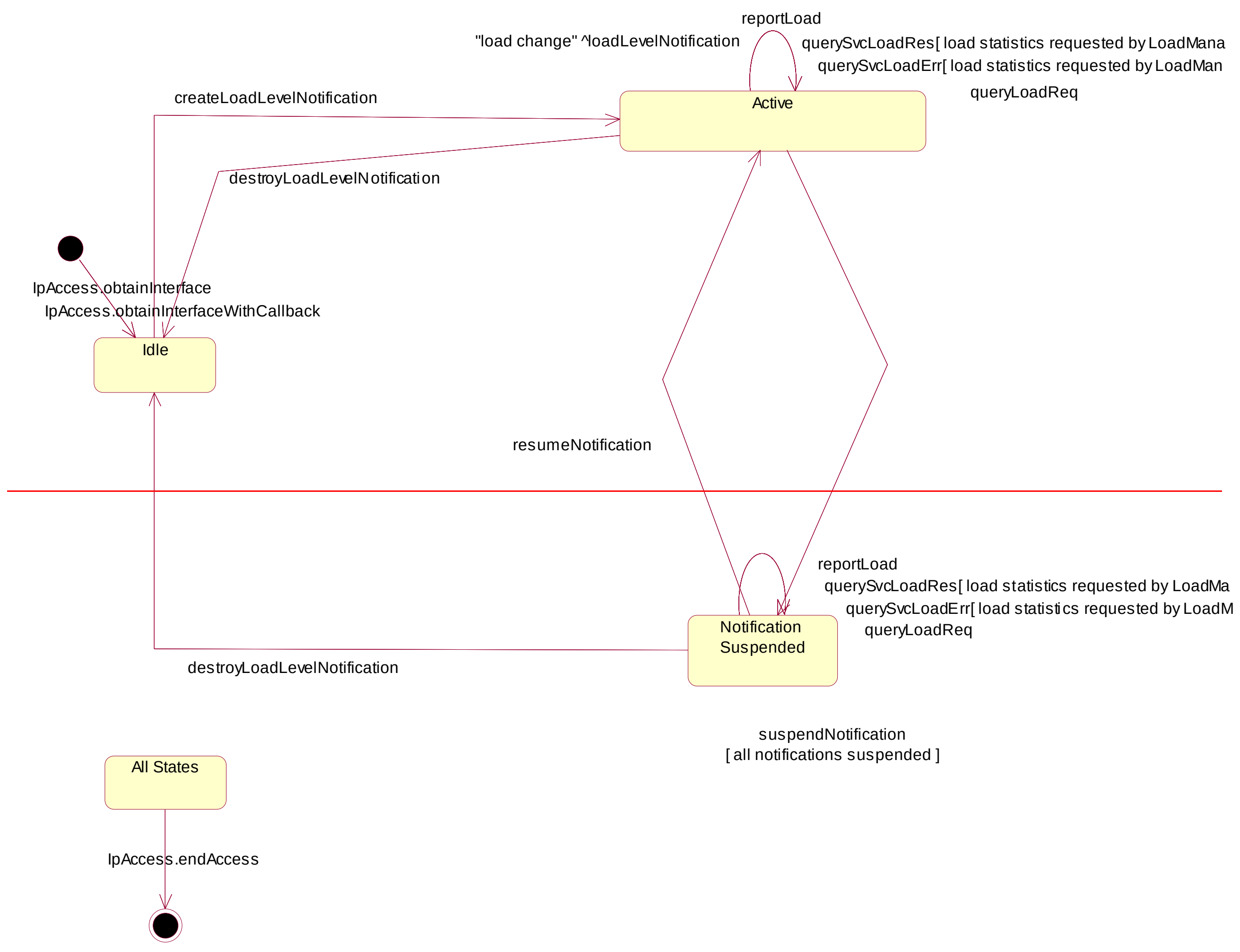

reportLoad
"load change" ^loadLevelNotification
querySvcLoadRes[load statistics requested by LoadMana
querySvcLoadErr[load statistics requested by LoadMan
queryLoadReq
createLoadLevelNotification
Active
destroyLoadLevelNotification
IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
resumeNotification
reportLoad
querySvcLoadRes[load statistics requested by LoadMa
querySvcLoadErr[load statistics requested by LoadM
queryLoadReq
Notification Suspended
destroyLoadLevelNotification
suspendNotification
[all notifications suspended]
All States
IpAccess.endAccess

Figure : State Transition Diagram for IpLoadManager

7.4.3.1.1 Idle State

In this state the application has obtained an interface reference of the LoadManager from the IpAccess interface.

7.4.3.1.2 Notification Suspended State

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

7.4.3.1.3 Active State

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().

7.4.3.2 State Transition Diagrams for LoadManagerInternal

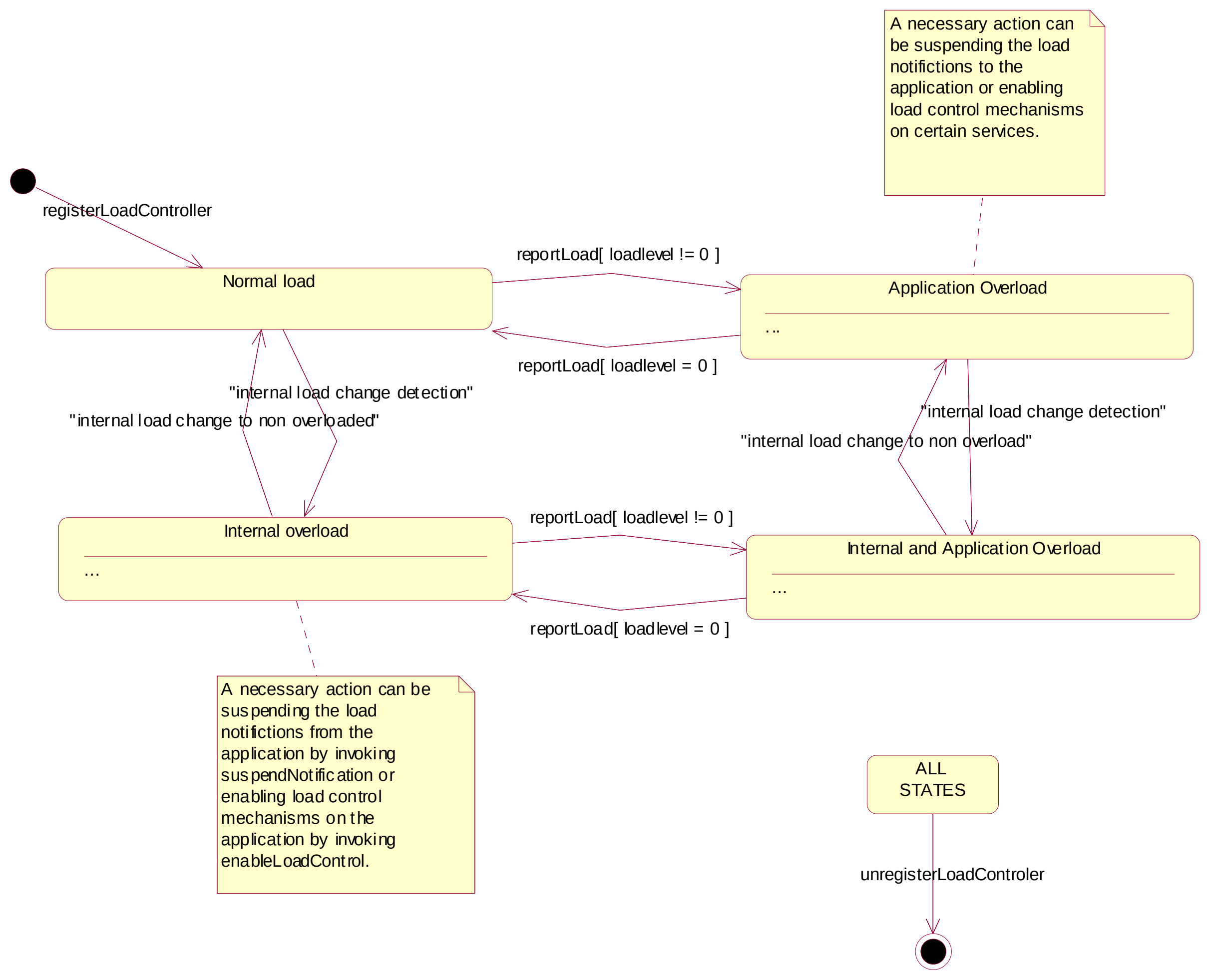

A necessary action can be suspending the load notifictions to the application or enabling load control mechanisms on certain services.
registerLoadController
Normal load
reportLoad[loadlevel != 0]
Application Overload
...
reportLoad[loadlevel = 0]
"internal load change detection"
"internal load change to non overloaded"
"internal load change detection"
"internal load change to non overload"
Internal overload
...
reportLoad[loadlevel != 0]
Internal and Application Overload
...
reportLoad[loadlevel = 0]
A necessary action can be suspending the load notifictions from the application by invoking suspendNotification or enabling load control mechanisms on the application by invoking enableLoadControl.
ALL STATES
unregisterLoadControler

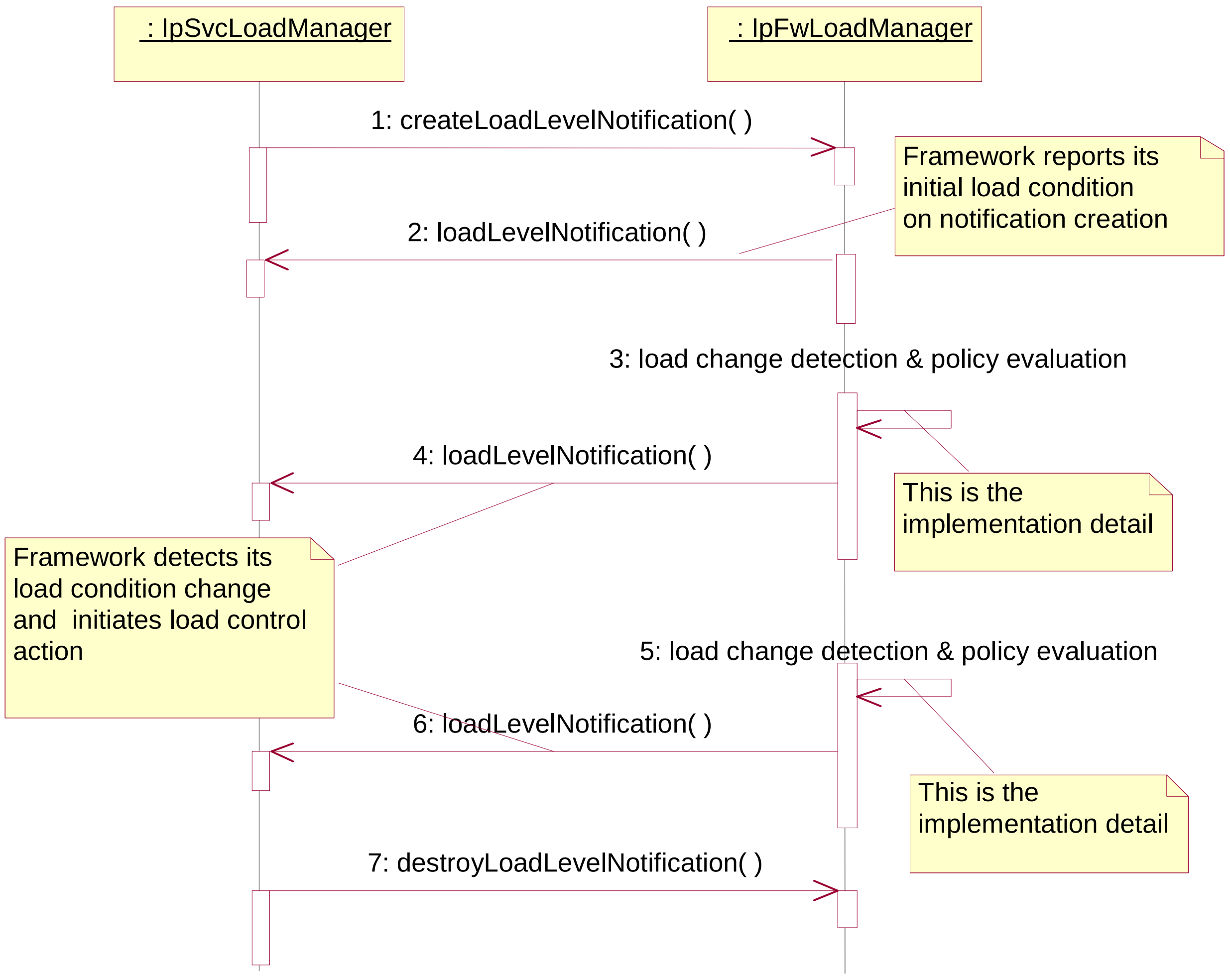
Figure : State Transition Diagram for LoadManagerInternal

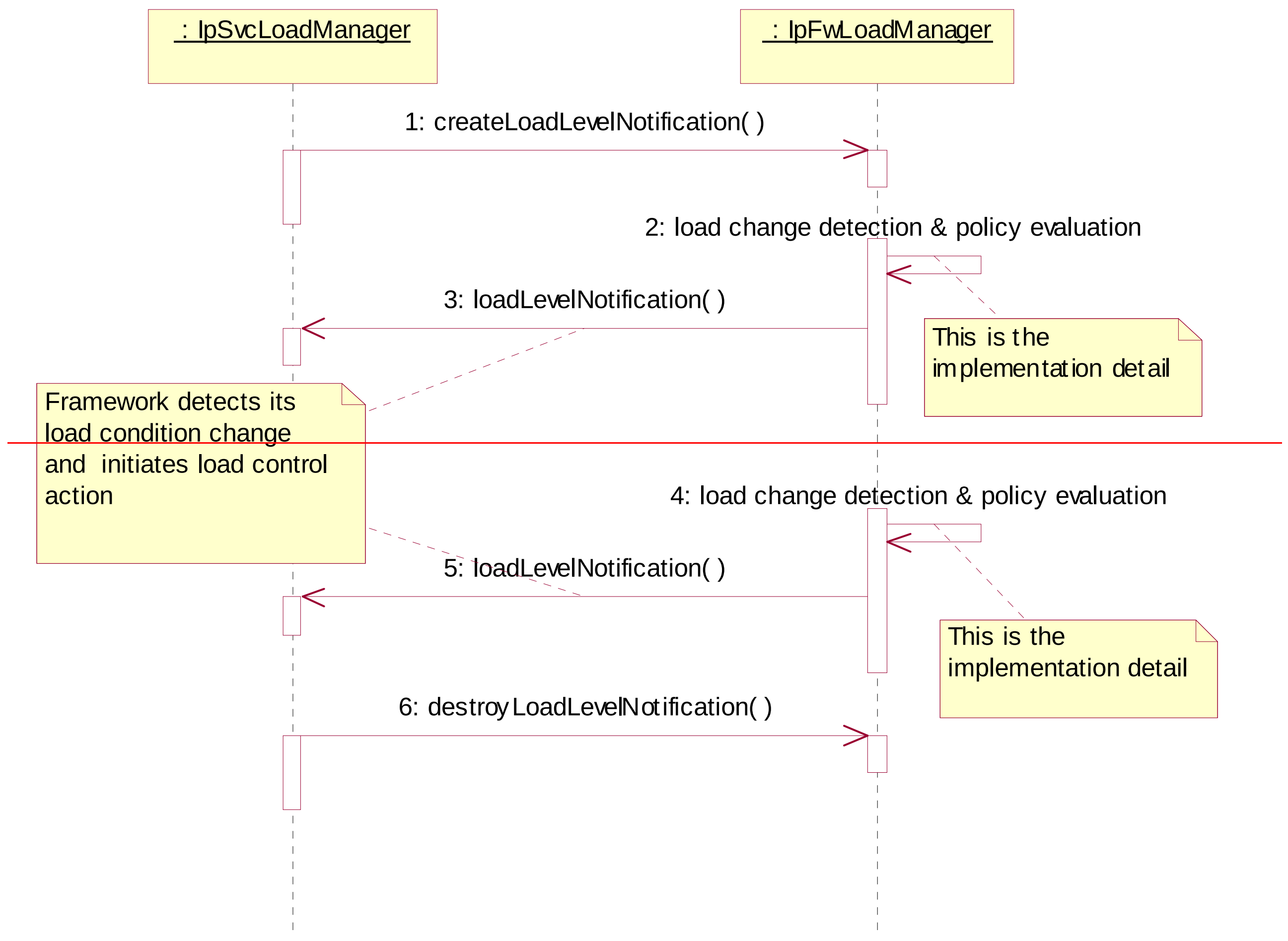
************ End of change # 5 *****************

************ Start of change # 6 *****************

8.1.4.1 Load Management: Service callback registration and load control

This sequence diagram shows how a service registers itself and the framework invokes load management function based on policy


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


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

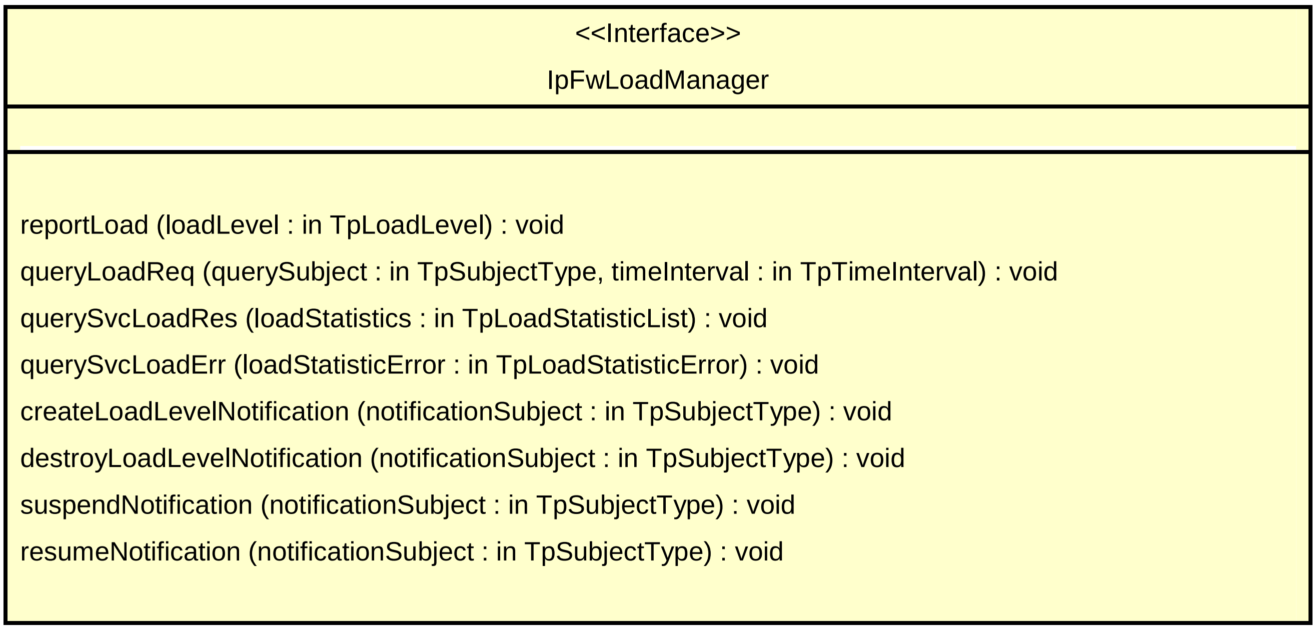
************ End of change # 6 *****************

************ Start of change # 7 *****************

8.3.2.7 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.


<<Interface>>

IpFwLoadManager

reportLoad (loadLevel : in TpLoadLevel) : void

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

querySvcLoadRes (loadStatistics : in TpLoadStatisticList) : void

querySvcLoadErr (loadStatisticError : in TpLoadStatisticError) : void

createLoadLevelNotification (notificationSubject : in TpSubjectType) : void

destroyLoadLevelNotification (notificationSubject : in TpSubjectType) : void

suspendNotification (notificationSubject : in TpSubjectType) : void

resumeNotification (notificationSubject : in TpSubjectType) : void

8.3.2.7.1 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. In addition this method shall be called by the service instance in order to report current load status, when load notifications are first requested, or resumed after suspension.

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`

8.3.2.7.2 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 application) 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`

8.3.2.7.3 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`

8.3.2.7.4 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`

8.3.2.7.5 Method createLoadLevelNotification()

The service instance 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. Upon receipt of this method the framework shall inform the service instance of the current framework or application load using the loadLevelNotification method on the corresponding IpSvcLoadManager.

Parameters

`notificationSubject : in TpSubjectType`

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

Raises

`TpCommonExceptions`

8.3.2.7.6 Method destroyLoadLevelNotification()

The service instance 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

`notificationSubject : in TpSubjectType`

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

Raises

`TpCommonExceptions`

8.3.2.7.7 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

`notificationSubject : in TpSubjectType`

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

Raises

`TpCommonExceptions`

8.3.2.7.8 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. Upon receipt of this method the framework shall inform the service instance of the current framework or application load using the loadLevelNotification method on the corresponding IpSvcLoadManager.

Parameters

`notificationSubject : 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`

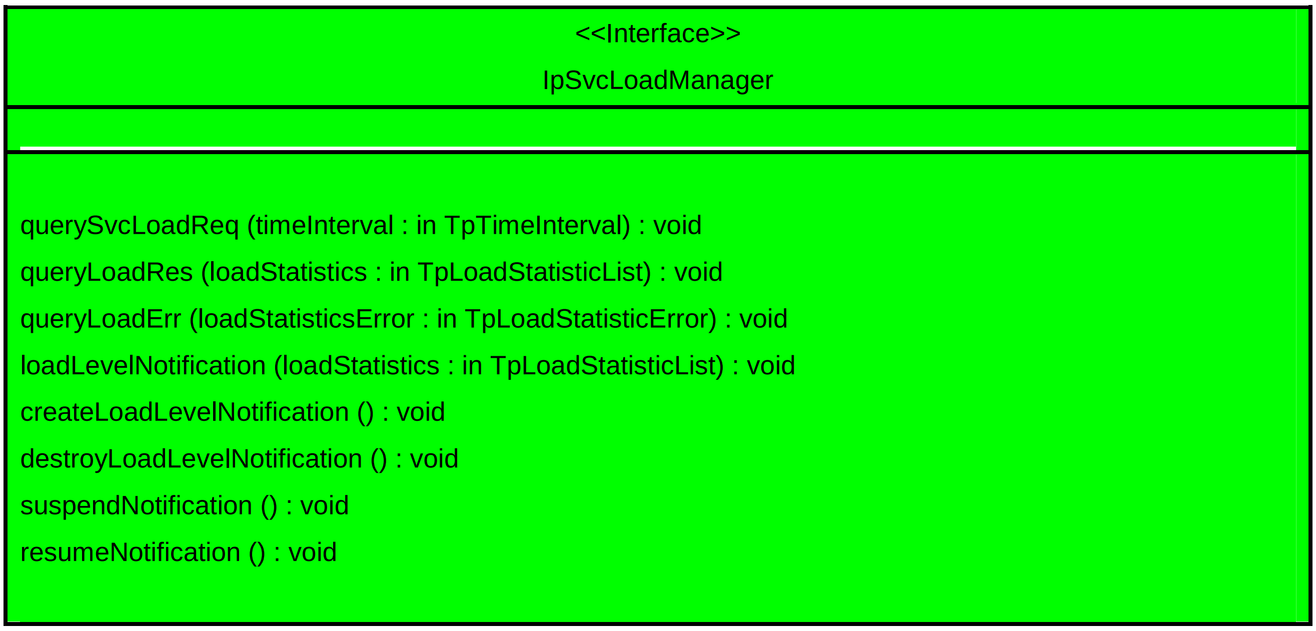
************ End of change # 7 *****************

************ Start of change # 8 *****************

8.3.2.8 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 IpAccess interface.


<<Interface>>
IpSvcLoadManager
querySvcLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
createLoadLevelNotification () : void
destroyLoadLevelNotification () : void
suspendNotification () : void
resumeNotification () : void

8.3.2.8.1 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`

8.3.2.8.2 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`

8.3.2.8.3 Method queryLoadErr()

The framework uses this method to return an error response to the service 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`

8.3.2.8.4 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 or framework which has been registered for load level notifications) this method is invoked on the SCF. In addition this method shall be invoked on the SCF in order to provide a notification of current load status, when load notifications are first requested, or resumed after suspension.

Parameters

`loadStatistics : in TpLoadStatisticList`

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

Raises

`TpCommonExceptions`

8.3.2.8.5 Method createLoadLevelNotification()

The framework uses this method to register to receive notifications of load level changes associated with the service instance. Upon receipt of this method the service instance shall inform the framework of the current load using the reportLoad method on the corresponding IpFwLoadManager.

Parameters

No Parameters were identified for this method

Raises

`TpCommonExceptions`

8.3.2.8.6 Method destroyLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

Raises

`TpCommonExceptions`

8.3.2.8.7 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`

8.3.2.8.8 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. Upon receipt of this method the service instance shall inform the framework of the current load using the reportLoad method on the corresponding IpFwLoadManager.

Parameters
No Parameters were identified for this method

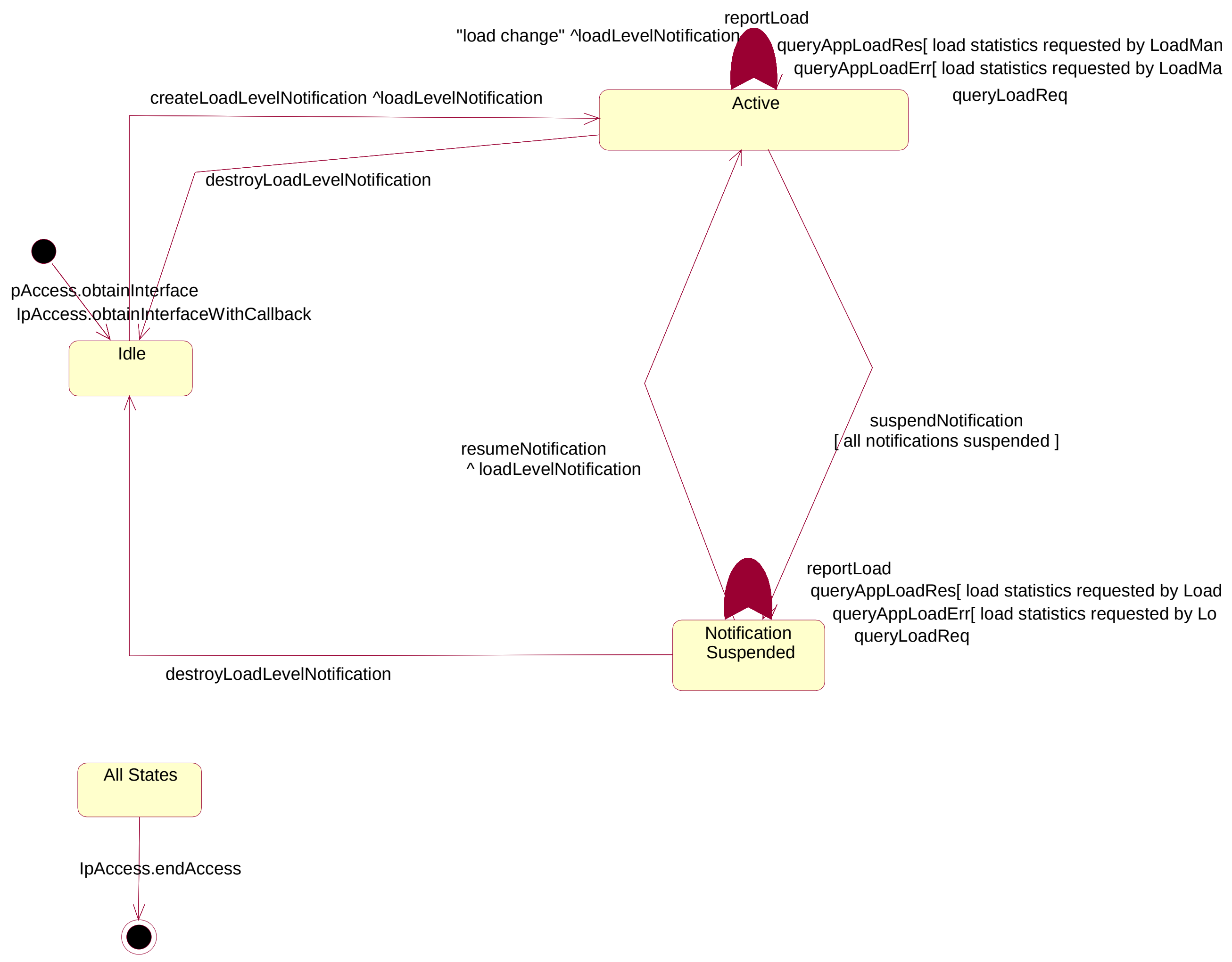
Raises

`TpCommonExceptions`

************ End of change # 8 *****************

************ Start of change # 9 *****************

8.4.4.1 State Transition Diagrams for IpFwLoadManager


reportLoad
"load change" ^loadLevelNotification
queryAppLoadRes[load statistics requested by LoadMan
queryAppLoadErr[load statistics requested by LoadMa
queryLoadReq
createLoadLevelNotification ^loadLevelNotification
Active
destroyLoadLevelNotification
pAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
suspendNotification
[all notifications suspended]
resumeNotification
^ loadLevelNotification
reportLoad
queryAppLoadRes[load statistics requested by Load
queryAppLoadErr[load statistics requested by Lo
queryLoadReq
Notification Suspended
destroyLoadLevelNotification
All States
IpAccess.endAccess

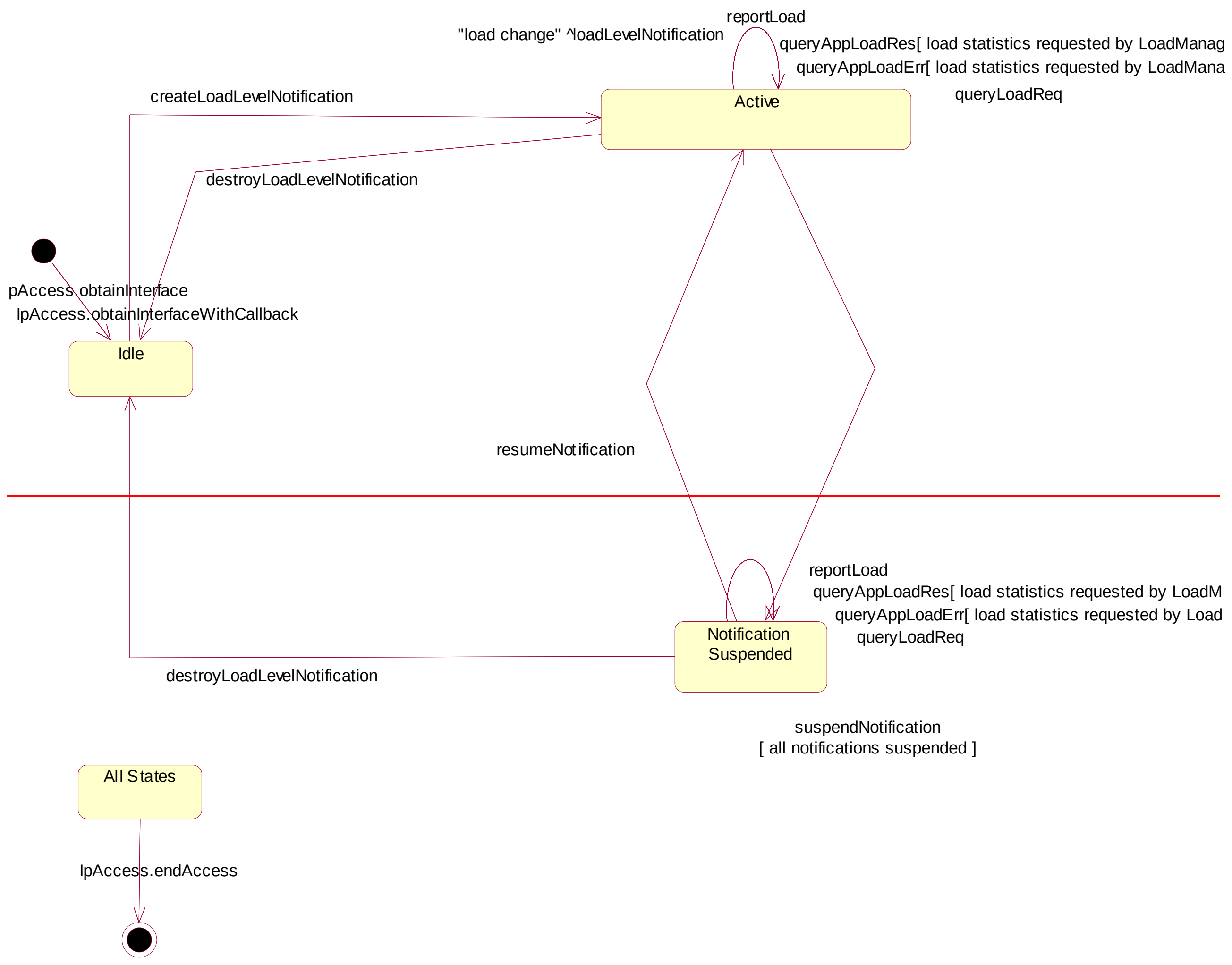

reportLoad
"load change" ^loadLevelNotification
queryAppLoadRes[load statistics requested by LoadManag
queryAppLoadErr[load statistics requested by LoadMana
queryLoadReq
createLoadLevelNotification
Active
destroyLoadLevelNotification
pAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
resumeNotification
reportLoad
queryAppLoadRes[load statistics requested by LoadM
queryAppLoadErr[load statistics requested by Load
queryLoadReq
Notification Suspended
destroyLoadLevelNotification
suspendNotification
[all notifications suspended]
All States
IpAccess.endAccess

Figure : State Transition Diagram for IpFwLoadManager

************ Start of change # 9 *****************

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-030075**
Meeting #22, Bangkok, THAILAND, 27 – 31 January 2003

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**073**	⌘**rev**	**-**	⌘	Current version:	**4.6.0** ⌘

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

Proposed change affects: UICC apps⌘ [ ] ME [ ] Radio Access Network [ ] Core Network [X]

Title:	⌘	Add Initial Load Notification report for Framework Integrity Management Load Notification model		
Source:	⌘	N5		
Work item code:	⌘	OSA1	***Date:*** ⌘	14/02/2003
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)* *Rel-6 (Release 6)*

Reason for change:	⌘	When load notifications are enabled or resumed, currently there is no initial load report. Changes in load subsequently result in load reports in which the current load is reported. In the absence of an initial load report it is not possible to determine whether the load condition is improving or worsening.
Summary of change:	⌘	For all methods that enable or resume load notifications modify behaviour to ensure that an initial load report is produced
Consequences if not approved:	⌘	Load Level Notification is incomplete and cannot be used as intended.

Clauses affected:	⌘	7.1.2.1, 7.1.2.5, 7.3.3.7, 7.3.3.8, 7.4.3.1, 8.1.4.1, 8.3.4.7, 8.3.4.8, 8.4.4.1				
		Y	**N**			
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/specs/CR.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.

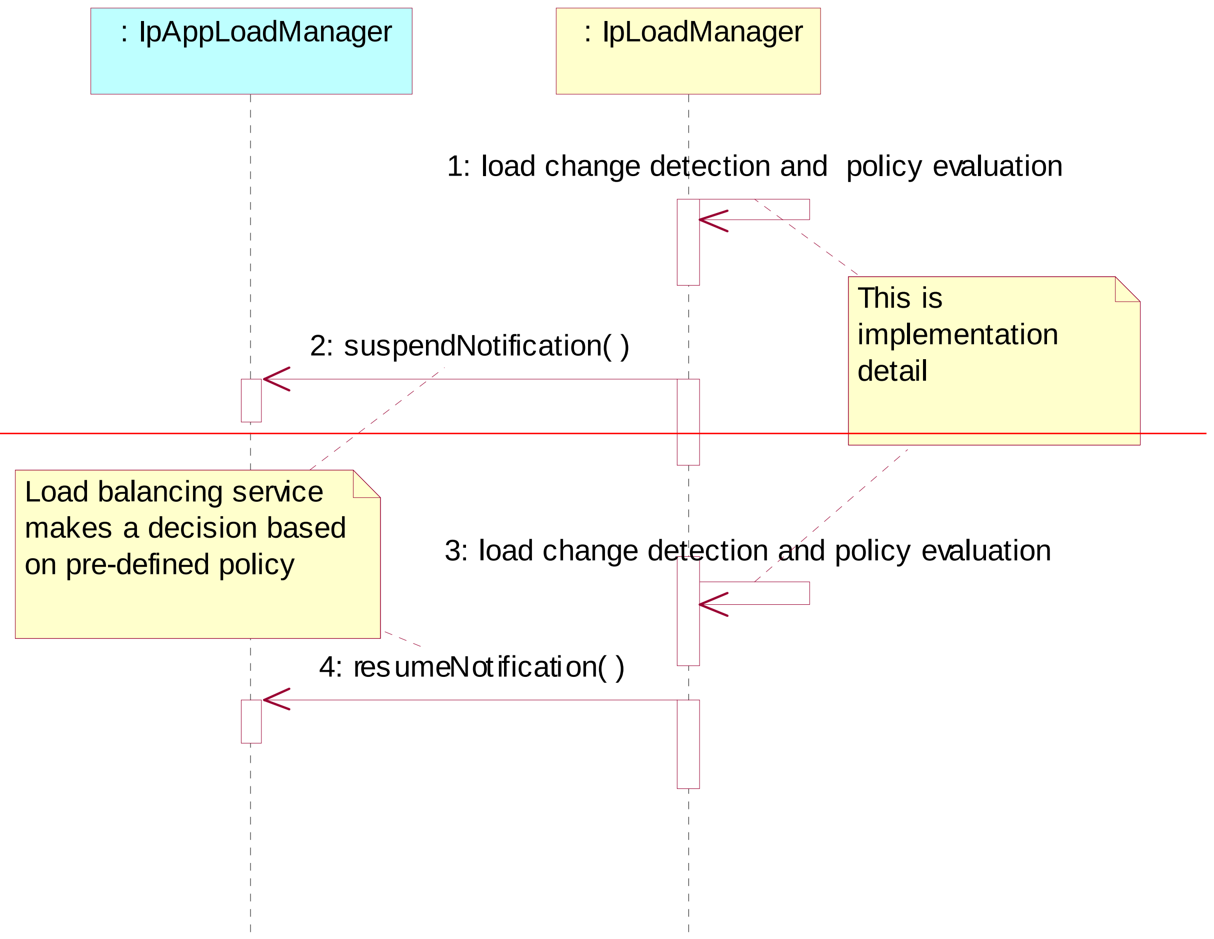
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

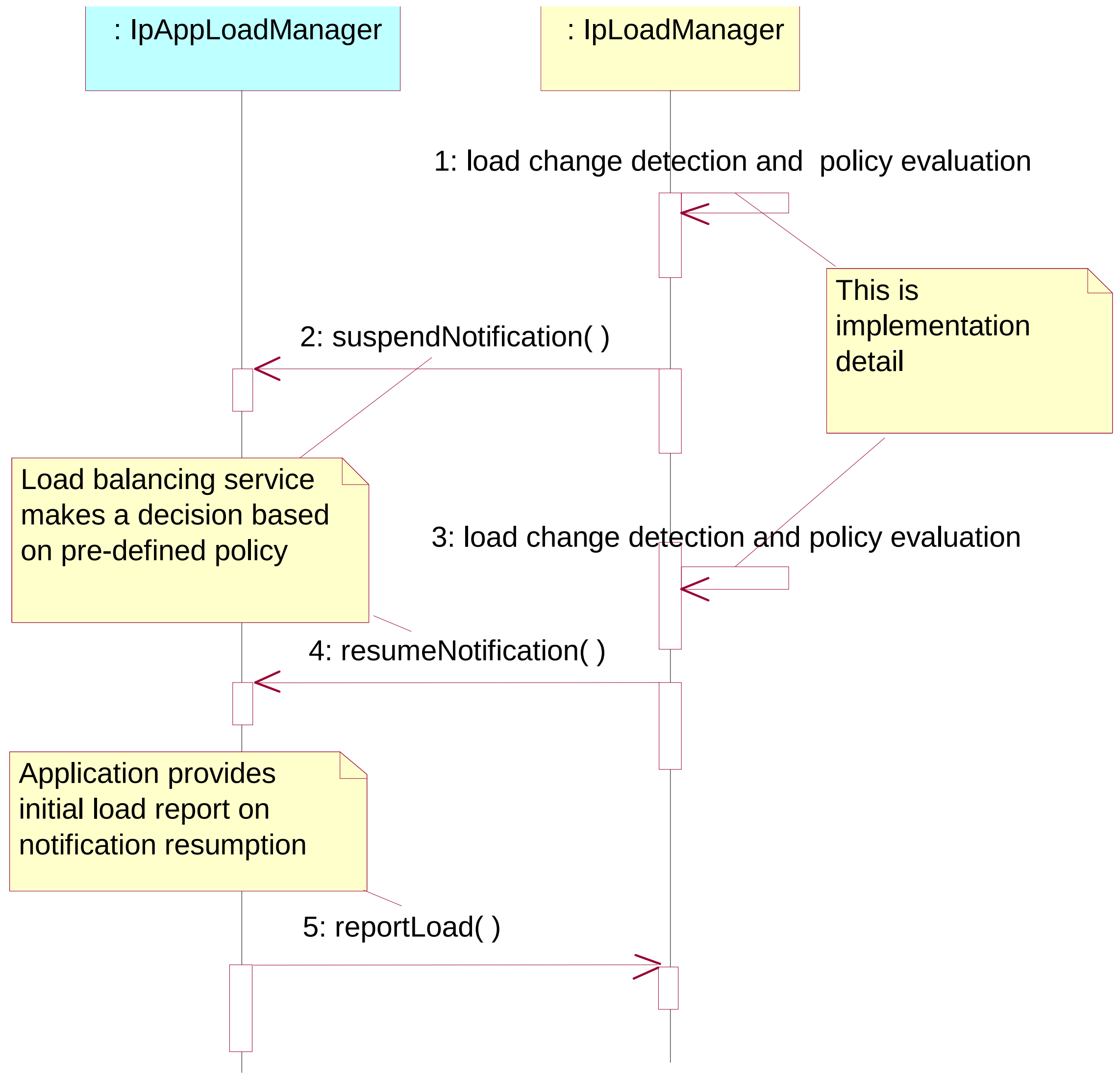
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

************ Start of change # 1 *****************

7.1.2.1 Load Management: Suspend/resume notification from application

This sequence diagram shows the scenario of suspending or resuming notifications from the application based on the evaluation of the load balancing policy as a result of the detection of a change in load level of the framework.


: IpAppLoadManager
: IpLoadManager
1: load change detection and policy evaluation
This is implementation detail
2: suspendNotification( )
Load balancing service makes a decision based on pre-defined policy
3: load change detection and policy evaluation
4: resumeNotification( )

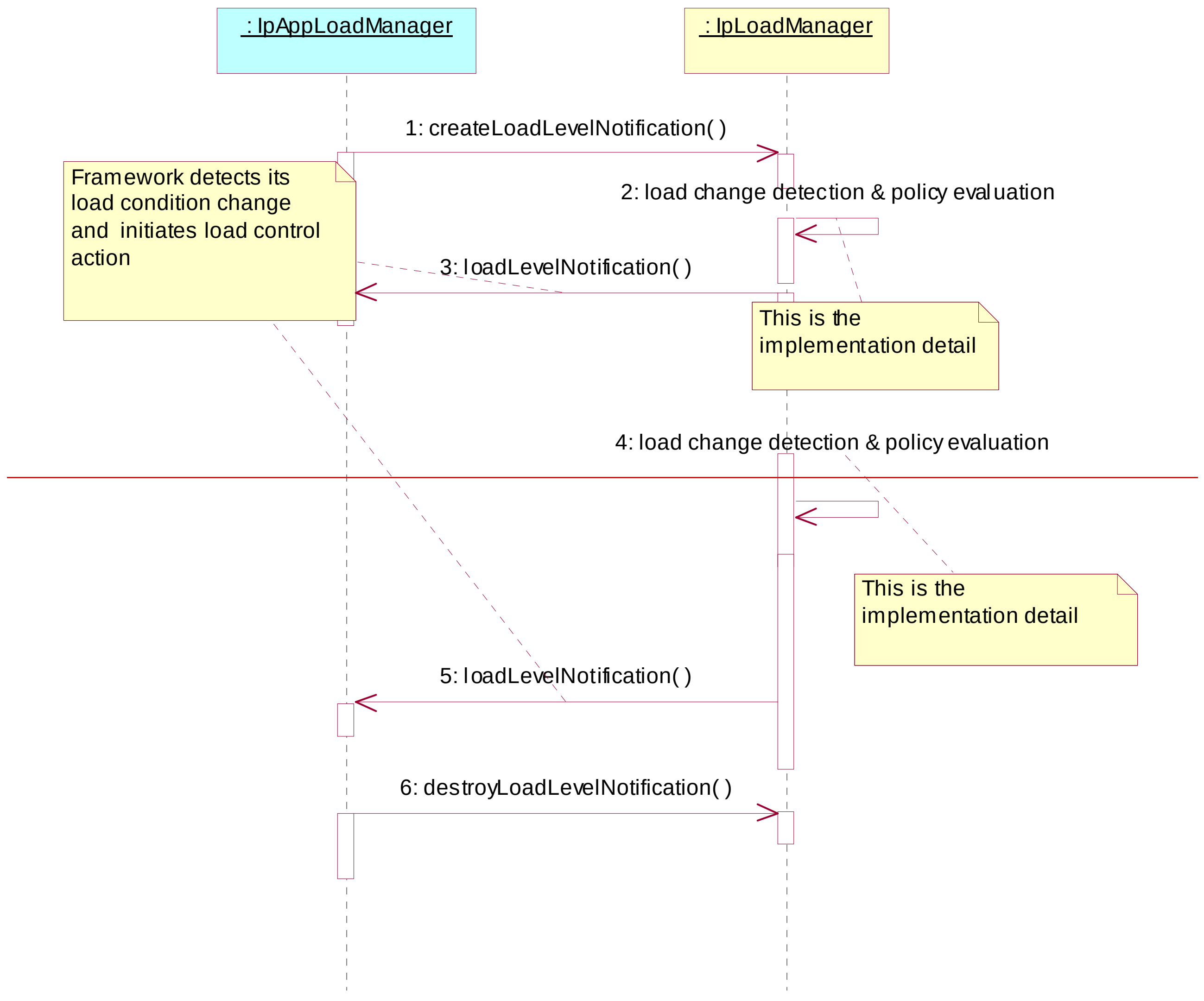

: IpAppLoadManager
: IpLoadManager
1: load change detection and policy evaluation
This is implementation detail
2: suspendNotification( )
Load balancing service makes a decision based on pre-defined policy
3: load change detection and policy evaluation
4: resumeNotification( )
Application provides initial load report on notification resumption
5: reportLoad( )

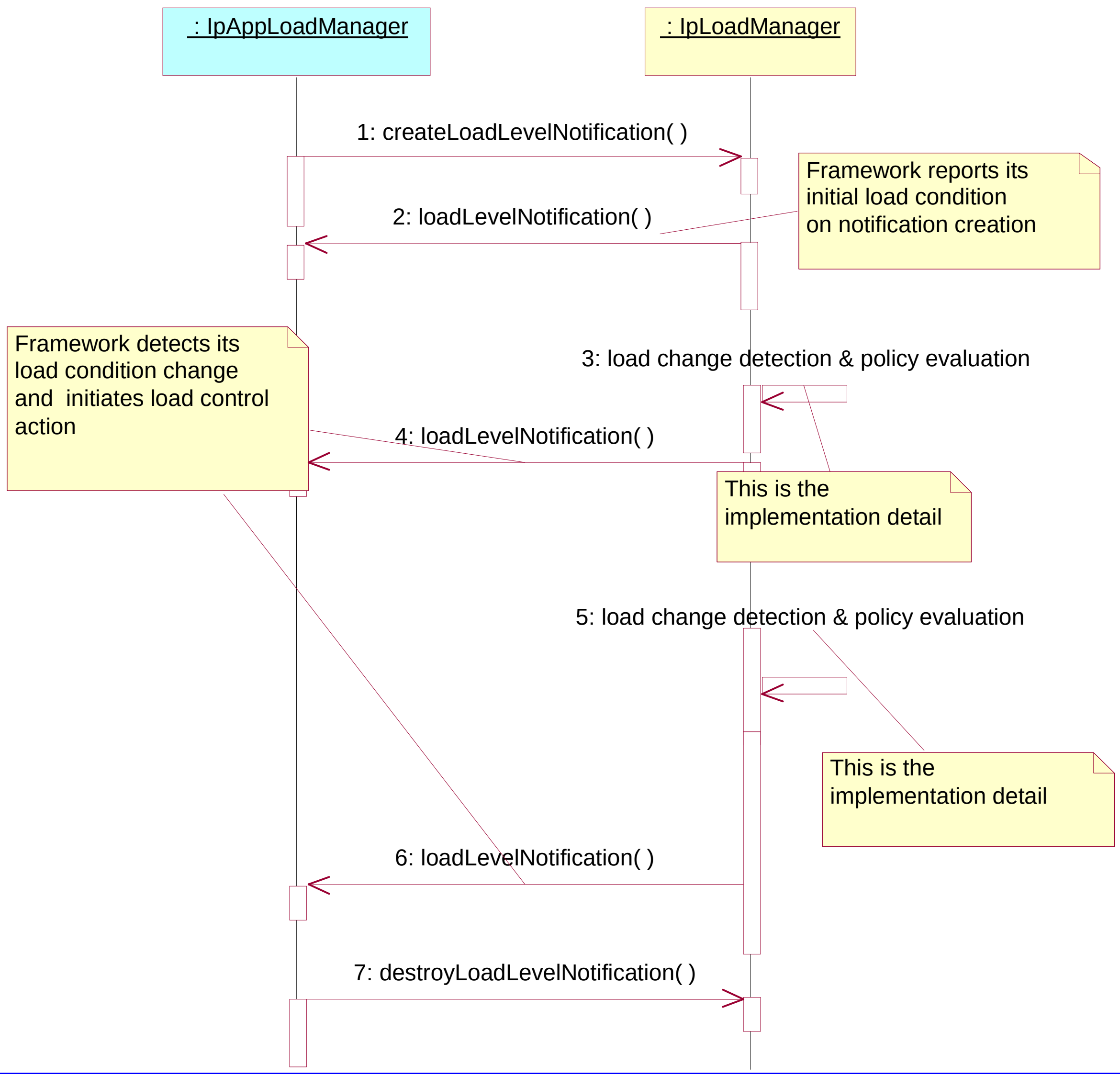
************ End of change # 1 *****************

************ Start of change # 2 *****************

7.1.2.5 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( )


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

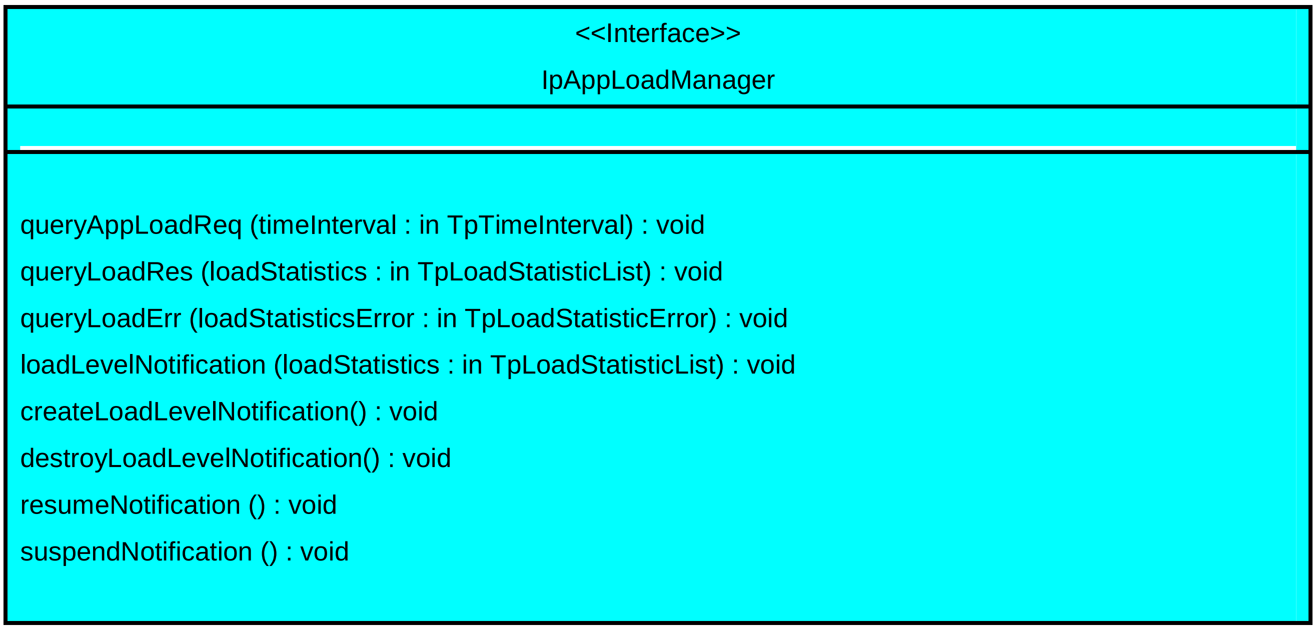
************ End of change # 2 *****************

************ Start of change # 3 *****************

7.3.3.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

The client application developer supplies the load manager application interface to handle requests, reports and other responses from the framework load manager function. 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() method on the IpAccess interface.



```
<<Interface>>
IpAppLoadManager

queryAppLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
createLoadLevelNotification() : void
destroyLoadLevelNotification() : void
resumeNotification () : void
suspendNotification () : void
```

Method
`queryAppLoadReq()`

The framework uses this method to request the application to provide load statistics records for the application.

Parameters

`timeInterval : in TpTimeInterval`

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

Method
`queryLoadRes()`

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

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the framework-supplied load statistics

Method
`queryLoadErr()`

The framework uses this method to return an error response to the application 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 IpLoadManager interface.

Parameters

`loadStatisticsError : in TpLoadStatisticError`

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

Method
`loadLevelNotification()`

Upon detecting load condition change, (e.g. load level changing from 0 to 1, 0 to 2, 1 to 0, for the SCFs or framework which have been registered for load level notifications) this method is invoked on the application. In addition this method shall be invoked on the application in order to provide a notification of current load status, when load notifications are first requested, or resumed after suspension.

Parameters

`loadStatistics : in TpLoadStatisticList`

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

Method
`createLoadLevelNotification()`

The framework uses this method to register to receive notifications of load level changes associated with the application. Upon receipt of this method the client application shall inform the framework of the current load using the reportLoad method on the corresponding IpLoadManager.

Parameters
No Parameters were identified for this method

Method
`destroyLoadLevelNotification()`

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

Parameters
No Parameters were identified for this method

Method
`resumeNotification()`

The framework uses this method to request the application to resume sending it notifications: e.g. after a period of suspension during which the framework handled a temporary overload condition. Upon receipt of this method the client application shall inform the framework of the current load using the reportLoad method on the corresponding IpLoadManager.

Parameters
No Parameters were identified for this method

Method
`suspendNotification()`

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

Parameters
No Parameters were identified for this method

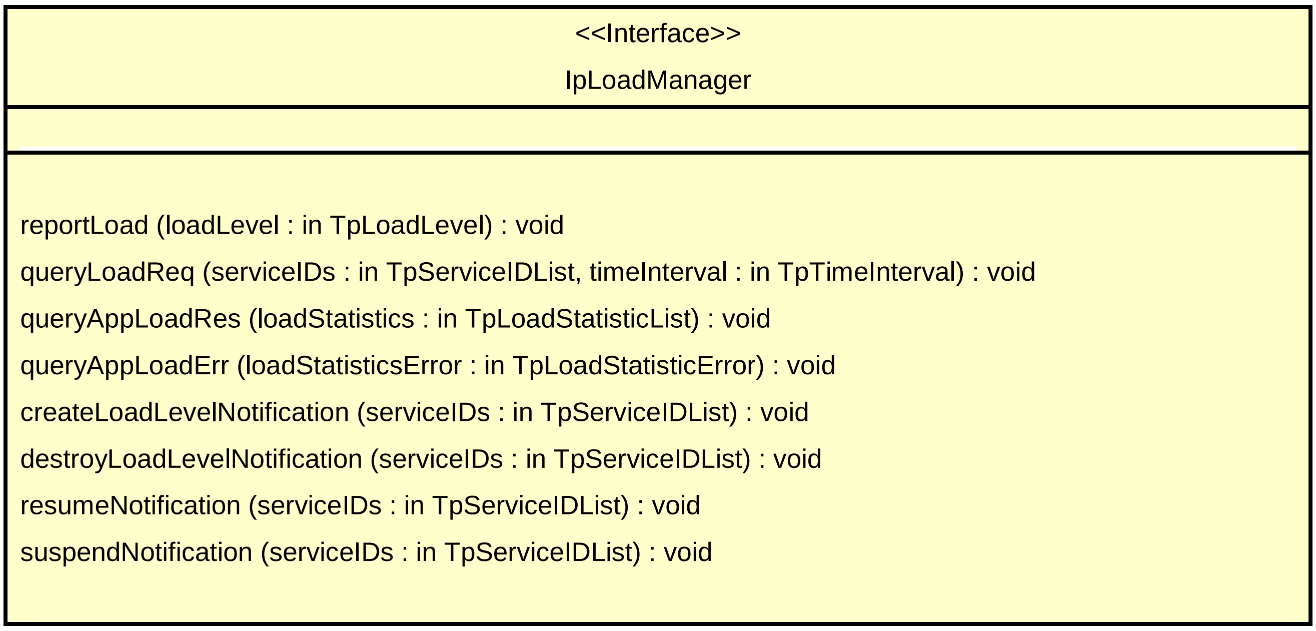
************ End of change # 3 *****************

************ Start of change # 4 *****************

7.3.3.8 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) : void
queryLoadReq (serviceIDs : in TpServiceIDList, timeInterval : in TpTimeInterval) : void
queryAppLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryAppLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
createLoadLevelNotification (serviceIDs : in TpServiceIDList) : void
destroyLoadLevelNotification (serviceIDs : in TpServiceIDList) : void
resumeNotification (serviceIDs : in TpServiceIDList) : void
suspendNotification (serviceIDs : in TpServiceIDList) : void

Method
`reportLoad()`

The client application uses this method to report its current load level (0,1, or 2) to the framework: e.g. when the load level on the application has changed. In addition this method shall be called by the application in order to report current load status, when load notifications are first requested, or resumed after suspension.

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

Parameters

`loadLevel : in TpLoadLevel`

Specifies the application's load level.

Raises

`TpCommonExceptions`

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.   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, 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, P_UNAUTHORISED_PARAMETER_VALUE`

Method
`queryAppLoadRes()`

The client application 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 queryAppLoadReq method on the IpAppLoadManager interface.

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the application-supplied load statistics.

Raises

TpCommonExceptions

Method
queryAppLoadErr()

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

Parameters

loadStatisticsError : in TpLoadStatisticError

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

Raises

TpCommonExceptions

Method
createLoadLevelNotification()

The client application uses this method to register to receive notifications of load level changes associated with either the framework or with its instances of the 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. Upon receipt of this method the framework shall inform the client application of the current framework or service instance load using the loadLevelNotification method on the corresponding IpAppLoadManager.

Parameters

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method
destroyLoadLevelNotification()

The client application uses this method to unregister for notifications of load level changes associated with either the framework or with its instances of the 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 level changes should no longer be reported. To unregister for framework load control, the serviceIDs parameter must be an empty list.

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 either the framework 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. 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. Upon receipt of this method the framework shall inform the client application of the current framework or service instance load using the loadLevelNotification method on the corresponding IpAppLoadManager.

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework 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.

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 either the framework or with its instances of the 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 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.

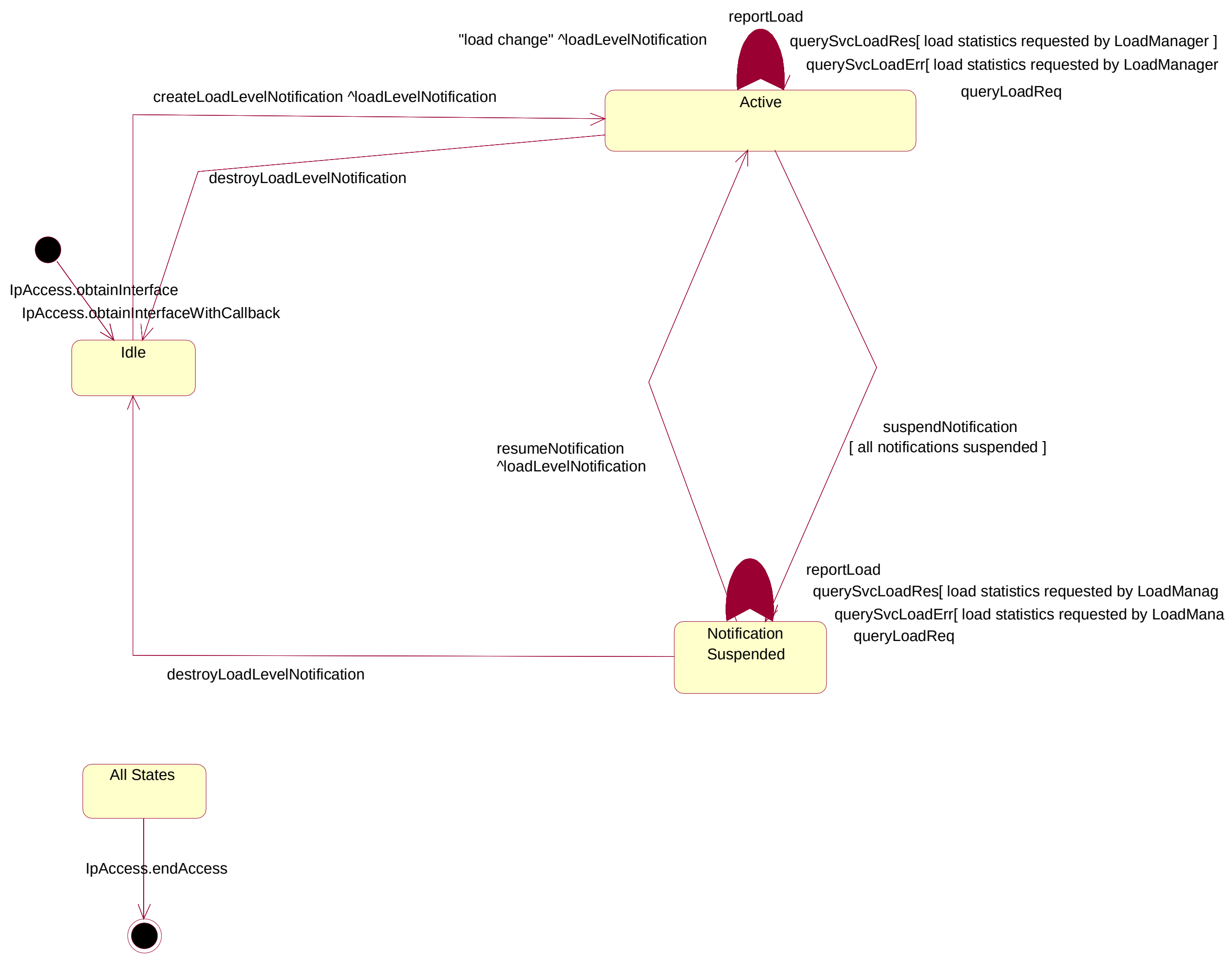
Raises

`TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE`

************ End of change # 4 *****************

************ Start of change # 5 *****************

7.4.3.1 State Transition Diagrams for IpLoadManager


reportLoad
"load change" ^loadLevelNotification
querySvcLoadRes[load statistics requested by LoadManager]
querySvcLoadErr[load statistics requested by LoadManager
queryLoadReq
createLoadLevelNotification ^loadLevelNotification
Active
destroyLoadLevelNotification
IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
suspendNotification
[all notifications suspended]
resumeNotification
^loadLevelNotification
reportLoad
querySvcLoadRes[load statistics requested by LoadManag
querySvcLoadErr[load statistics requested by LoadMana
queryLoadReq
Notification
Suspended
destroyLoadLevelNotification
All States
IpAccess.endAccess

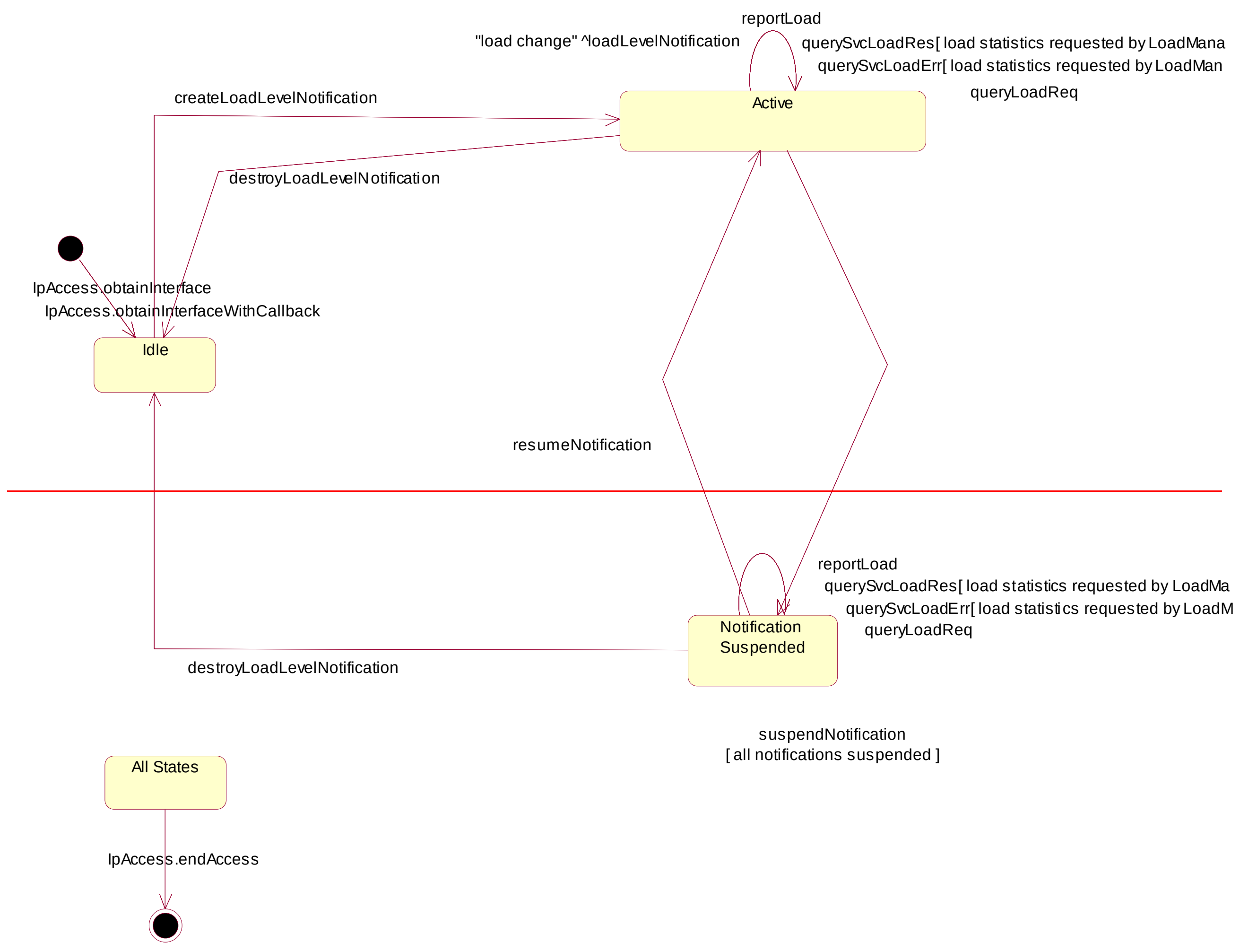

reportLoad
"load change" ^loadLevelNotification
querySvcLoadRes[load statistics requested by LoadMana
querySvcLoadErr[load statistics requested by LoadMan
queryLoadReq
createLoadLevelNotification
Active
destroyLoadLevelNotification
IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
resumeNotification
reportLoad
querySvcLoadRes[load statistics requested by LoadMa
querySvcLoadErr[load statistics requested by LoadM
queryLoadReq
Notification Suspended
destroyLoadLevelNotification
suspendNotification
[all notifications suspended]
All States
IpAccess.endAccess

Figure : State Transition Diagram for IpLoadManager

7.4.3.1.1 Idle State

In this state the application has obtained an interface reference of the LoadManager from the IpAccess interface.

7.4.3.1.2 Notification Suspended State

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

7.4.3.1.3 Active State

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().

7.4.3.2 State Transition Diagrams for LoadManagerInternal

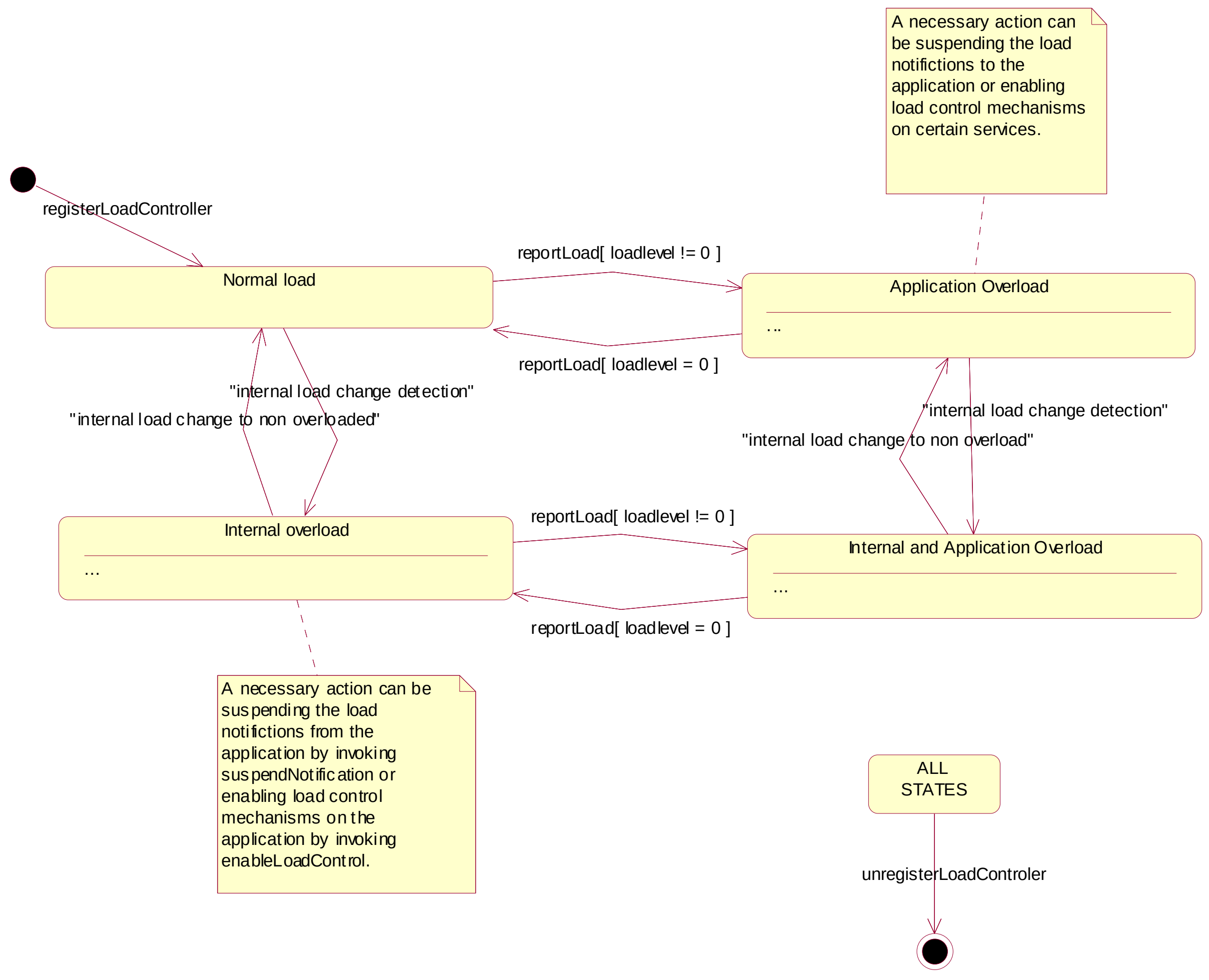

A necessary action can be suspending the load notifictions to the application or enabling load control mechanisms on certain services.
registerLoadController
Normal load
reportLoad[loadlevel != 0]
Application Overload
...
reportLoad[loadlevel = 0]
"internal load change detection"
"internal load change to non overloaded"
"internal load change detection"
"internal load change to non overload"
Internal overload
...
reportLoad[loadlevel != 0]
Internal and Application Overload
...
reportLoad[loadlevel = 0]
A necessary action can be suspending the load notifictions from the application by invoking suspendNotification or enabling load control mechanisms on the application by invoking enableLoadControl.
ALL STATES
unregisterLoadControler

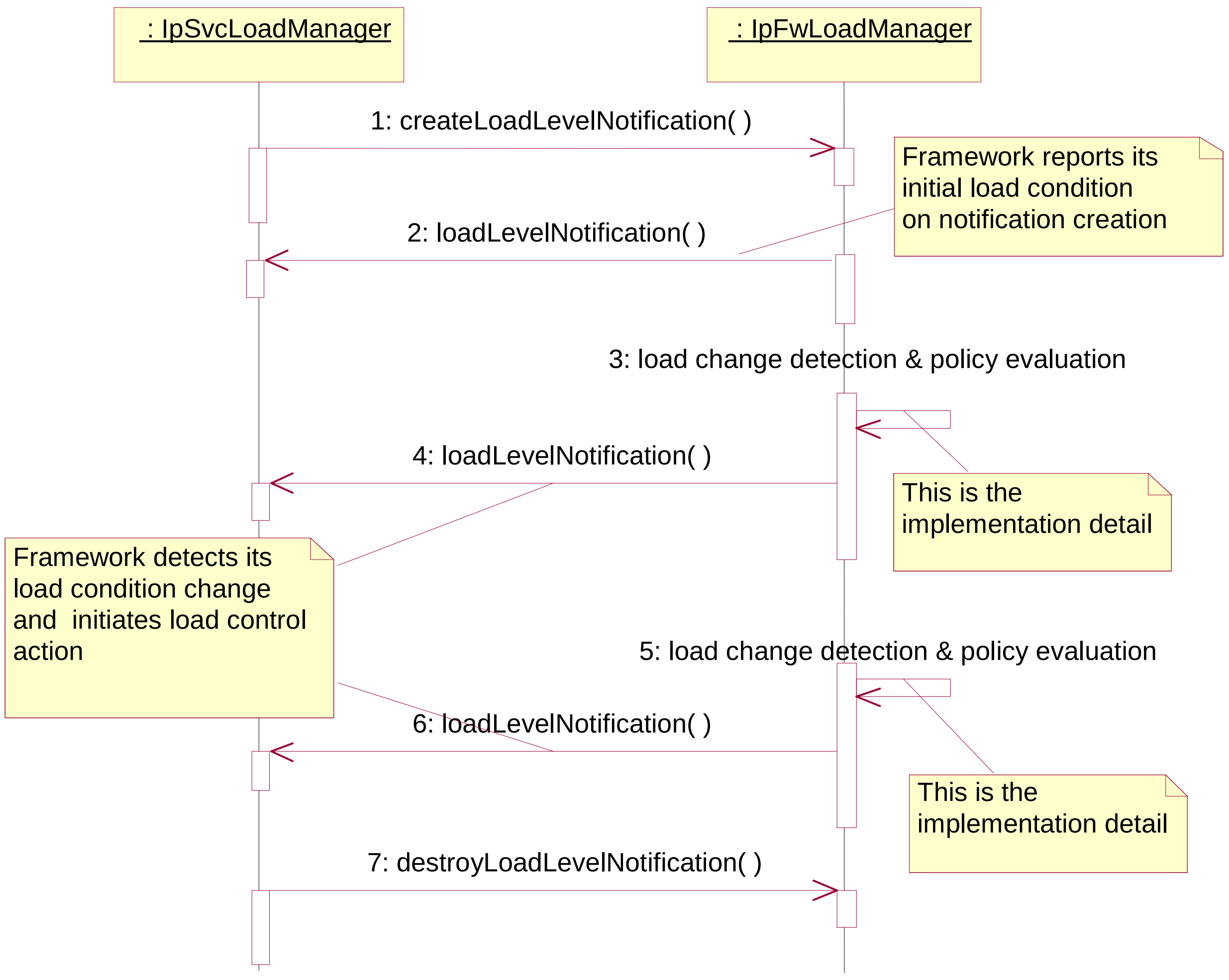
Figure : State Transition Diagram for LoadManagerInternal

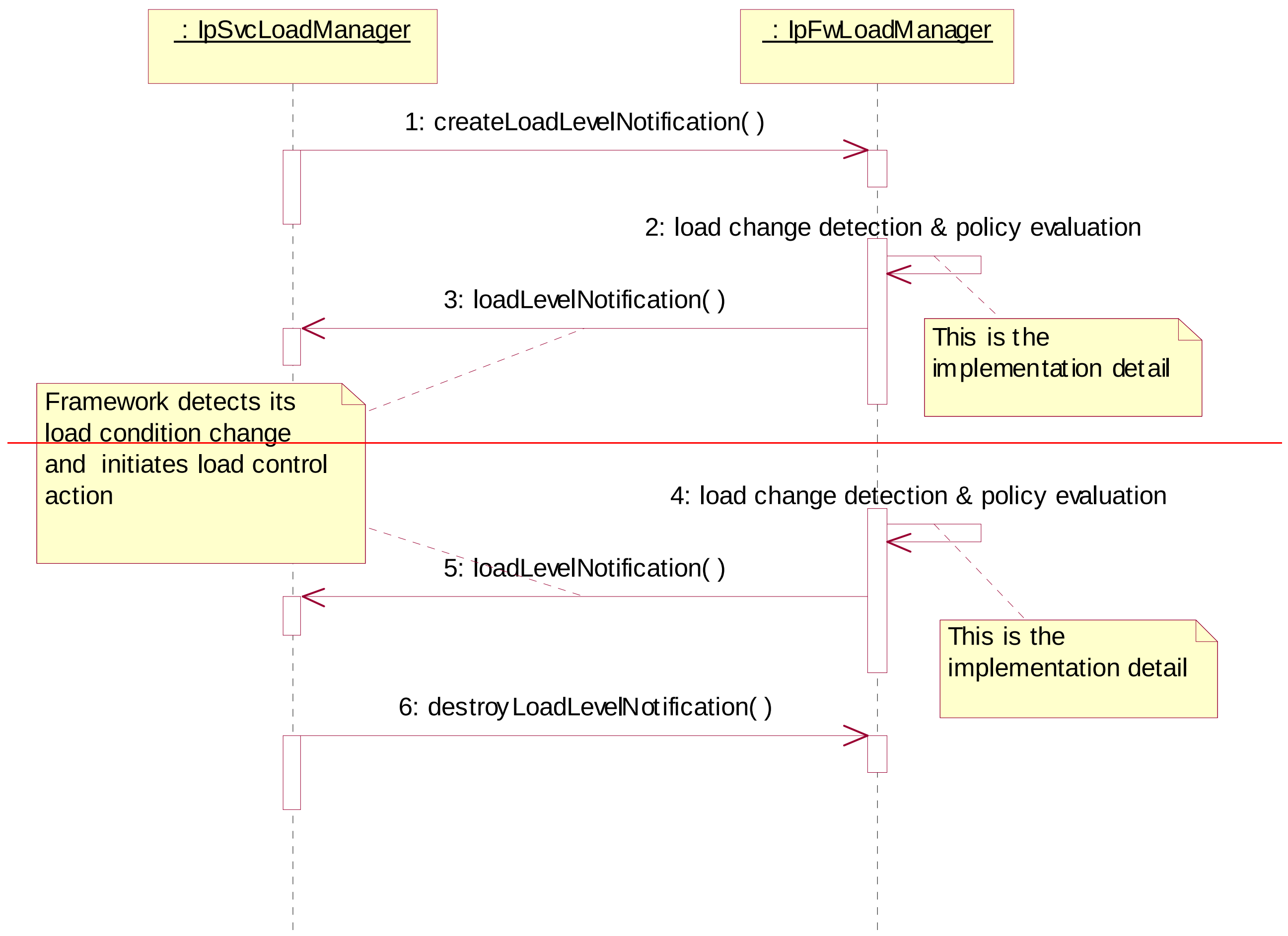
************ End of change # 5 *****************

************ Start of change # 6 *****************

8.1.4.1 Load Management: Service callback registration and load control

This sequence diagram shows how a service registers itself and the framework invokes load management function based on policy


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


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

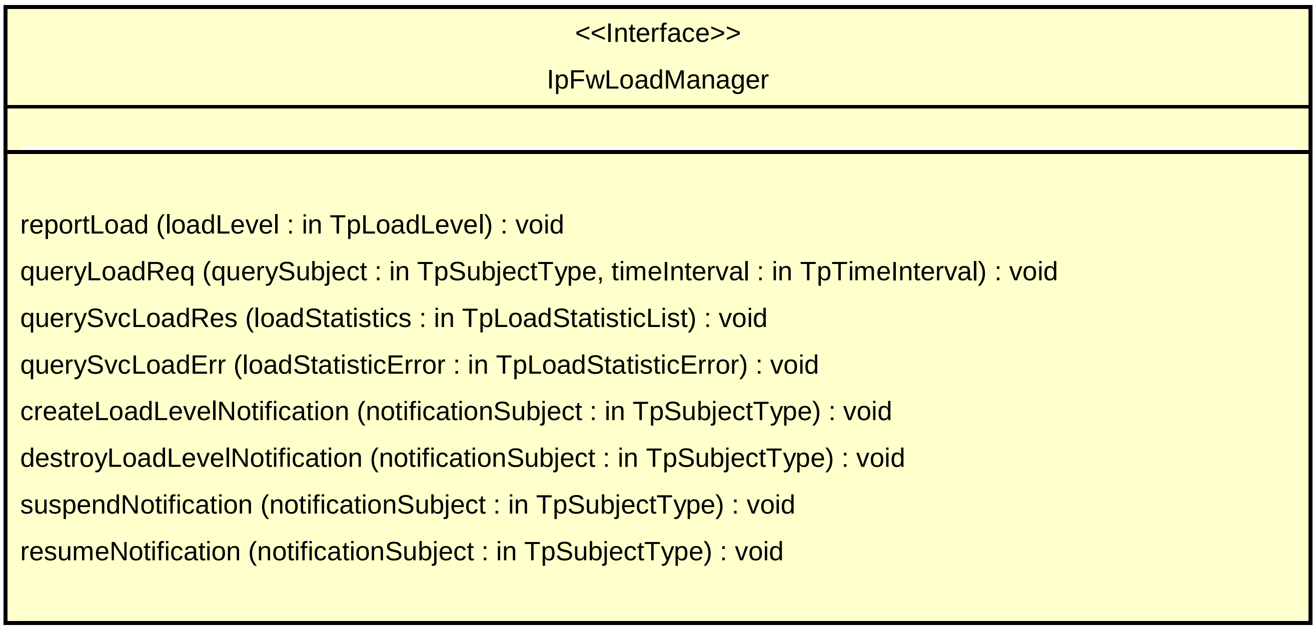
************ End of change # 6 *****************

************ Start of change # 7 *****************

8.3.4.7 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.



<<Interface>>

IpFwLoadManager

reportLoad (loadLevel : in TpLoadLevel) : void

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

querySvcLoadRes (loadStatistics : in TpLoadStatisticList) : void

querySvcLoadErr (loadStatisticError : in TpLoadStatisticError) : void

createLoadLevelNotification (notificationSubject : in TpSubjectType) : void

destroyLoadLevelNotification (notificationSubject : in TpSubjectType) : void

suspendNotification (notificationSubject : in TpSubjectType) : void

resumeNotification (notificationSubject : 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. In addition this method shall be called by the service instance in order to report current load status, when load notifications are first requested, or resumed after suspension.

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 application) 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 instance 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. Upon receipt of this method the framework shall inform the service instance of the current framework or application load using the loadLevelNotification method on the corresponding IpSvcLoadManager.

Parameters

`notificationSubject : in TpSubjectType`

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

Raises

`TpCommonExceptions`

Method
`destroyLoadLevelNotification()`

The service instance 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

`notificationSubject : 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

`notificationSubject : 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. Upon receipt of this method the framework shall inform the service instance of the current framework or application load using the loadLevelNotification method on the corresponding IpSvcLoadManager.

Parameters

`notificationSubject : 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`

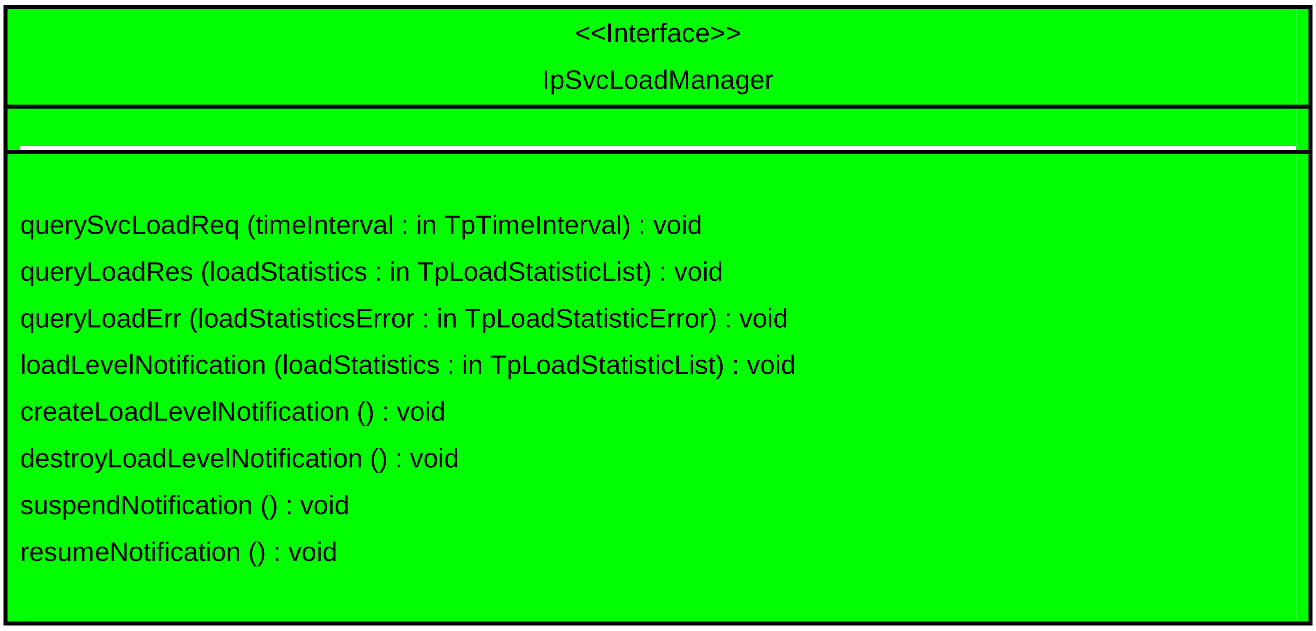
************ End of change # 7 *****************

************ Start of change # 8 *****************

8.3.4.8 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 IpAccess interface.



querySvcLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
createLoadLevelNotification () : void
destroyLoadLevelNotification () : 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 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 or framework which has been registered for load level notifications) this method is invoked on the SCF. In addition this method shall be invoked on the SCF in order to provide a notification of current load status, when load notifications are first requested, or resumed after suspension.

Parameters

`loadStatistics : in TpLoadStatisticList`

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

Raises

`TpCommonExceptions`

Method
`createLoadLevelNotification()`

The framework uses this method to register to receive notifications of load level changes associated with the service instance. Upon receipt of this method the service instance shall inform the framework of the current load using the reportLoad method on the corresponding IpFwLoadManager.

Parameters
No Parameters were identified for this method

Raises

`TpCommonExceptions`

Method
destroyLoadLevelNotification()

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

Parameters
No Parameters were identified for this method

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. Upon receipt of this method the service instance shall inform the framework of the current load using the reportLoad method on the corresponding IpFwLoadManager.

Parameters
No Parameters were identified for this method

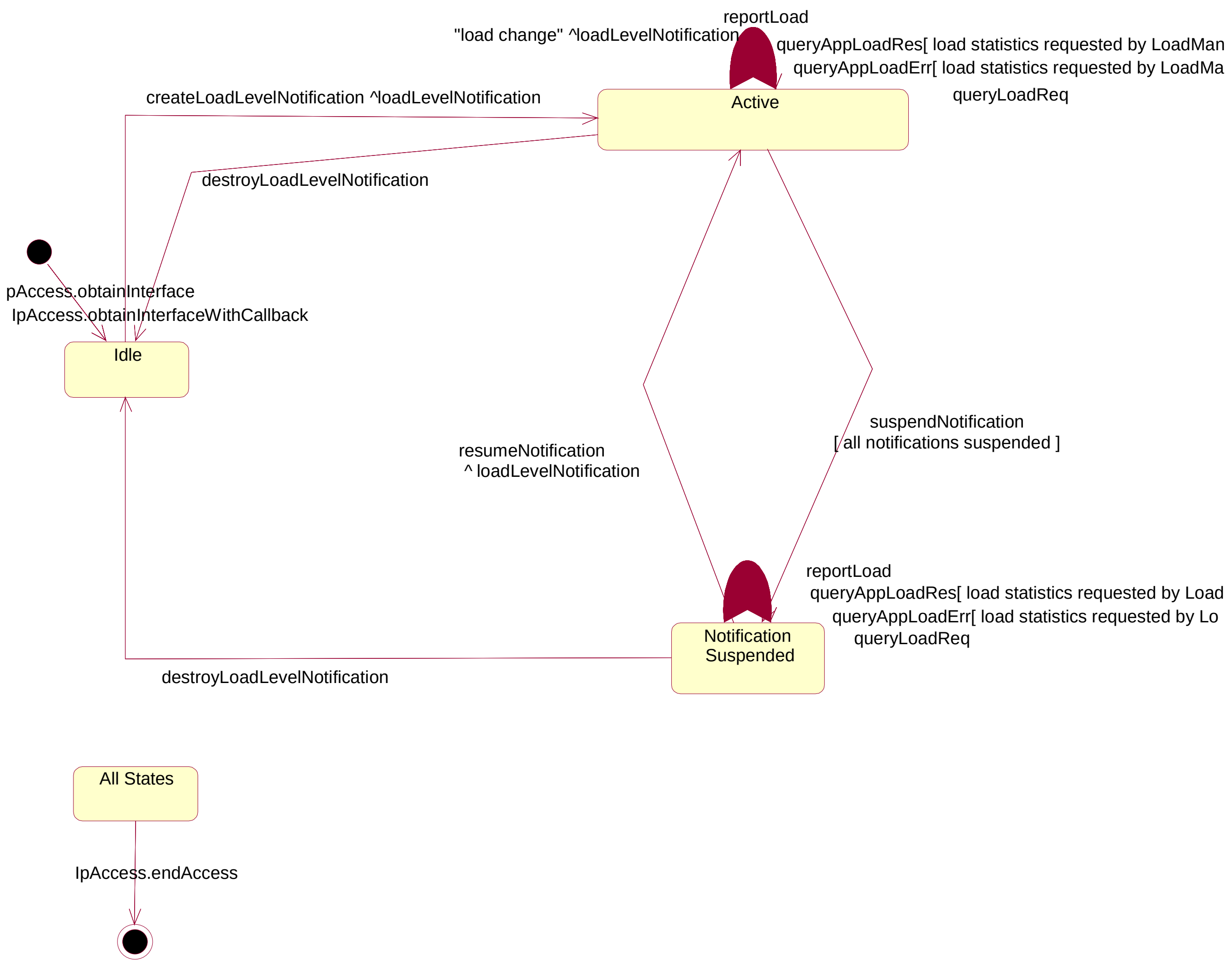
Raises

TpCommonExceptions

************ End of change # 8 *****************

************ Start of change # 9 *****************

8.4.4.1 State Transition Diagrams for IpFwLoadManager


reportLoad
"load change" ^loadLevelNotification
queryAppLoadRes[load statistics requested by LoadMan
queryAppLoadErr[load statistics requested by LoadMa
queryLoadReq
Active
createLoadLevelNotification ^loadLevelNotification
destroyLoadLevelNotification
pAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
suspendNotification
[all notifications suspended]
resumeNotification
^ loadLevelNotification
reportLoad
queryAppLoadRes[load statistics requested by Load
queryAppLoadErr[load statistics requested by Lo
queryLoadReq
Notification Suspended
destroyLoadLevelNotification
All States
IpAccess.endAccess

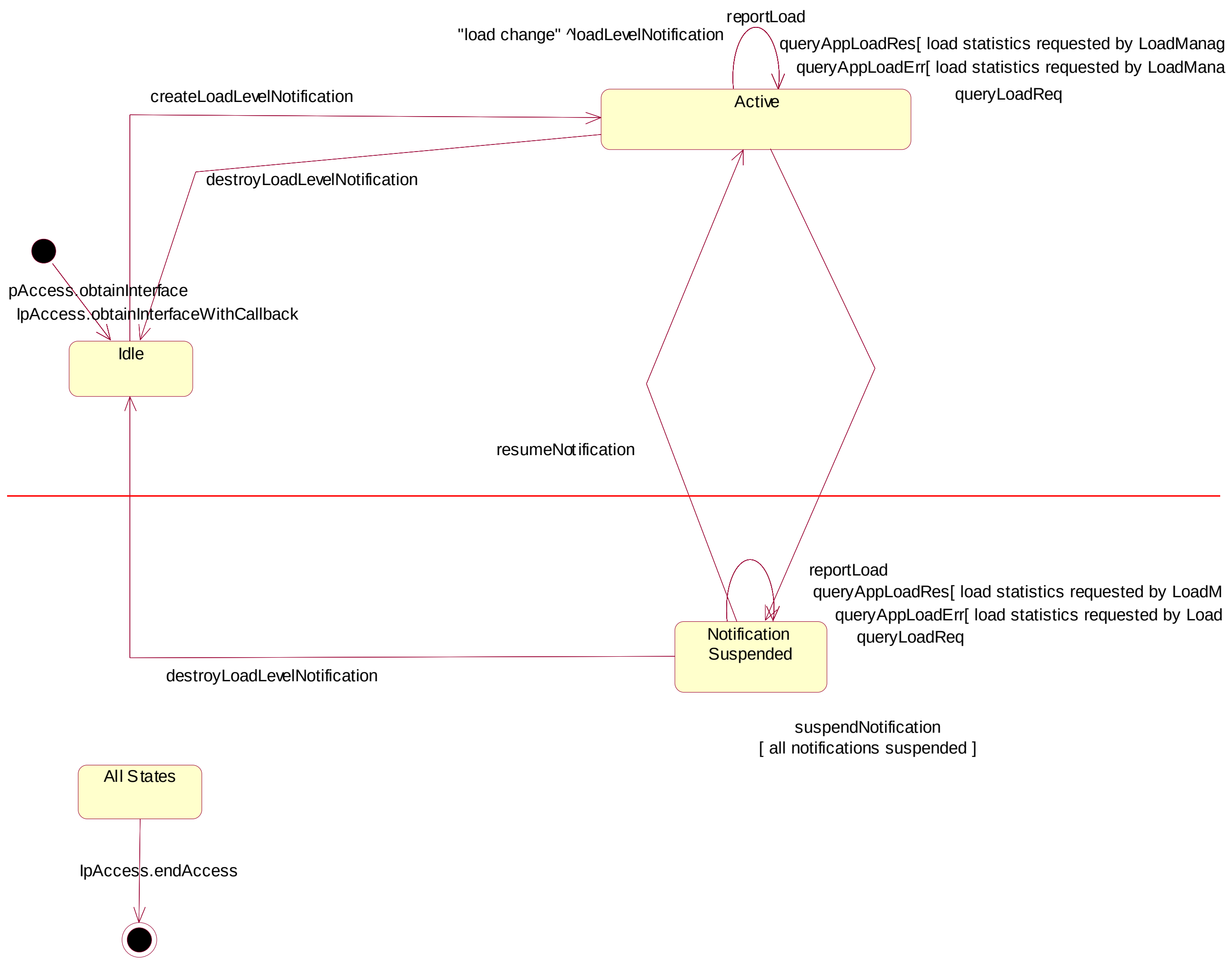

reportLoad
"load change" ^loadLevelNotification
queryAppLoadRes[load statistics requested by LoadManag
queryAppLoadErr[load statistics requested by LoadMana
queryLoadReq
createLoadLevelNotification
Active
destroyLoadLevelNotification
pAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Idle
resumeNotification
reportLoad
queryAppLoadRes[load statistics requested by LoadM
queryAppLoadErr[load statistics requested by Load
queryLoadReq
Notification Suspended
destroyLoadLevelNotification
suspendNotification
[all notifications suspended]
All States
IpAccess.endAccess

Figure : State Transition Diagram for IpFwLoadManager

************ Start of change # 9 *****************

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020894**
Meeting #20, Miami/ FL, USA, 23 – 27 September 2002

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**072**	⌘ **rev**	**-**	⌘	Current version:	**4.6.0** ⌘

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

Proposed change affects:	UICC apps⌘	☐	ME	☐	Radio Access Network	☐	Core Network	X

Title:	⌘	Correction of status of methods to interfaces in clause 6.3		
Source:	⌘	N5		
Work item code:	⌘	OSA1	***Date:*** ⌘	27/09/2002
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)* *Rel-6 (Release 6)*

Reason for change:	⌘	There is no requirement in the standard about the necessity to implement all or only some of the methods defined for an interface.
Summary of change:	⌘	Clarify which methods are mandatory and which are optional.
Consequences if not approved:	⌘	Application developers will not know which methods will actually be available.

Clauses affected:	⌘	6.3.1 Trust and Security Management Interface Classes				
		Y	**N**			
Other specs affected:	⌘		X	Other core specifications	⌘	
			X	Test specifications		
			X	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/specs/CR.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 Overview of the Framework

This clause 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 figure below). The description of the Framework in the present document separates the interfaces into two distinct sets: Framework to Application interfaces and Framework to Service interfaces.

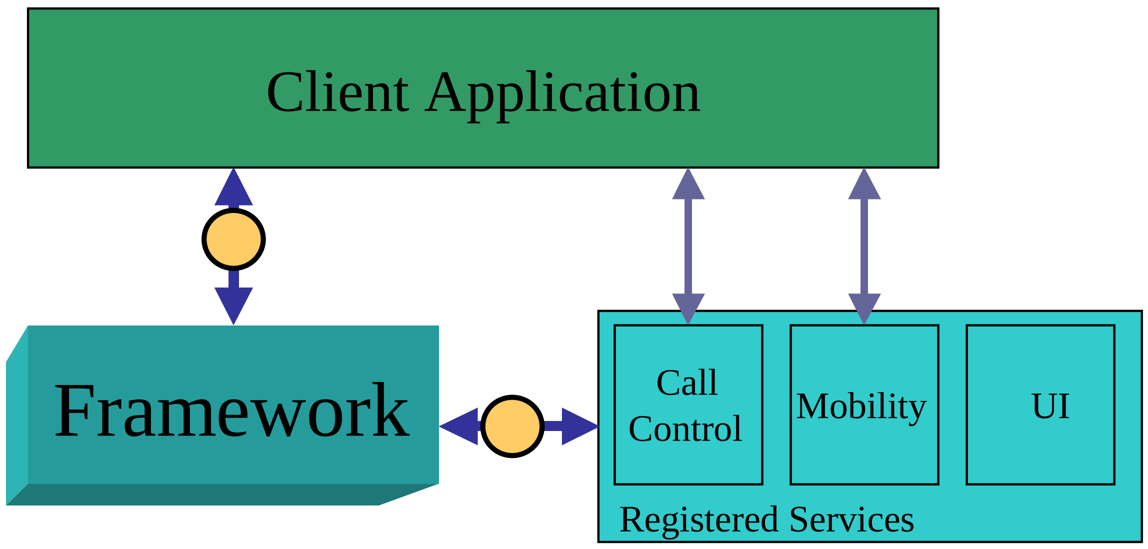


Figure:

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 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.

- **Authorisation:** Authorisation is distinguished from authentication in that authorisation is the action of determining what a previously authenticated application is allowed to do. Authentication shall precede authorisation. Once authenticated, an application is authorised to access certain SCFs.

- **Discovery of Framework and network SCFs:** After successful authentication, applications can obtain available Framework interfaces and use the discovery interface to obtain information on authorised network SCFs. The Discovery interface can be used at any time after successful authentication.

- **Establishment of service agreement:** Before any application can interact with a network SCF, a service agreement shall be established. A service agreement may consist of an off-line (e.g. by physically exchanging documents) and an on-line part. The application has to sign the on-line part of the service agreement before it is allowed to access any network SCF.

- **Access to network SCFs:** The Framework shall provide access control functions to authorise the access to SCFs or service data for any API method from an application, with the specified security level, context, domain, etc.

Basic mechanism between Framework and Service Capability Server (SCS):

- **Registering of network SCFs**. SCFs offered by a SCS can be registered at the Framework. In this way the Framework can inform the Applications upon request about available SCFs (Discovery). For example, this mechanism is applied when installing or upgrading an SCS.

The following clauses describe each aspect of the Framework in the following order:

- The *sequence diagrams* give the reader a practical idea of how the Framework is implemented.
- The *class diagrams* clause shows how each of the interfaces applicable to the Framework relate to one another.
- The *interface specification* clause describes in detail each of the interfaces shown within the class diagram part.
- The *State Transition Diagrams (STD)* show the transition between states in the Framework. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.
- The *data definitions* clause shows a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the common data types part of the present document (29.198-2).

4.1 General requirements on support of methods

An implementation of this API which supports or implements a method described in the present document, shall support or implement the functionality described for that method, for at least one valid set of values for the parameters of that method.

Where a method is not supported by an implementation of a Framework or Service interface, the exception P_METHOD_NOT_SUPPORTED shall be returned to any call of that method.

Where a method is not supported by an implementation of an Application interface, a call to that method shall be possible, and no exception shall be returned.

6.3 Interface Classes

6.3.1 Trust and Security Management Interface Classes

The Trust and Security Management Interfaces provide:

- the first point of contact for a client to access a Framework provider;
- the authentication methods for the client and Framework provider to perform an authentication protocol;
- the client with the ability to select a service capability feature to make use of;
- the client with a portal to access other Framework interfaces.

The process by which the client accesses the Framework provider has been separated into 3 stages, each supported by a different Framework interface:

1) Initial Contact with the Framework;
2) Authentication to the Framework;
3) Access to Framework and Service Capability Features.

6.3.1.1 Interface Class IpClientAPILevelAuthentication

Inherits from: IpInterface.

If the IpClientAPILevelAuthentication interface is implemented by a client, ~~all methods~~ authenticate(), abortAuthentication() and authenticationSucceeded() methods shall be implemented.

<<Interface>> IpClientAPILevelAuthentication
authenticate (challenge : in TpOctetSet) : TpOctetSet abortAuthentication () : void authenticationSucceeded () : void

Method
`authenticate()`

This method is used by the framework to authenticate the client. The challenge will be encrypted using the mechanism prescribed by selectEncryptionMethod. The client 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 client on the IpAPILevelAuthentication interface.

Returns <response> : 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().

Parameters

`challenge : in TpOctetSet`

The challenge presented by the framework to be responded to by the client. 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

`TpOctetSet`

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 client responded incorrectly to a challenge in which case no further communication with the client 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 has been properly authenticated.

Parameters

No Parameters were identified for this method

Method

`authenticationSucceeded()`

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

Parameters

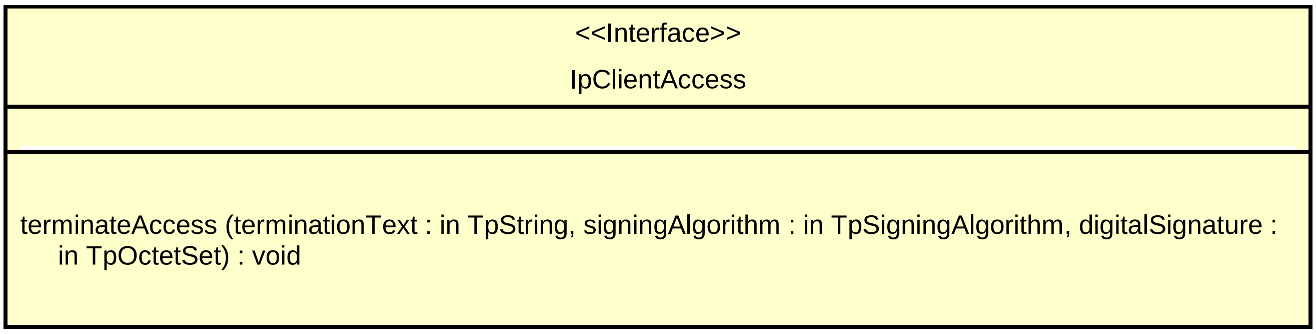
No Parameters were identified for this method

6.3.1.2 Interface Class IpClientAccess

Inherits from: IpInterface.

IpClientAccess interface is offered by the client to the framework to allow it to initiate interactions during the access session.

This interface and ~~its~~the terminateAccess() method shall be implemented by a client.


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

Method
`terminateAccess()`

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

After terminateAccess() is invoked, the client will no longer be authenticated with the framework. The client 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, and should take steps to terminate the client's access session WITHOUT invoking terminateAccess() on the client. This follows a generally accepted security model where the framework has decided that it can no longer trust the client 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, the P_INVALID_SIGNING_ALGORITHM exception will be thrown.

`digitalSignature : in TpOctetSet`

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

Raises

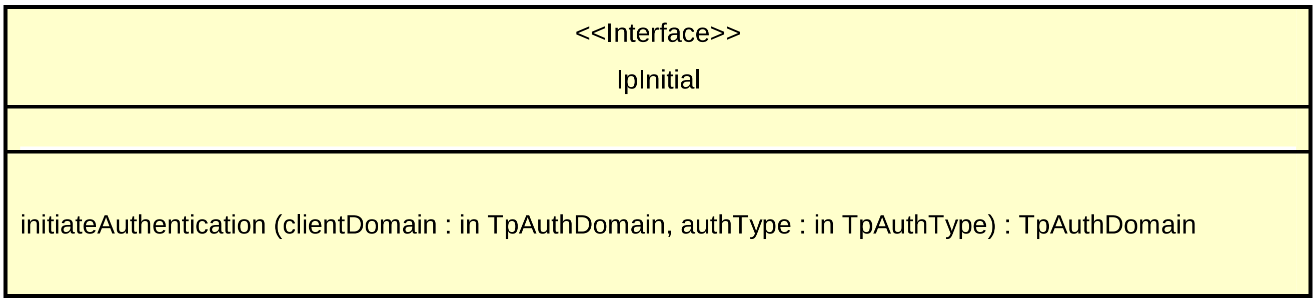
`TpCommonExceptions, P_INVALID_SIGNING_ALGORITHM, P_INVALID_SIGNATURE`

6.3.1.3 Interface Class IpInitial

Inherits from: IpInterface.

The Initial Framework interface is used by the client to initiate the mutual authentication with the Framework.

This interface and ~~its~~the initiateAuthentication() method shall be implemented by a client.


<<Interface>>
IpInitial
initiateAuthentication (clientDomain : in TpAuthDomain, authType : in TpAuthType) : TpAuthDomain

Method
`initiateAuthentication()`

This method is invoked by the client to start the process of mutual authentication with the framework, and request the use of a specific authentication method.

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

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

The domainID parameter is an identifier for the framework (i.e. TpFwID). It is used to identify the framework to the client.

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 client uses this interface to authenticate with the framework.

Parameters

clientDomain : in TpAuthDomain

This identifies the client domain to the framework, and provides a reference to the domain's authentication interface.

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

The domainID parameter is an identifier either for a client application (i.e. TpClientAppID) or for an enterprise operator (i.e. TpEntOpID), or for an existing registered service (i.e. TpServiceID) or for a service supplier (i.e. TpServiceSupplierID). It is used to identify the client domain to the framework, (see authenticate() on IpAPILevelAuthentication). If the framework does not recognise the domainID, the framework returns an error code (P_INVALID_DOMAIN_ID).

The authInterface parameter is a reference to call the authentication interface of the client. The type of this interface is defined by the authType parameter. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

authType : in TpAuthType

This identifies the type of authentication mechanism requested by the client. It provides operators and clients with the opportunity to use an alternative to the API level Authentication interface, e.g. an implementation specific authentication mechanism like CORBA Security, using the IpAuthentication interface, or Operator specific Authentication interfaces. OSA API level Authentication is the default authentication mechanism (P_OSA_AUTHENTICATION). If P_OSA_AUTHENTICATION is selected, then the clientDomain and fwDomain authInterface parameters are references to interfaces of type Ip(Client)APILevelAuthentication. If P_AUTHENTICATION is selected, the fwDomain authInterface parameter references to interfaces of type IpAuthentication which is used when an underlying distribution technology authentication mechanism is used.

Returns

TpAuthDomain

Raises

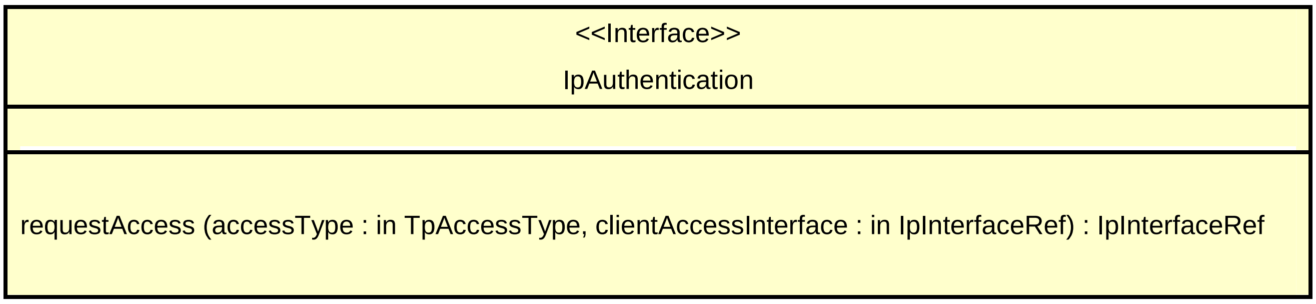
TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE

6.3.1.4 Interface Class IpAuthentication

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.

One of IpAuthentication or IpAPILevelAuthentication interfaces shall be implemented by a Framework. The requestAccess() method shall be implemented in each.


<<Interface>>
IpAuthentication
requestAccess (accessType : in TpAccessType, clientAccessInterface : in IpInterfaceRef) : IpInterfaceRef

Method
requestAccess()

Once client and framework are authenticated, the client invokes the requestAccess operation on the IpAuthentication or IpAPILevelAuthentication interface. This allows the client to request the type of access they require. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Operators can define their own access interfaces to satisfy client requirements for different types of access.)

If this method is called before the client and framework have successfully completed the authentication process, then the request fails, and an error code (P_ACCESS_DENIED) is returned.

Returns <fwAccessInterface> : This provides the reference for the client to call the access interface of the framework.

Parameters

accessType : in TpAccessType

This identifies the type of access interface requested by the client. If the framework does not provide the type of access identified by accessType, then an error code (P_INVALID_ACCESS_TYPE) is returned.

clientAccessInterface : in IpInterfaceRef

This provides the reference for the framework to call the access interface of the client. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

Returns

IpInterfaceRef

Raises

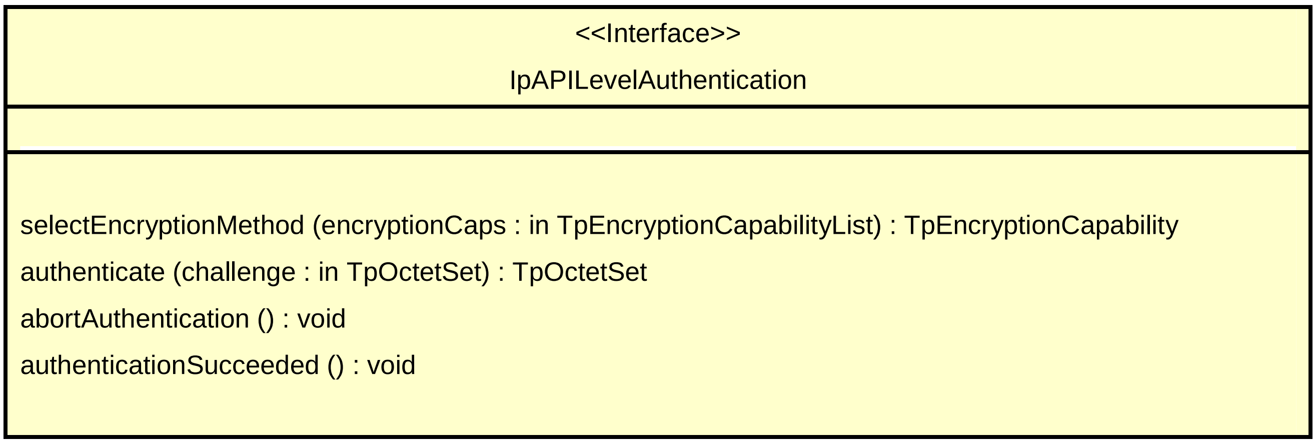
TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_ACCESS_TYPE, P_INVALID_INTERFACE_TYPE

6.3.1.5 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

The API Level Authentication Framework interface is used by client 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.

If the IpAPILevelAuthentication interface is implemented by a Framework, ~~all~~the ~~methods~~ selectEncryptionMethod(), authenticate(), abortAuthentication() and authenticationSucceeded() methods shall be implemented. IpAPILevelAuthentication inherits the requirements of IpAuthentication, therefore requestAccess() shall be implemented.


<<Interface>>

IpAPILevelAuthentication

selectEncryptionMethod (encryptionCaps : in TpEncryptionCapabilityList) : TpEncryptionCapability

authenticate (challenge : in TpOctetSet) : TpOctetSet

abortAuthentication () : void

authenticationSucceeded () : void

Method
selectEncryptionMethod()

The client uses this method to initiate the authentication process. The framework returns its preferred mechanism. This should be within capability of the client. If a mechanism that is acceptable to the framework within the capability of the client cannot be found, the framework 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 client's authenticate() method (the wait is to ensure that the client can initialise any resources necessary to use the prescribed encryption method).

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

Parameters

encryptionCaps : in TpEncryptionCapabilityList

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

Returns

TpEncryptionCapability

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY

Method
authenticate()

This method is used by the client 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 client. The domainID received in the initiateAuthentication() can be used by the framework to reference the correct public key for the client (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.

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

Parameters

`challenge : in TpOctetSet`

The challenge presented by the client 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

`TpOctetSet`

Raises

`TpCommonExceptions, P_ACCESS_DENIED`

Method

`abortAuthentication()`

The client uses this method to abort the authentication process. This method is invoked if the client no longer wishes to continue the authentication process, (unless the client responded incorrectly to a challenge in which case no further communication with the client 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 has been properly authenticated.

Parameters
No Parameters were identified for this method

Raises

`TpCommonExceptions,P_ACCESS_DENIED`

Method

`authenticationSucceeded()`

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

Parameters
No Parameters were identified for this method

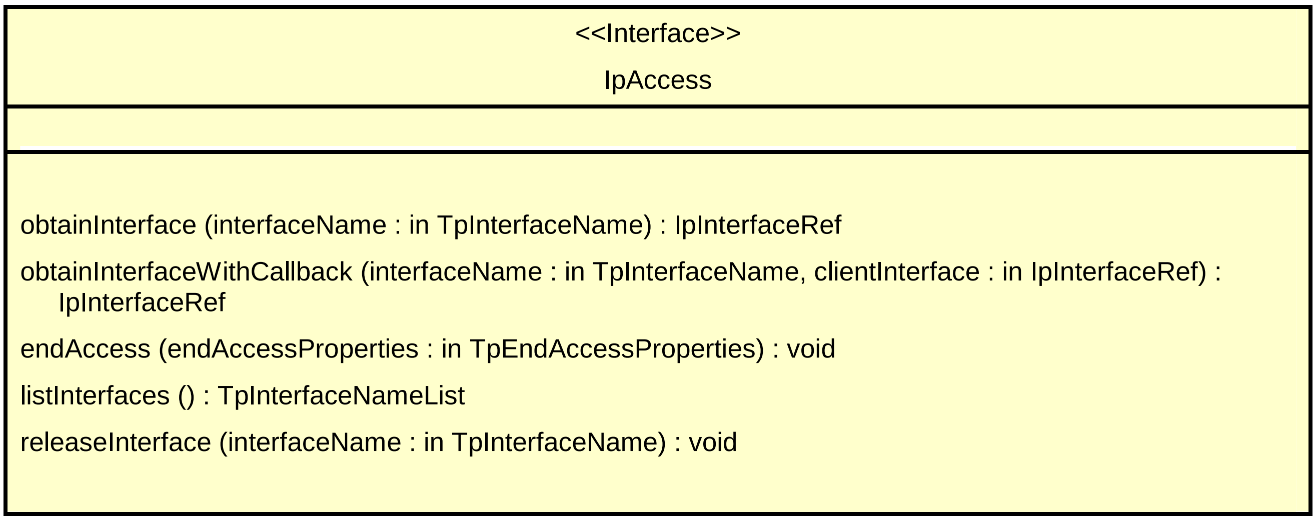
Raises

`TpCommonExceptions, P_ACCESS_DENIED`

6.3.1.6 Interface Class IpAccess

Inherits from: IpInterface.

This interface shall be implemented by a Framework. As a minimum requirement the obtainInterface(), ~~and~~ obtainInterfaceWithCallback() and endAccess() methods shall be implemented.


<<Interface>>
IpAccess
obtainInterface (interfaceName : in TpInterfaceName) : IpInterfaceRef
obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, clientInterface : in IpInterfaceRef) : IpInterfaceRef
endAccess (endAccessProperties : in TpEndAccessProperties) : void
listInterfaces () : TpInterfaceNameList
releaseInterface (interfaceName : in TpInterfaceName) : void

Method
obtainInterface()

This method is used to obtain other framework interfaces. The client uses this method to obtain interface references to other framework interfaces. (The obtainInterfaceWithCallback method should be used if the client 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 to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

Returns

IpInterfaceRef

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME

Method
obtainInterfaceWithCallback()

This method is used to obtain other framework interfaces. The client 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 to which a reference to the interface is requested. If the interfaceName is invalid, the framework returns an error code (P_INVALID_INTERFACE_NAME).

`clientInterface : in IpInterfaceRef`

This is the reference to the client interface, which is used for callbacks. If a client 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).

Returns

`IpInterfaceRef`

Raises

`TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME, P_INVALID_INTERFACE_TYPE`

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 will no longer be authenticated with the framework. The client 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`

Method

listInterfaces()

The client 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().

Returns <frameworkInterfaces> : The frameworkInterfaces parameter contains a list of interfaces that the framework makes available.

Parameters

No Parameters were identified for this method

Returns

`TpInterfaceNameList`

Raises

`TpCommonExceptions, P_ACCESS_DENIED`

Method
`releaseInterface()`

The client 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 during this access session, then the P_TASK_REFUSED exception will be thrown.

Raises

`TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME`

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-021087**
Meeting #21, Dublin, IRELAND, 28 – 31 October 2002

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-03 CR 067** ⌘ **rev - ⌘** Current version: **5.1.0** ⌘

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

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Correction of Sequence for Framework – Service load management
Source:	⌘	N5
Work item code:	⌘	OSA2
Date:	⌘	31/10/2002
Category:	⌘	**A**
Release:	⌘	*REL-5*

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

Use one of the following releases:

2	*(GSM Phase 2)*
R96	*(Release 1996)*
R97	*(Release 1997)*
R98	*(Release 1998)*
R99	*(Release 1999)*
Rel-4	*(Release 4)*
Rel-5	*(Release 5)*
Rel-6	*(Release 6)*

Reason for change:	⌘	Sequence 8.1.4.2 includes incorrect text that directly contradicts the functionality and description of the suspendNotification load management method
Summary of change:	⌘	Correct the sequence diagram by removing the suspendNotification message flow
Consequences if not approved:	⌘	Contradictory use of the suspendNotification method may be assumed by developers resulting in incorrect implemenation and interoperability problems

Clauses affected:	⌘	8.1.4.2				
		Y	**N**			
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/specs/CR.htm. Below is a brief summary:

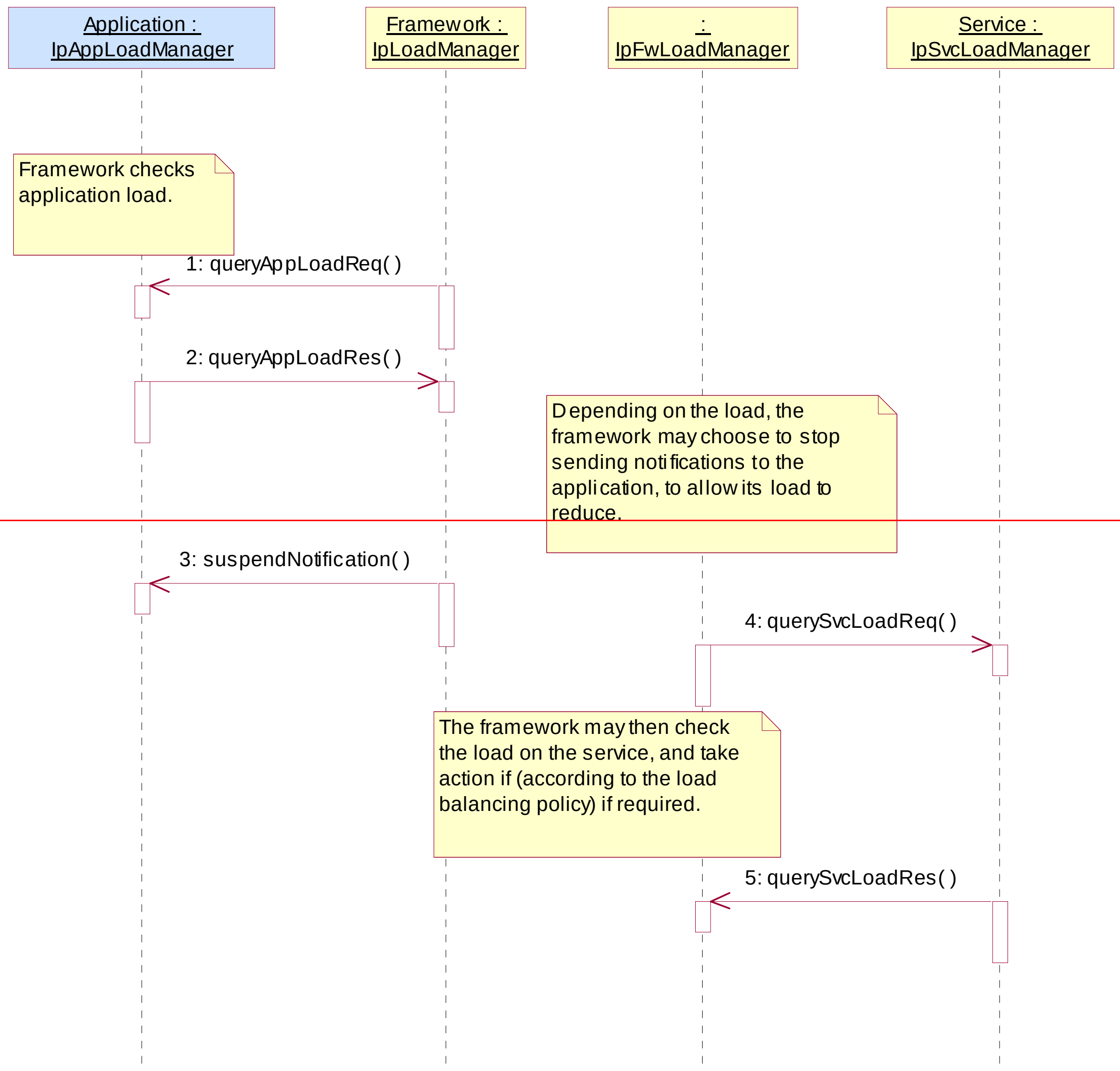
1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

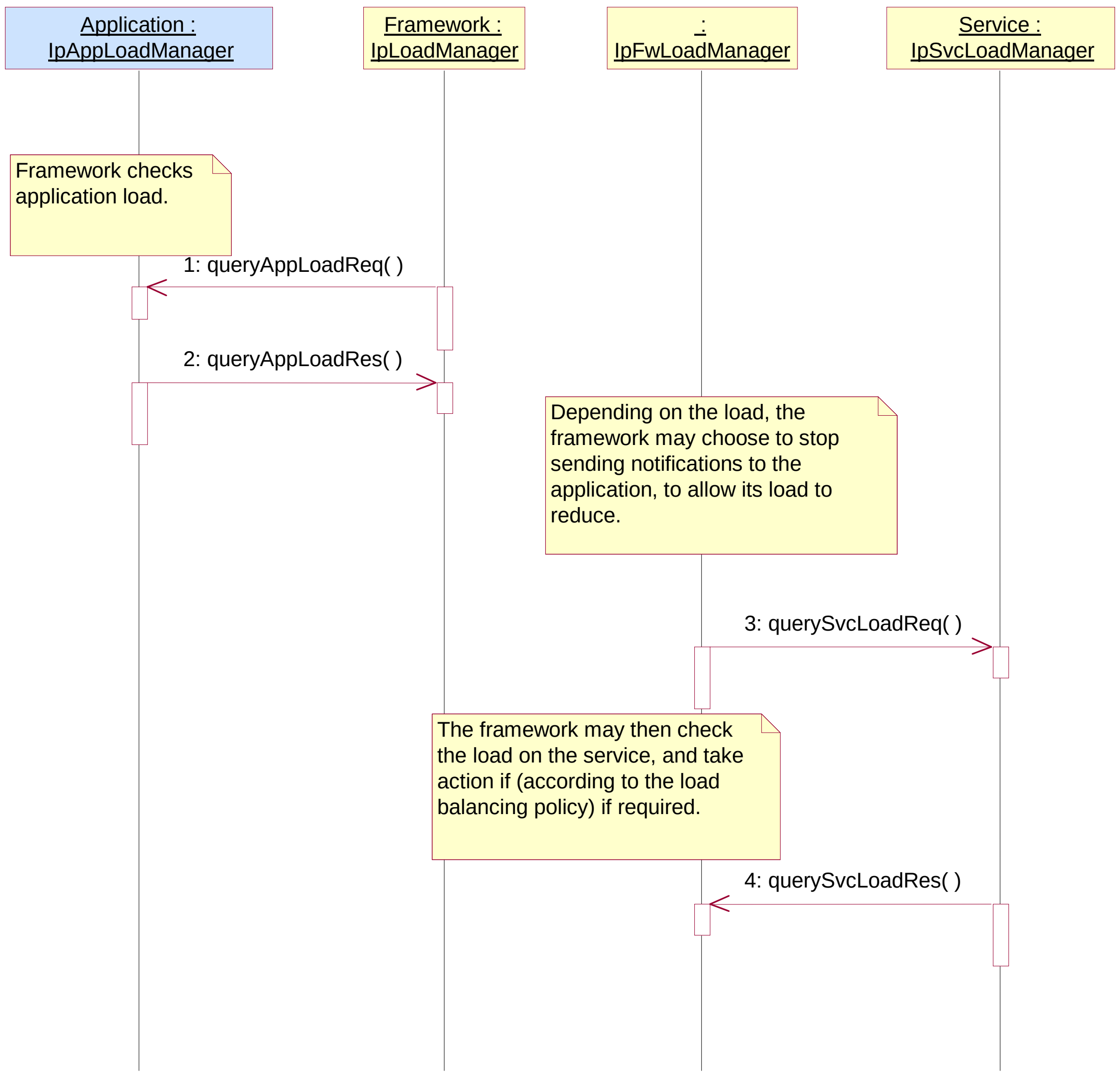
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest 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.

************** Change #1: Sequence Diagram Svc-Fw interface ****************************

8.1.4.2 Load Management: Client and Service Load Balancing


Application : IpAppLoadManager
Framework : IpLoadManager
: IpFwLoadManager
Service : IpSvcLoadManager
Framework checks application load.
1: queryAppLoadReq( )
2: queryAppLoadRes( )
Depending on the load, the framework may choose 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( )


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

**************** End of Change #1: Sequence Diagram Svc-Fw interface ******************************

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-021086**
Meeting #21, Dublin, IRELAND, 28 – 31 October 2002

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-03 CR 066** ⌘**rev** **-** ⌘ Current version: **4.6.0** ⌘

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

Proposed change affects:	UICC apps⌘		ME		Radio Access Network		Core Network **X**

Title:	⌘	Correction of Sequence for Framework – Service load management
Source:	⌘	N5
Work item code:	⌘	OSA1
Date:	⌘	31/10/2002
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)*
Rel-6	*(Release 6)*

Reason for change:	⌘	Sequence 8.1.4.2 includes incorrect text that directly contradicts the functionality and description of the suspendNotification load management method
Summary of change:	⌘	Correct the sequence diagram by removing the suspendNotification message flow
Consequences if not approved:	⌘	Contradictory use of the suspendNotification method may be assumed by developers resulting in incorrect implemenation and interoperability problems

Clauses affected:	⌘	8.1.4.2				
		Y	**N**			
Other specs affected:	⌘		**X**	Other core specifications	⌘	
			X	Test specifications		
			X	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/specs/CR.htm. Below is a brief summary:

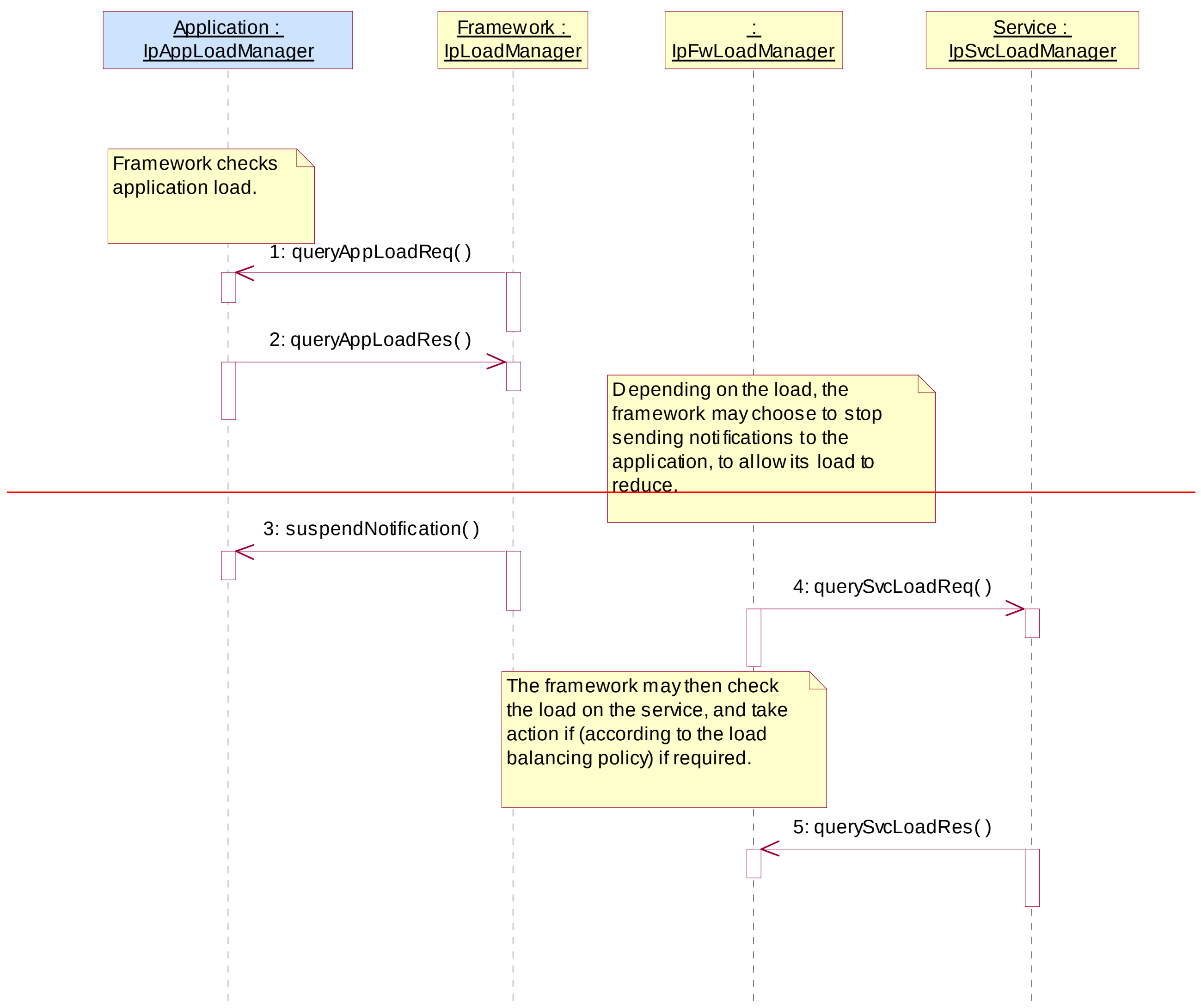
1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

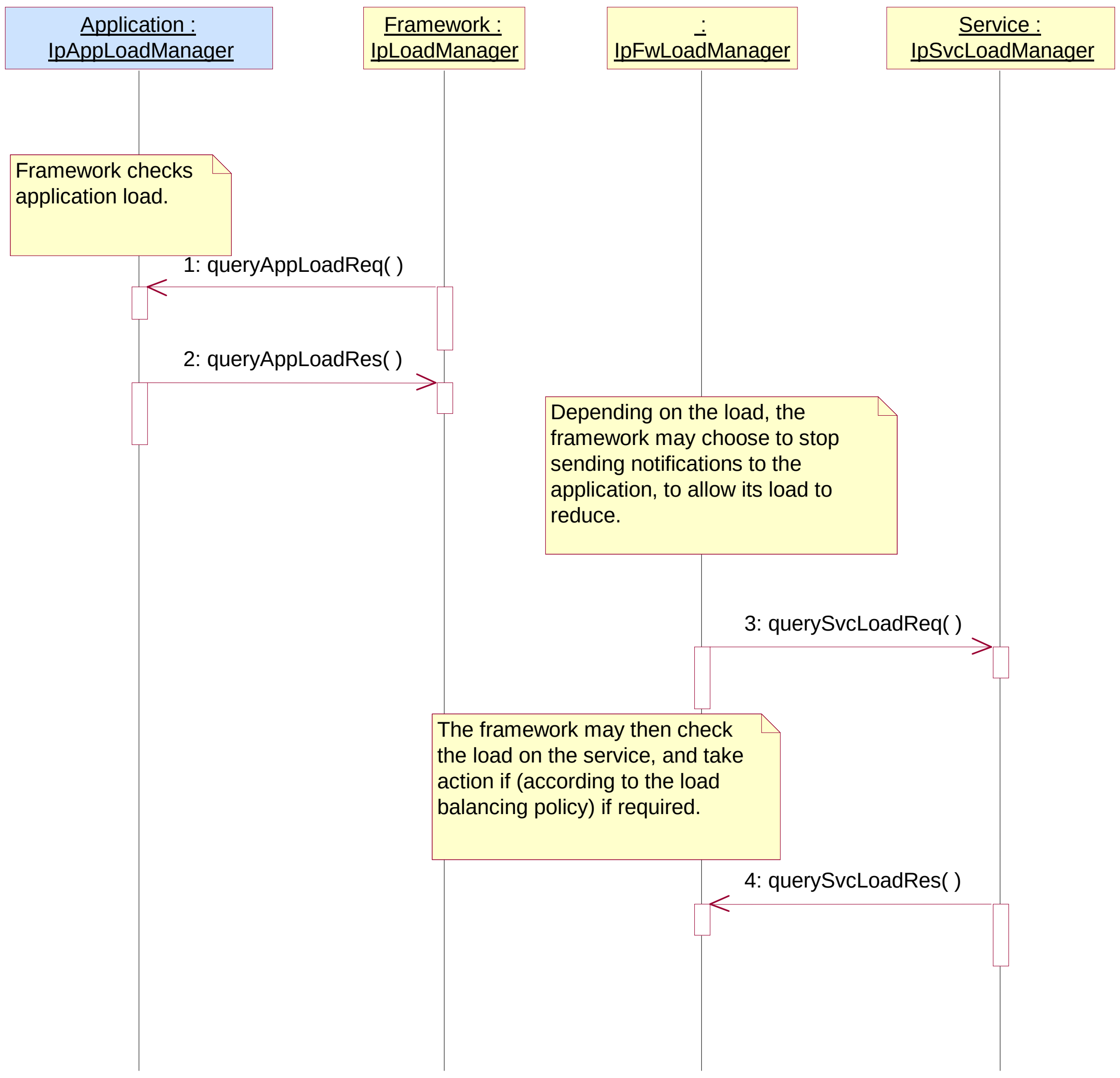
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest 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.

**************** Change #1: Sequence Diagram Svc-Fw interface ******************************

8.1.4.2 Load Management: Client and Service Load Balancing


Application : IpAppLoadManager
Framework : IpLoadManager
: IpFwLoadManager
Service : IpSvcLoadManager
Framework checks application load.
1: queryAppLoadReq( )
2: queryAppLoadRes( )
Depending on the load, the framework may choose 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( )


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

**************** End of Change #1: Sequence Diagram Svc-Fw interface ********************************

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-021132**
Meeting #21, Dublin, IRELAND, 28 – 31 October 2002

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**065**	⌘**rev**	**-**	⌘ Current version:	**5.1.0** ⌘

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

Proposed change affects:	UICC apps⌘ ☐	ME ☐	Radio Access Network ☐	Core Network **X**

Title: ⌘	Enable creation/destruction of load level notifications at the request of Framework		
Source: ⌘	N5		
Work item code: ⌘	OSA2	***Date:*** ⌘	31/10/2002
Category: ⌘	**A**	***Release:*** ⌘	*REL-5*
	Use one of the following categories: ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.		*Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *Rel-4 (Release 4)* *Rel-5 (Release 5)* *Rel-6 (Release 6)*

Reason for change: ⌘	Load Management supports both push and pull mechanisms. Although the APIs define methods on the Framework interface to receive autonomous load notifications from either App or Svc, there is no existing mechanism whereby the Framework can request that this mode of operation take place. The current APIs allow either the App or Svc to request that the Framework operate in this fashion.
Summary of change: ⌘	Introduce the createLoadLevelNotification and destroyLoadLevelNotification methods to the Interfaces, IpAppLoadManager, IpSvcLoadManager.
Consequences if not approved: ⌘	The Framework Load Management specification is incomplete and poorly defined. Implementations of Load Management functionality will be open to vendor specific interpretation with resulting interoperability problems.

Clauses affected: ⌘	7.1.2.5, 7.1.2.6; 7.3.2.7; 8.1.4.1, 8.1.4.2; 8.3.2.8			
	Y	**N**		
Other specs affected: ⌘		**X**	Other core specifications	⌘
		X	Test specifications	
		X	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/specs/CR.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.

Summary:

The OSA framework Load Management capability supports bidirectional exchange of load management information between Applications, Framework and SCSs, with the Framework acting in a central role of manager and dispatcher or load notifications. Applications can obtain load information relating to Framework and SCSs, the Framework can obtain load information relating to Applications and SCSs, and the SCSs can obtain load management information relating to Applications and Framework.

Two mechanisms are provided to supported load management. A Pull mechanism whereby the interested party makes an explicit request to obtain (pull) the load status of the interrogated party, and a push mechanism whereby the interested party registers an interest to receive load information (via push) from the interrogated party at some point in the future.

The pull mechanism is supported by the IpAppLoadManager::queryAppLoadReq, IpLoadManager::queryLoadReq, IpSvcLoadManager::querySvcLoadReq and IpFwLoadManager::queryLoadReq methods. The Pull mechanism is therefore fully supported across all load management interfaces.

The push mechanism is supported from Framework to Application through use of the IpLoadManager::(createLoadLevelNotification, destroyLoadLevelNotification, suspendNotification, resumeNotification) and IpAppLoadManager::loadLevelNotification methods. Likewise the Framework may also push load notifications to the SCS through using IpFwLoadManager::(createLoadLevelNotification, destroyLoadLevelNotification, suspendNotification, resumeNotification) and IpSvcLoadManager::loadLevelNotification.

However load notifications from either Application or SCS pushed towards the framework are not supported in the same way, with the result that there is no explicit mechanism to commence or cease such notifications from applications or SCSs. The methods currently supported are IpAppLoadManager::(suspendNotification,resumeNotification) and IpLoadManager::reportLoad for applications to push notifications to the framework, and IpSvcLoadManager::(suspendNotification, resumeNotification) and IpFwLoadManager::reportLoad for SCSs to push notifications to the framework.

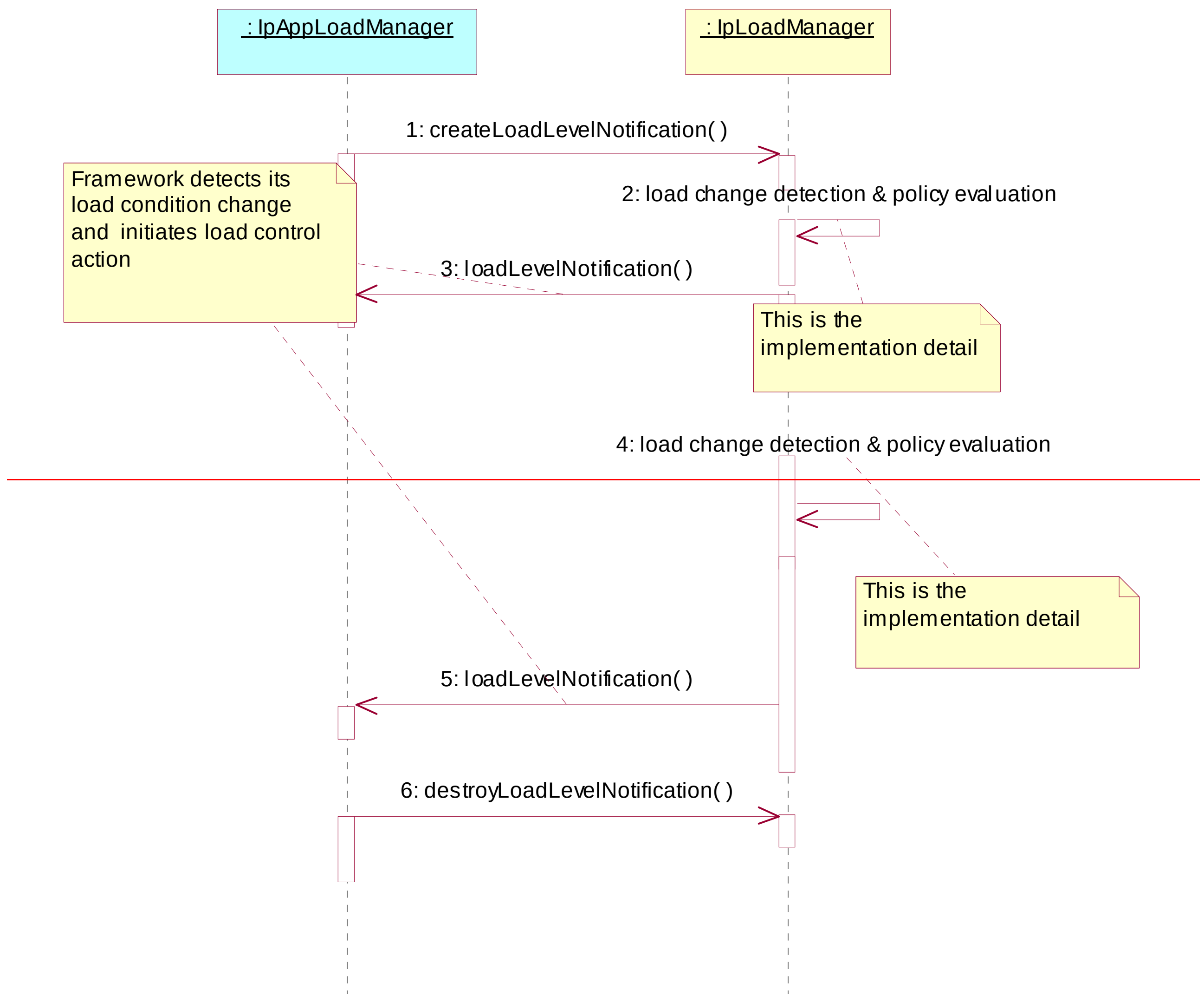
This current imbalance in push mechanisms may result in vendor specific implementations and behaviour that is not fully under the control of the framework. For example applications or services may choose to always assume that load notification is enabled, or interpret the suspend/resume methods for an additional purpose beyond their current specified role.

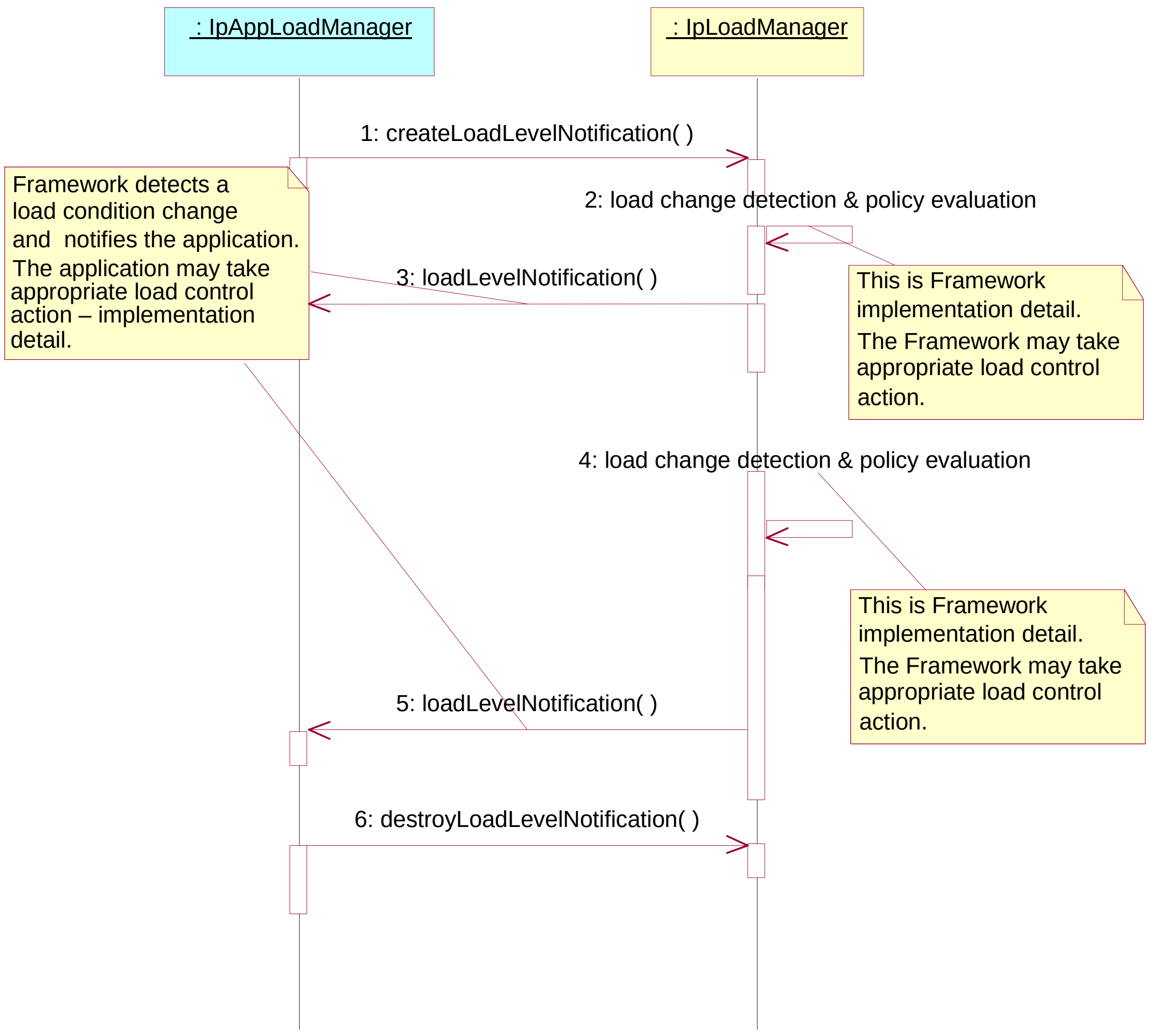
AePONA propose that to guarantee defined behaviour of load management interfaces between Applications, Framework and SCSs, the createLoadNotification and destroyLoadNotification methods are required in both IpAppLoadManager and IpSvcLoadManager interfaces in order to fully support the correct push mechanism. This document outlines the changes required to the framework specification in order to support this.

**************** Change #1: Modified Sequence Diagram App-Fw interface ******************************

7.1.2.5 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( )

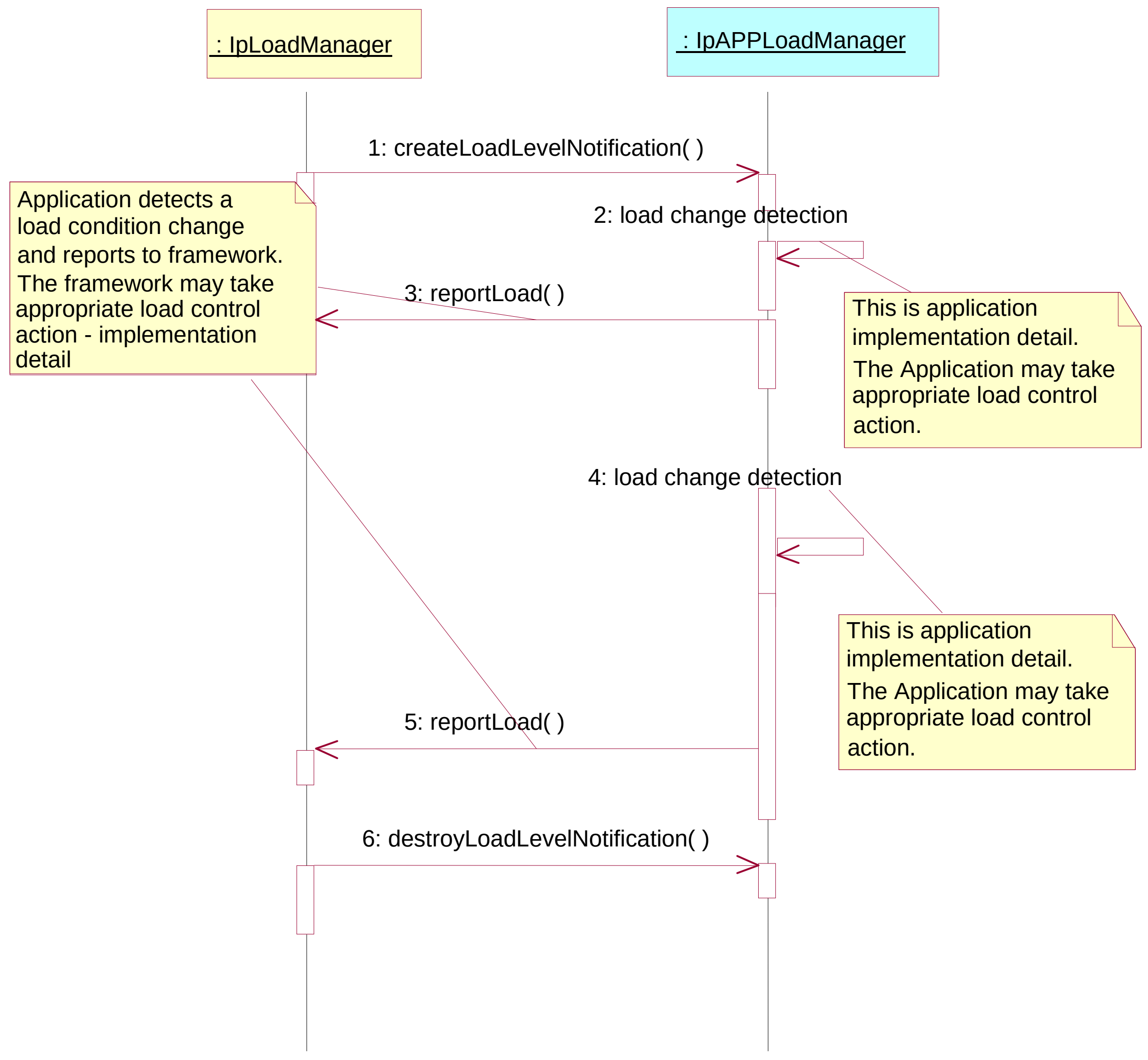

: IpAppLoadManager
: IpLoadManager
1: createLoadLevelNotification( )
2: load change detection & policy evaluation
Framework detects a load condition change and notifies the application. The application may take appropriate load control action – implementation detail.
3: loadLevelNotification( )
This is Framework implementation detail. The Framework may take appropriate load control action.
4: load change detection & policy evaluation
This is Framework implementation detail. The Framework may take appropriate load control action.
5: loadLevelNotification( )
6: destroyLoadLevelNotification( )

**************** End of Change #1: Modified Sequence Diagram App-Fw interface ***************************

**************** Change #2: Additional Sequence Diagram App-Fw interface *******************************

7.1.2.6 Load Management: Framework callback registration and Application load control

This sequence diagram shows how the framework registers itself and the application invokes load management function to inform the framework of application load.


: IpLoadManager
: IpAPPLoadManager
1: createLoadLevelNotification( )
2: load change detection
3: reportLoad( )
Application detects a load condition change and reports to framework. The framework may take appropriate load control action - implementation detail
This is application implementation detail. The Application may take appropriate load control action.
4: load change detection
This is application implementation detail. The Application may take appropriate load control action.
5: reportLoad( )
6: destroyLoadLevelNotification( )

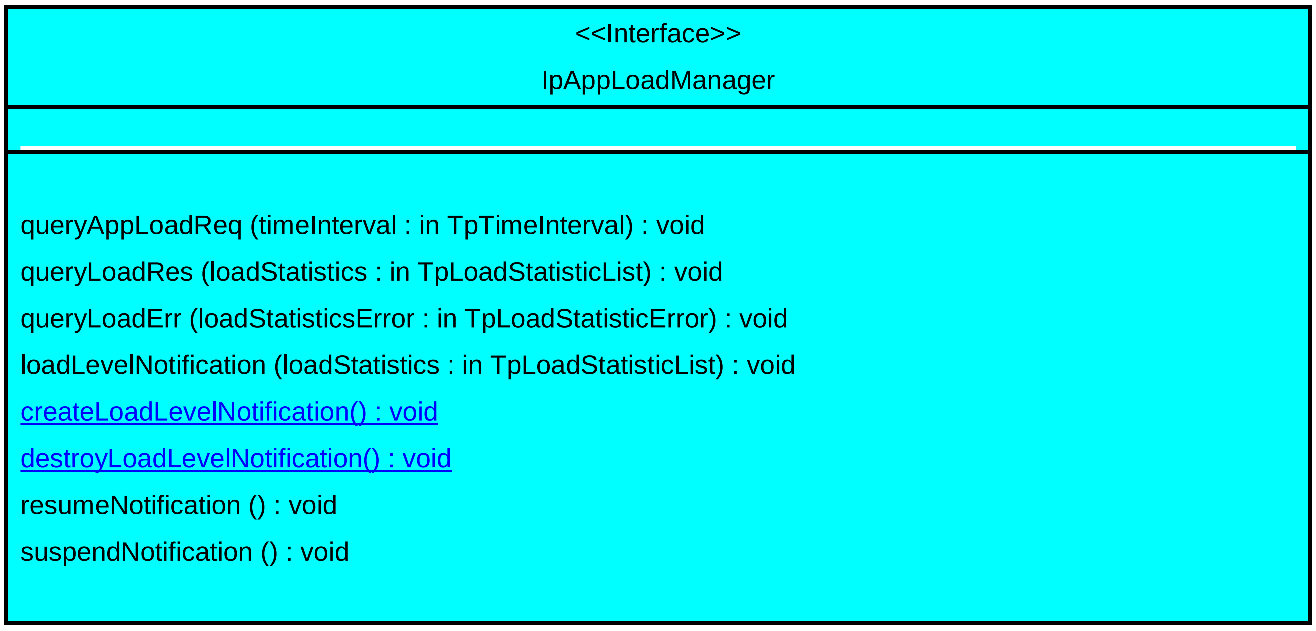
**************** End of Change #2: Additional Sequence Diagram App-Fw interface ***********************

**************** Change #3: Class Definitions App-Fw interface *****************************

7.3.2.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

The client application developer supplies the load manager application interface to handle requests, reports and other responses from the framework load manager function. 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() method on the IpAccess interface.


<<Interface>>
IpAppLoadManager
queryAppLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
createLoadLevelNotification() : void
destroyLoadLevelNotification() : void
resumeNotification () : void
suspendNotification () : void

7.3.2.7.1 Method queryAppLoadReq()

The framework uses this method to request the application to provide load statistics records for the application.

Parameters

`timeInterval : in TpTimeInterval`

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

7.3.2.7.2 Method queryLoadRes()

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

Parameters

`loadStatistics : in TpLoadStatisticList`

Specifies the framework-supplied load statistics

7.3.2.7.3 Method queryLoadErr()

The framework uses this method to return an error response to the application 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 IpLoadManager interface.

Parameters

`loadStatisticsError : in TpLoadStatisticError`

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

7.3.2.7.4 Method loadLevelNotification()

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

Parameters

`loadStatistics : in TpLoadStatisticList`

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

7.3.2.7.5 Method createLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

7.3.2.7.6 Method destroyLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

~~7.1.2.6.5~~7.3.2.7.7 Method resumeNotification()

The framework uses this method to request the application 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

~~7.1.2.6.6~~7.3.2.7.8 Method suspendNotification()

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

Parameters

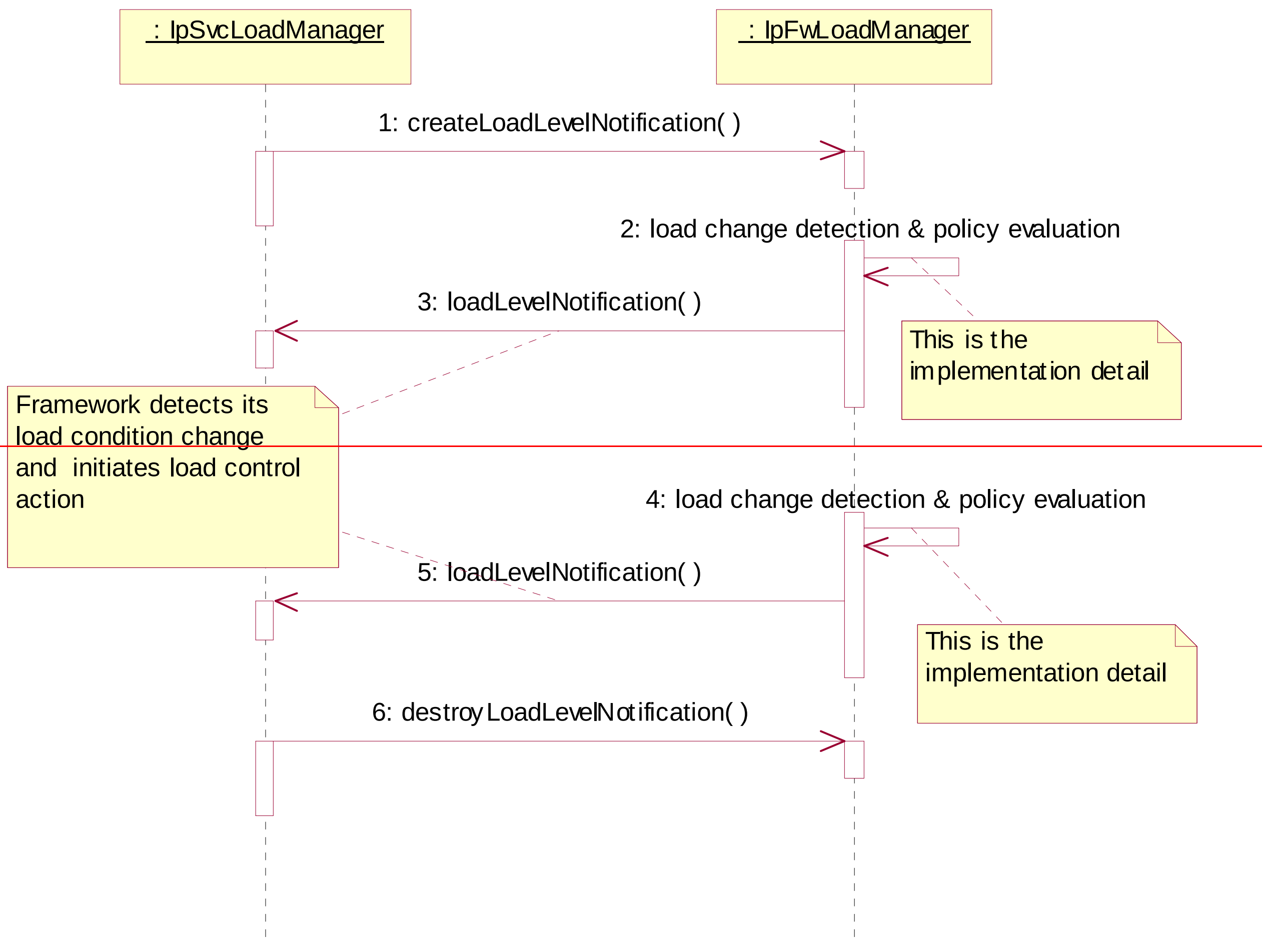
No Parameters were identified for this method

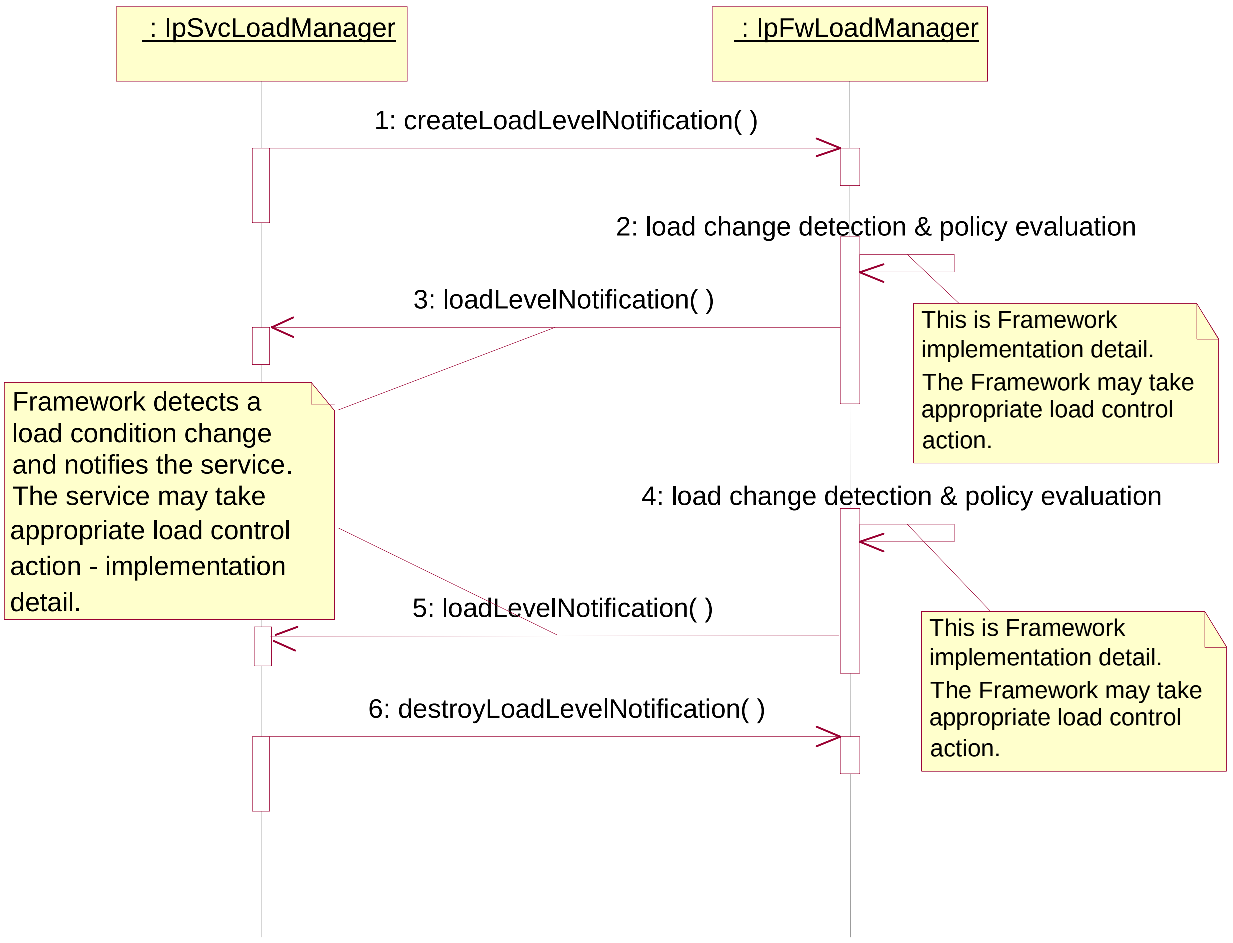
**************** End of Change #3: Class Definitions App-Fw interface *****************************

**************** Change #4: Modified Sequence Diagram Svc-Fw interface *****************************

8.1.4.1 Load Management: Service callback registration and load control

This sequence diagram shows how a service registers itself and the framework invokes load management function based on policy


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

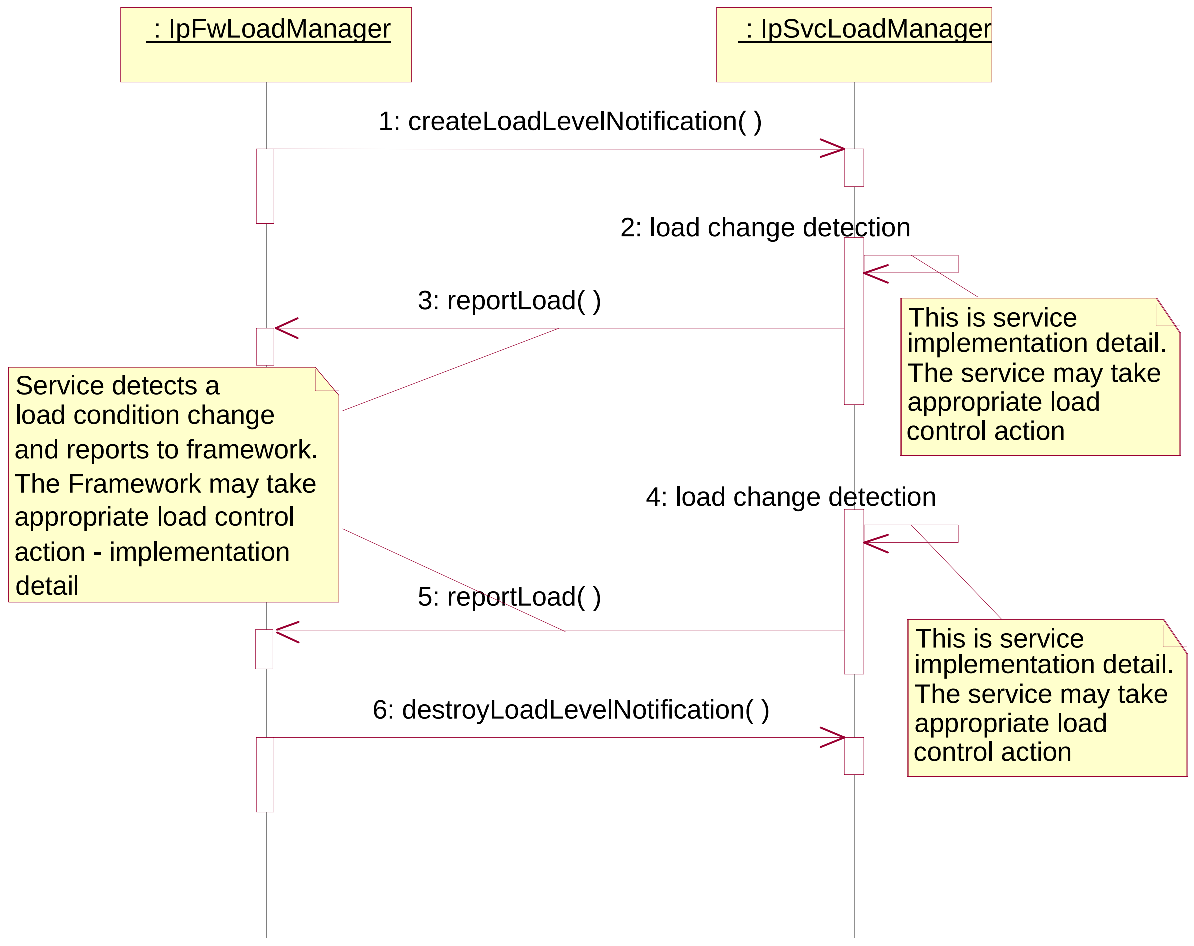

: IpSvcLoadManager
: IpFwLoadManager
1: createLoadLevelNotification( )
2: load change detection & policy evaluation
3: loadLevelNotification( )
This is Framework implementation detail. The Framework may take appropriate load control action.
Framework detects a load condition change and notifies the service. The service may take appropriate load control action - implementation detail.
4: load change detection & policy evaluation
5: loadLevelNotification( )
This is Framework implementation detail. The Framework may take appropriate load control action.
6: destroyLoadLevelNotification( )

**************** End of Change #4: Modified Sequence Diagram App-Fw interface ************************

**************** Change #5: Additional Sequence Diagram Svc-Fw interface ******************************

8.1.4.2 Load Management: Framework callback registration and service load control

This sequence diagram shows how the framework registers itself and the service invokes load management function to inform the framework of service load


: IpFwLoadManager
: IpSvcLoadManager
1: createLoadLevelNotification( )
2: load change detection
3: reportLoad( )
This is service implementation detail. The service may take appropriate load control action
Service detects a load condition change and reports to framework. The Framework may take appropriate load control action - implementation detail
4: load change detection
5: reportLoad( )
This is service implementation detail. The service may take appropriate load control action
6: destroyLoadLevelNotification( )

**************** End of Change #5: Additional Sequence Diagram Svc-Fw interface **************************

**************** Change #6: Class Definitions Svc-Fw interface ******************************

8.3.2.8 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 IpAccess interface.

<<Interface>> IpSvcLoadManager
querySvcLoadReq (timeInterval : in TpTimeInterval) : void queryLoadRes (loadStatistics : in TpLoadStatisticList) : void queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void createLoadLevelNotification () : void destroyLoadLevelNotification () : void suspendNotification () : void resumeNotification () : void

8.3.2.8.1 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`

8.3.2.8.2 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`

8.3.2.8.3 Method queryLoadErr()

The framework uses this method to return an error response to the service 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`

8.3.2.8.4 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 or framework which has 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`

8.3.2.8.5 Method createLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

Raises

`TpCommonExceptions`

8.3.2.8.6 Method destroyLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

Raises

`TpCommonExceptions`

~~7.1.2.6.13~~8.3.2.8.7 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`

~~7.1.2.6.14~~8.3.2.8.8 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`

**************** End of Change #6: Class Definitions Svc-Fw interface *****************************

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-021131**
Meeting #21, Dublin, IRELAND, 28 – 31 October 2002

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**064**	⌘ **rev**	**-**	⌘	Current version:	**4.6.0** ⌘

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

Proposed change affects: UICC apps⌘ [ ] ME [ ] Radio Access Network [ ] Core Network [X]

Title: ⌘	Enable creation/destruction of load level notifications at the request of Framework		
Source: ⌘	N5		
Work item code: ⌘	OSA1	***Date:*** ⌘	31/10/2002
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)* *Rel-6 (Release 6)*

Reason for change: ⌘	Load Management supports both push and pull mechanisms. Although the APIs define methods on the Framework interface to receive autonomous load notifications from either App or Svc, there is no existing mechanism whereby the Framework can request that this mode of operation take place. The current APIs allow either the App or Svc to request that the Framework operate in this fashion.
Summary of change: ⌘	Introduce the createLoadLevelNotification and destroyLoadLevelNotification methods to the Interfaces, IpAppLoadManager, IpSvcLoadManager.
Consequences if not approved: ⌘	The Framework Load Management specification is incomplete and poorly defined. Implementations of Load Management functionality will be open to vendor specific interpretation with resulting interoperability problems.

Clauses affected: ⌘	7.1.2.5, 7.1.2.6; 7.3.3.7; 8.1.4.1, 8.1.4.2; 8.3.4.8			
	Y	**N**		
Other specs affected: ⌘		**X**	Other core specifications	⌘
		X	Test specifications	
		X	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/specs/CR.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.

Summary:

The OSA framework Load Management capability supports bidirectional exchange of load management information between Applications, Framework and SCSs, with the Framework acting in a central role of manager and dispatcher or load notifications. Applications can obtain load information relating to Framework and SCSs, the Framework can obtain load information relating to Applications and SCSs, and the SCSs can obtain load management information relating to Applications and Framework.

Two mechanisms are provided to supported load management. A Pull mechanism whereby the interested party makes an explicit request to obtain (pull) the load status of the interrogated party, and a push mechanism whereby the interested party registers an interest to receive load information (via push) from the interrogated party at some point in the future.

The pull mechanism is supported by the IpAppLoadManager::queryAppLoadReq, IpLoadManager::queryLoadReq, IpSvcLoadManager::querySvcLoadReq and IpFwLoadManager::queryLoadReq methods. The Pull mechanism is therefore fully supported across all load management interfaces.

The push mechanism is supported from Framework to Application through use of the IpLoadManager::(createLoadLevelNotification, destroyLoadLevelNotification, suspendNotification, resumeNotification) and IpAppLoadManager::loadLevelNotification methods. Likewise the Framework may also push load notifications to the SCS through using IpFwLoadManager::(createLoadLevelNotification, destroyLoadLevelNotification, suspendNotification, resumeNotification) and IpSvcLoadManager::loadLevelNotification.

However load notifications from either Application or SCS pushed towards the framework are not supported in the same way, with the result that there is no explicit mechanism to commence or cease such notifications from applications or SCSs. The methods currently supported are IpAppLoadManager::(suspendNotification,resumeNotification) and IpLoadManager::reportLoad for applications to push notifications to the framework, and IpSvcLoadManager::(suspendNotification, resumeNotification) and IpFwLoadManager::reportLoad for SCSs to push notifications to the framework.

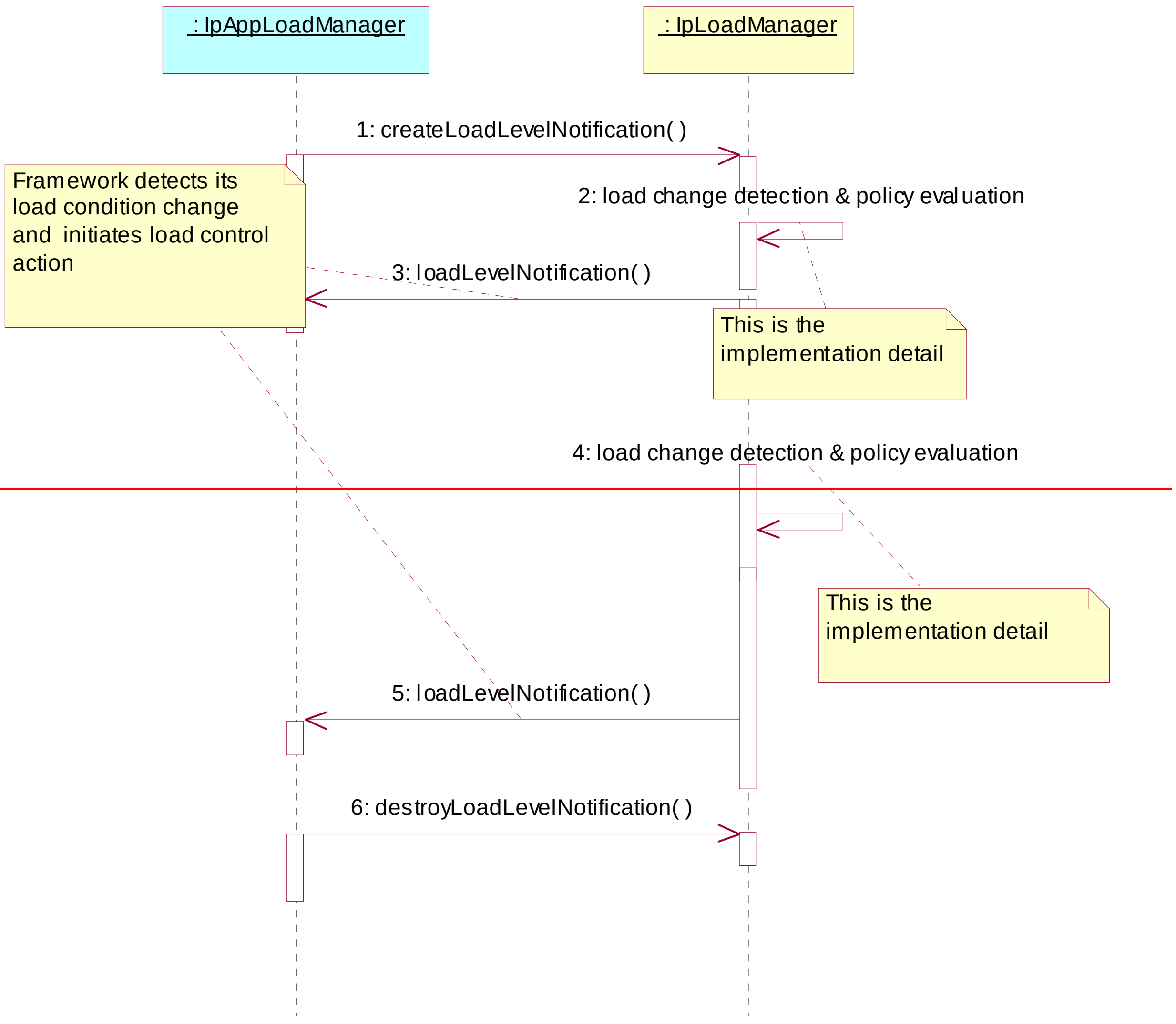
This current imbalance in push mechanisms may result in vendor specific implementations and behaviour that is not fully under the control of the framework. For example applications or services may choose to always assume that load notification is enabled, or interpret the suspend/resume methods for an additional purpose beyond their current specified role.

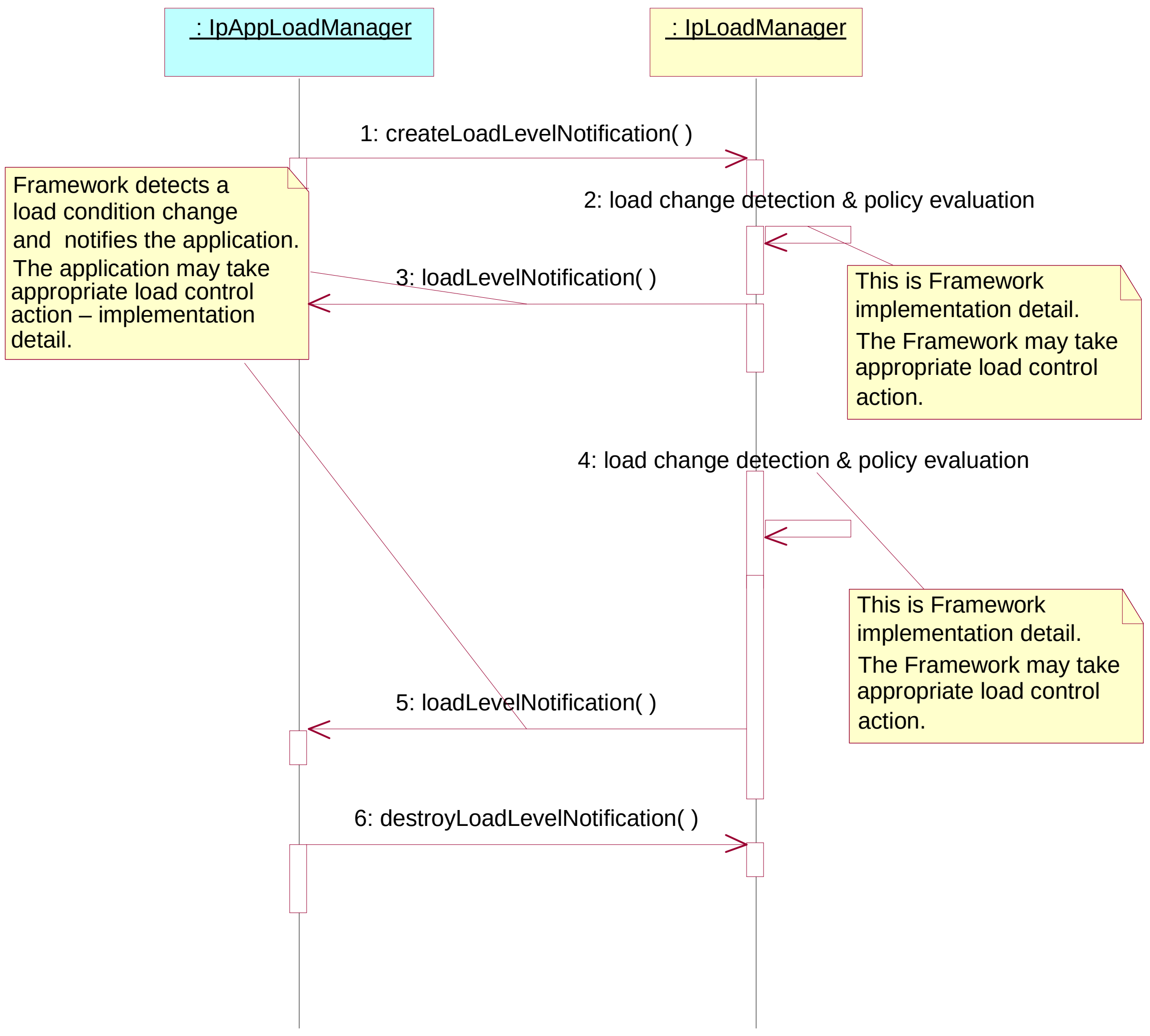
AePONA propose that to guarantee defined behaviour of load management interfaces between Applications, Framework and SCSs, the createLoadNotification and destroyLoadNotification methods are required in both IpAppLoadManager and IpSvcLoadManager interfaces in order to fully support the correct push mechanism. This document outlines the changes required to the framework specification in order to support this.

***************** Change #1: Modified Sequence Diagram App-Fw interface *******************************

7.1.2.5 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( )

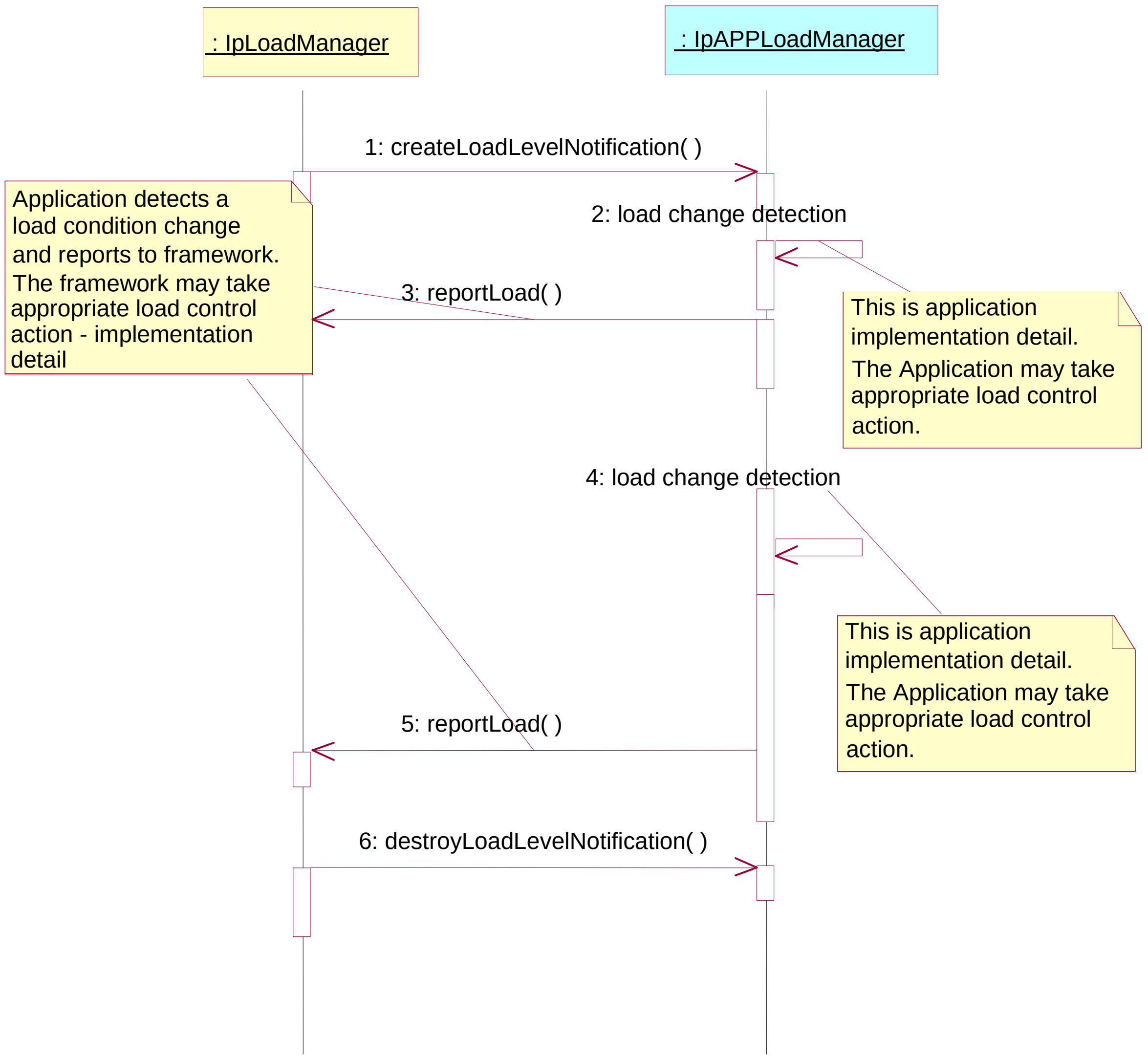

: IpAppLoadManager
: IpLoadManager
1: createLoadLevelNotification( )
2: load change detection & policy evaluation
Framework detects a load condition change and notifies the application. The application may take appropriate load control action – implementation detail.
3: loadLevelNotification( )
This is Framework implementation detail. The Framework may take appropriate load control action.
4: load change detection & policy evaluation
This is Framework implementation detail. The Framework may take appropriate load control action.
5: loadLevelNotification( )
6: destroyLoadLevelNotification( )

************** End of Change #1: Modified Sequence Diagram App-Fw interface *********************

**************** Change #2: Additional Sequence Diagram App-Fw interface ******************************

7.1.2.6 Load Management: Framework callback registration and Application load control

This sequence diagram shows how the framework registers itself and the application invokes load management function to inform the framework of application load.


: IpLoadManager
: IpAPPLoadManager
1: createLoadLevelNotification( )
2: load change detection
3: reportLoad( )
Application detects a load condition change and reports to framework. The framework may take appropriate load control action - implementation detail
This is application implementation detail. The Application may take appropriate load control action.
4: load change detection
This is application implementation detail. The Application may take appropriate load control action.
5: reportLoad( )
6: destroyLoadLevelNotification( )

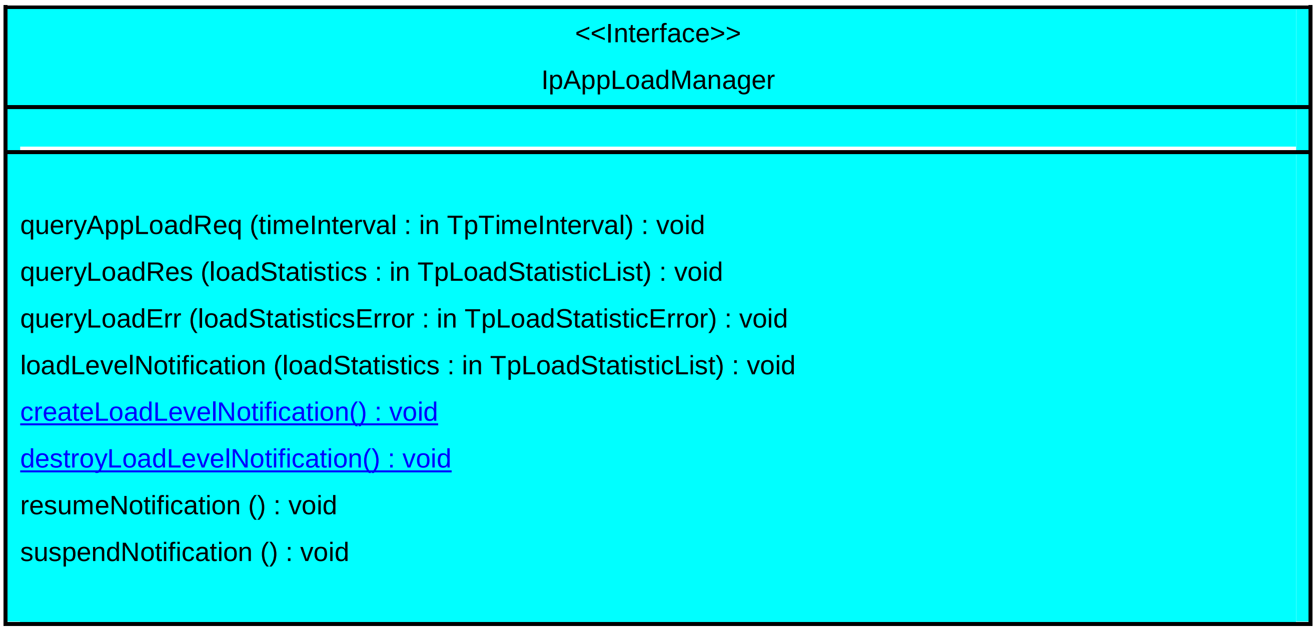
**************** End of Change #2: Additional Sequence Diagram App-Fw interface ***********************

**************** Change #3: Class Definitions App-Fw interface ******************************

7.3.3.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

The client application developer supplies the load manager application interface to handle requests, reports and other responses from the framework load manager function. 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() method on the IpAccess interface.



queryAppLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
createLoadLevelNotification() : void
destroyLoadLevelNotification() : void
resumeNotification () : void
suspendNotification () : void

Method
queryAppLoadReq()

The framework uses this method to request the application to provide load statistics records for the application.

Parameters

timeInterval : in TpTimeInterval

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

Method
queryLoadRes()

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

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics

Method
`queryLoadErr()`

The framework uses this method to return an error response to the application 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 IpLoadManager interface.

Parameters

`loadStatisticsError : in TpLoadStatisticError`

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

Method
`loadLevelNotification()`

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

Parameters

`loadStatistics : in TpLoadStatisticList`

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

Method
`createLoadLevelNotification()`

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

Parameters
No Parameters were identified for this method

Method
`destroyLoadLevelNotification()`

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

Parameters
No Parameters were identified for this method

Method
`resumeNotification()`

The framework uses this method to request the application 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

Method

`suspendNotification()`

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

Parameters

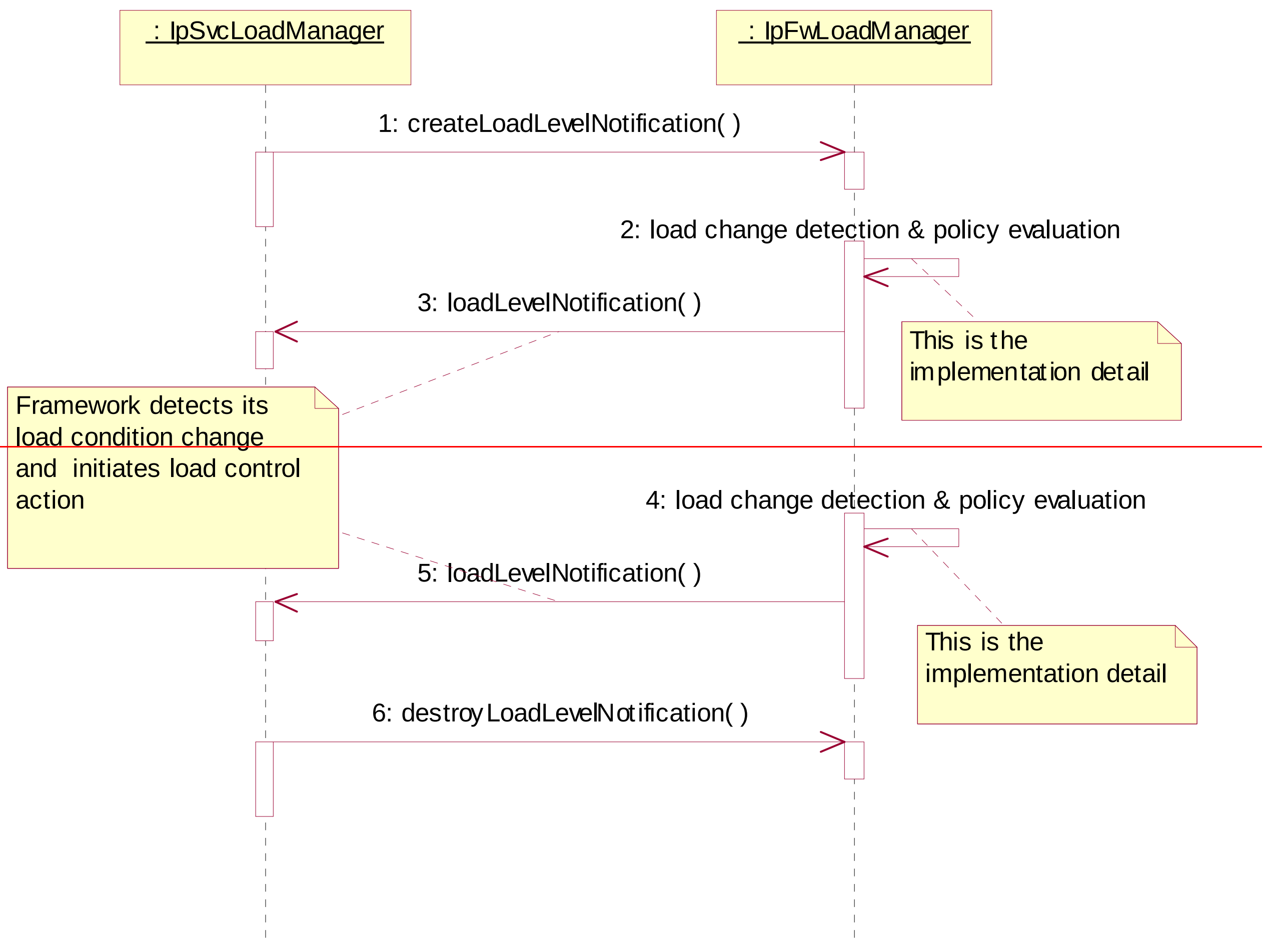
No Parameters were identified for this method

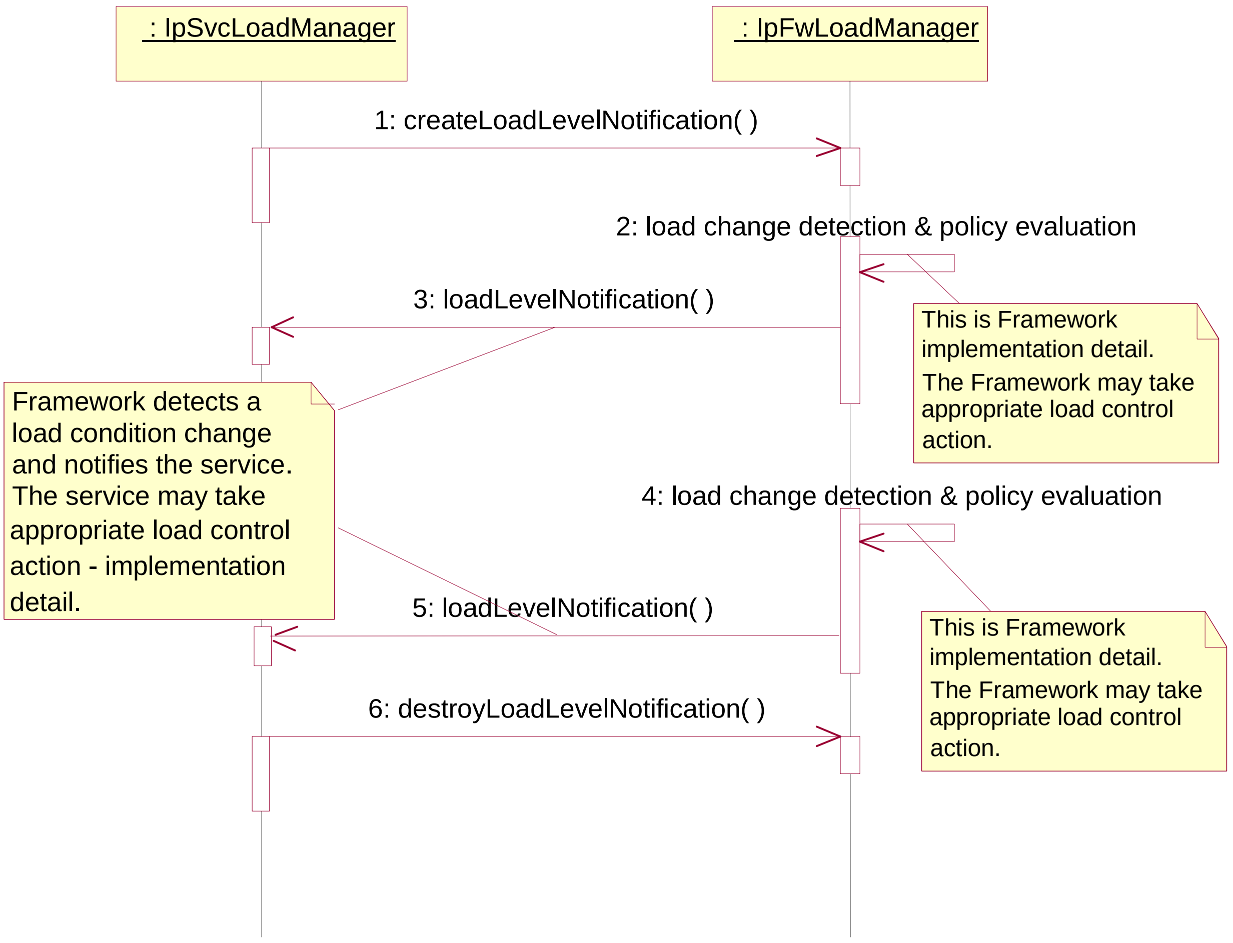
**************** End of Change #3: Class Definitions App-Fw interface *******************************

**************** Change #4: Modified Sequence Diagram Svc-Fw interface **************************

8.1.4.1 Load Management: Service callback registration and load control

This sequence diagram shows how a service registers itself and the framework invokes load management function based on policy


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

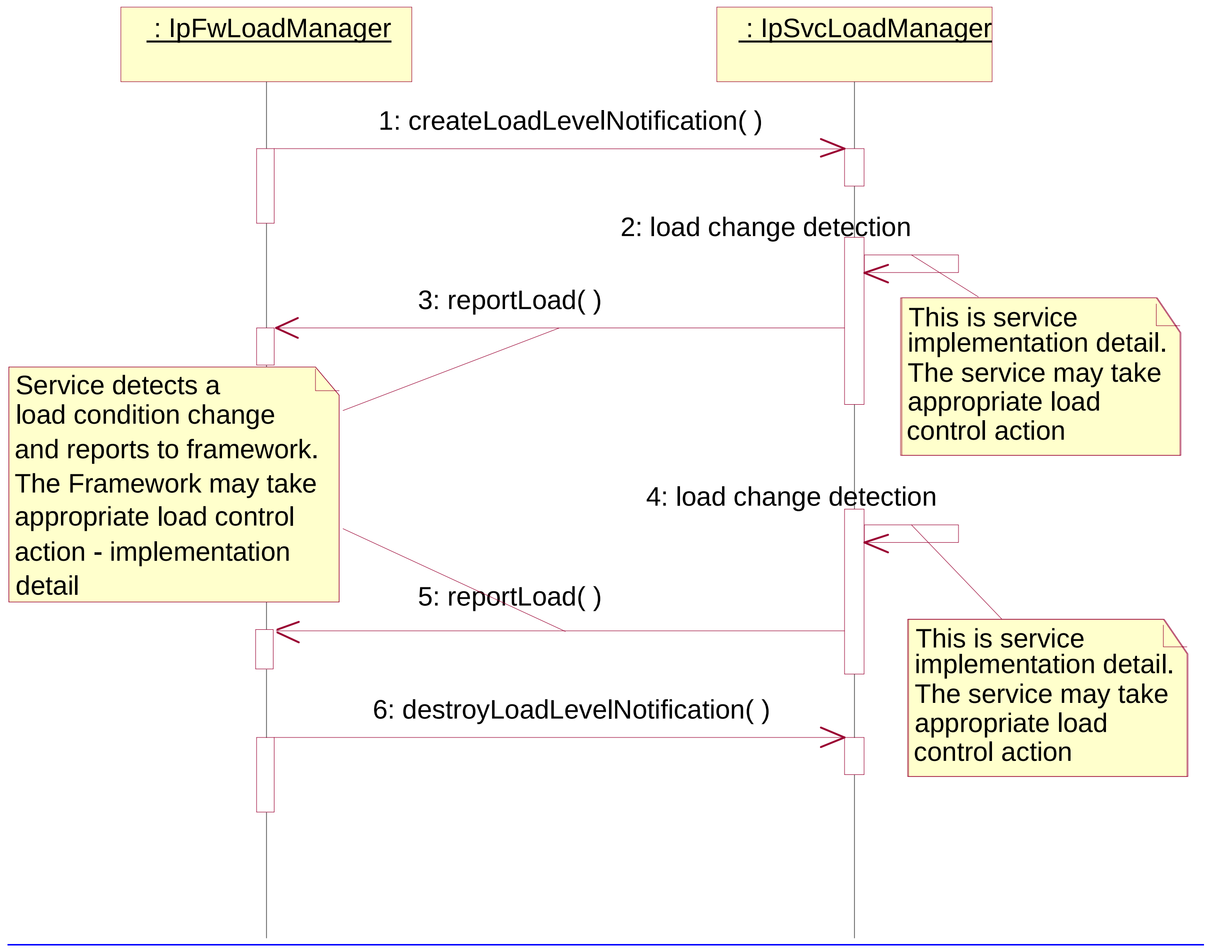

: IpSvcLoadManager
: IpFwLoadManager
1: createLoadLevelNotification( )
2: load change detection & policy evaluation
3: loadLevelNotification( )
This is Framework implementation detail. The Framework may take appropriate load control action.
Framework detects a load condition change and notifies the service. The service may take appropriate load control action - implementation detail.
4: load change detection & policy evaluation
5: loadLevelNotification( )
This is Framework implementation detail. The Framework may take appropriate load control action.
6: destroyLoadLevelNotification( )

************** **End of Change #4: Modified Sequence Diagram Svc-Fw interface** ***********************

************** **Change #5: Additional Sequence Diagram Svc-Fw interface** ****************************

8.1.4.2 Load Management: Framework callback registration and service load control

This sequence diagram shows how the framework registers itself and the service invokes load management function to inform the framework of service load


: IpFwLoadManager
: IpSvcLoadManager
1: createLoadLevelNotification( )
2: load change detection
3: reportLoad( )
This is service implementation detail. The service may take appropriate load control action
Service detects a load condition change and reports to framework. The Framework may take appropriate load control action - implementation detail
4: load change detection
5: reportLoad( )
This is service implementation detail. The service may take appropriate load control action
6: destroyLoadLevelNotification( )

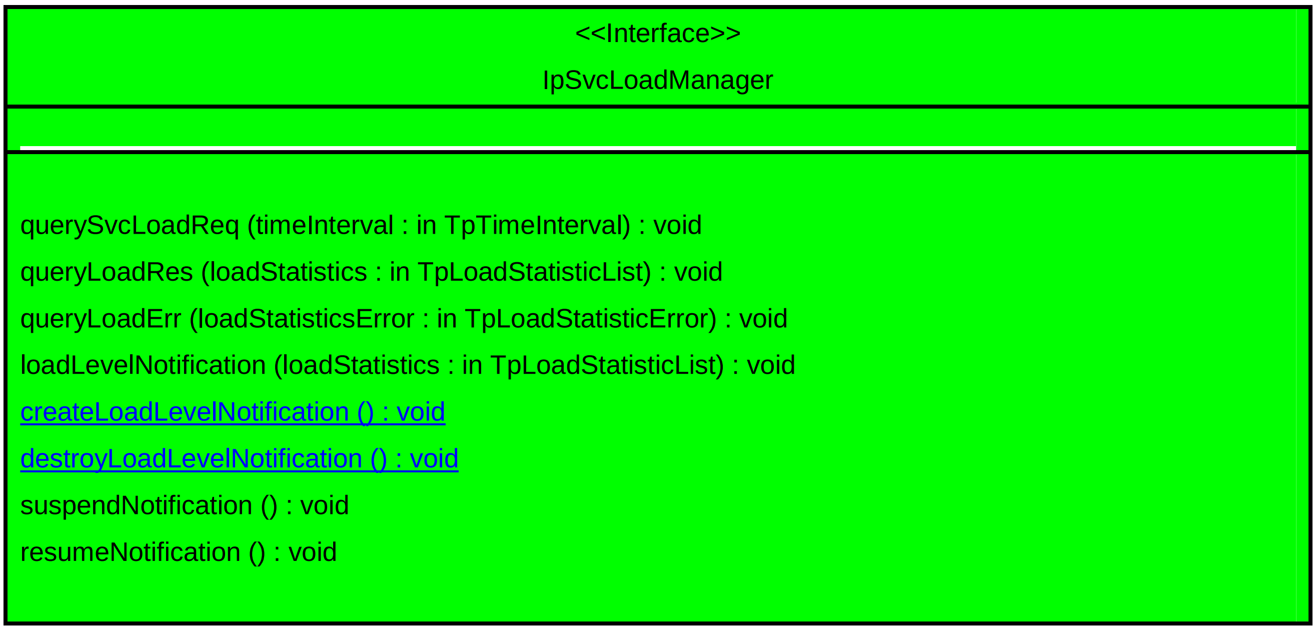
*************** End of Change #5: Additional Sequence Diagram Svc-Fw interface *************************

*************** Change #6: Class Definitions Svc-Fw interface *****************************

8.3.4.8 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 IpAccess interface.


<<Interface>>
IpSvcLoadManager
querySvcLoadReq (timeInterval : in TpTimeInterval) : void
queryLoadRes (loadStatistics : in TpLoadStatisticList) : void
queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : void
loadLevelNotification (loadStatistics : in TpLoadStatisticList) : void
createLoadLevelNotification () : void
destroyLoadLevelNotification () : 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 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 or framework which has 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
createLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

Method
destroyLoadLevelNotification()

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

Parameters

No Parameters were identified for this method

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`

************** End of Change #6: Class Definitions Svc-Fw interface ****************************

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-021071**
Meeting #21, Dublin, IRELAND, 28 – 31 October 2002

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-03 CR 063** ⌘ **rev** **-** ⌘ Current version: **5.1.0** ⌘

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

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network [X]

Title: ⌘	Correction to Initial Access Sequence Diagram	
Source: ⌘	N5	
Work item code: ⌘	OSA2	***Date:*** ⌘ 31/10/2002
Category: ⌘	**A**	***Release:*** ⌘ *REL-5*
	Use one of the following categories: ***F*** *(correction)* ***A*** *(corresponds to a correction in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)* Detailed explanations of the above categories can be found in 3GPP TR 21.900.	*Use one of the following releases:* *2 (GSM Phase 2)* *R96 (Release 1996)* *R97 (Release 1997)* *R98 (Release 1998)* *R99 (Release 1999)* *Rel-4 (Release 4)* *Rel-5 (Release 5)* *Rel-6 (Release 6)*

Reason for change: ⌘	A developer has reported the following error to ETSI Secretariat: In the Initial Access sequence diagram in Release-4 of the Framework, the requestAccess() method is shown as being invoked on IpInitial interface, when it should be invoked on IpAPILevelAuthentication. The same error exists in the Release 5 specification.
Summary of change: ⌘	Correct the sequence diagram to show requestAccess() being invoked on IpAPILevelAuthentication
Consequences if not approved: ⌘	Developers use these sequence diagrams as examples of how OSA/Parlay really behaves. Since they consider that these examples are provided by the real experts, they consider they must be right and should be followed. If we don't correct such errors, we are deliberately misleading developers, and can only expect interoperability problems at later stages. If we don't correct the errors which are reported by developers, they might believe that this specification is dead and should not be used.

Clauses affected: ⌘			6.1.1.1	
	Y	**N**		
Other specs affected: ⌘		**X**	Other core specifications	⌘
		X	Test specifications	
		X	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/specs/CR.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.

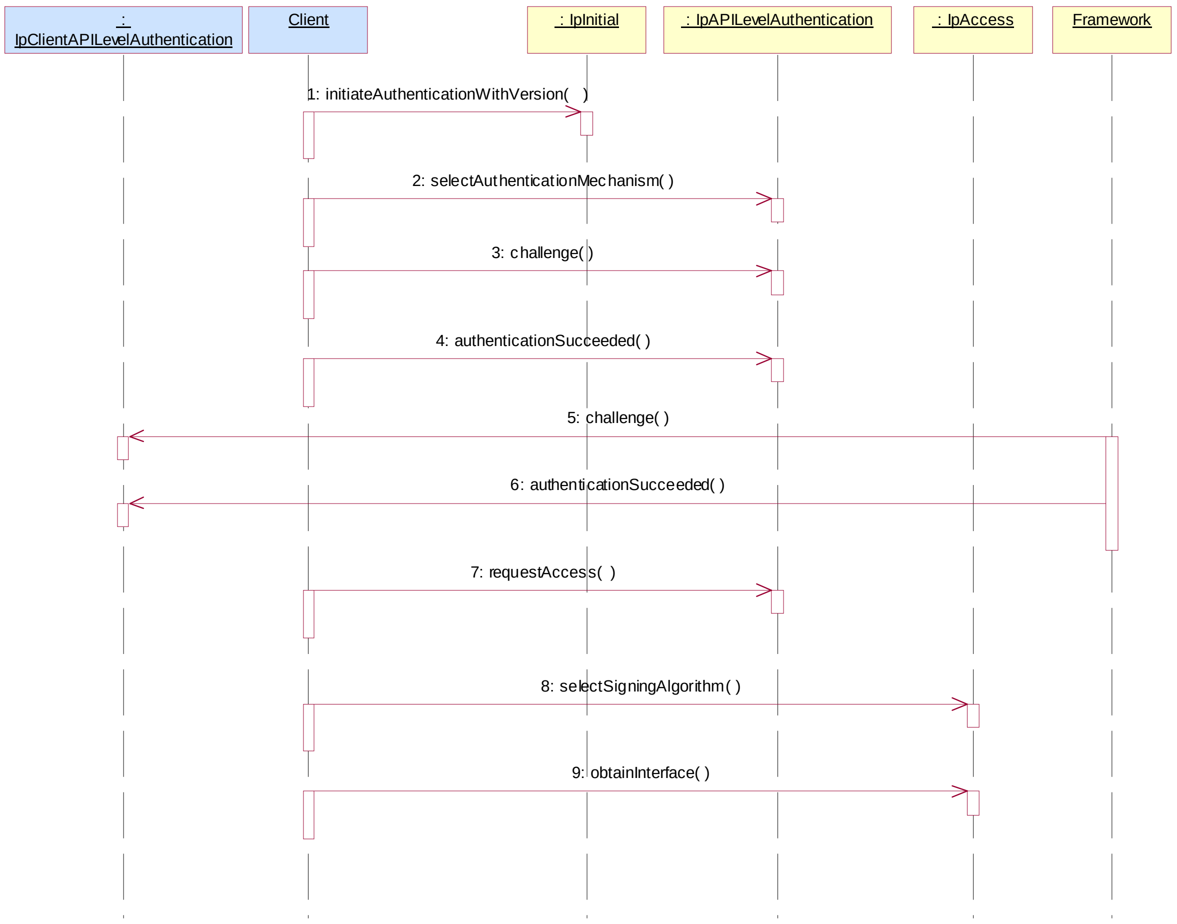
6.1.1.1 Initial Access

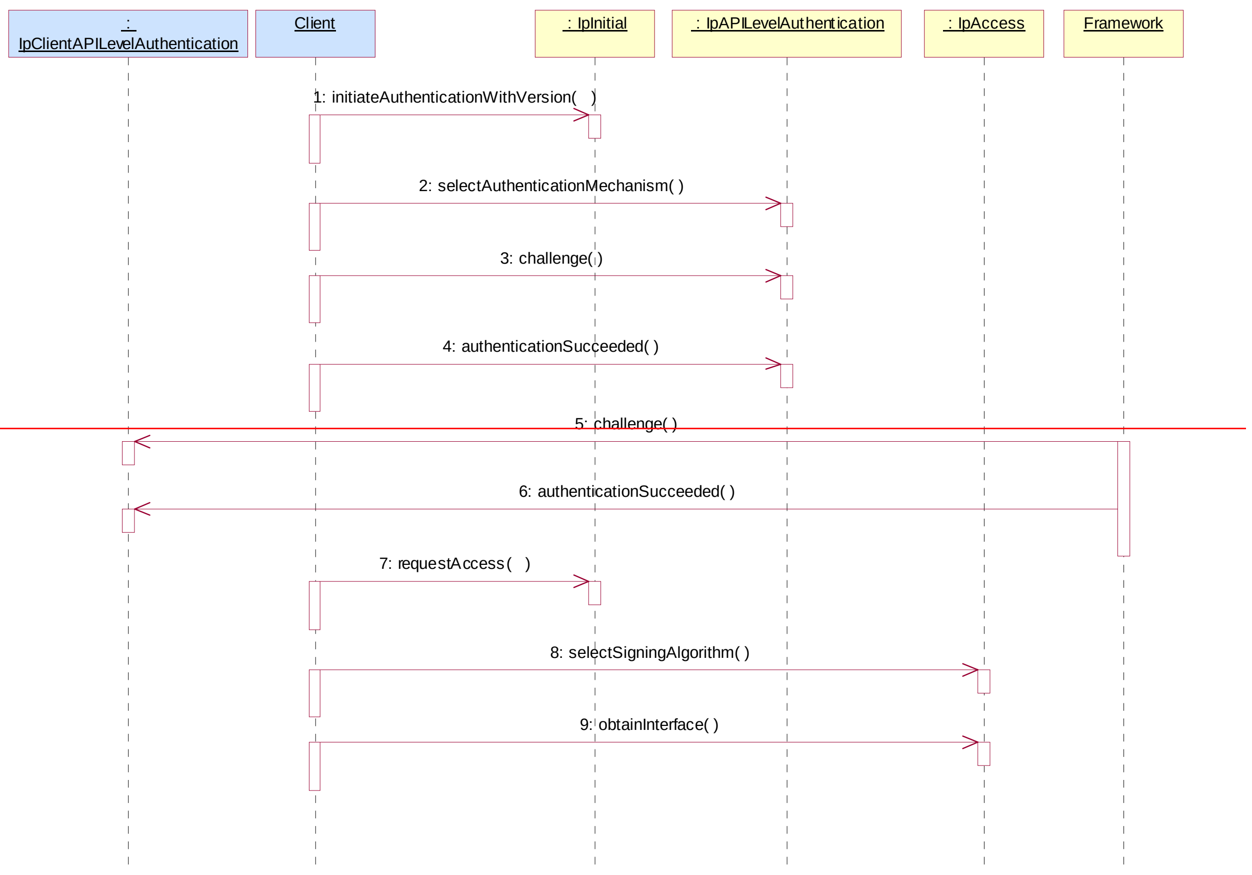
The following figure shows a client accessing the OSA Framework for the first time.

Before being authorized to use the OSA SCFs, the client must first of all authenticate itself with the Framework. For this purpose the client needs a reference to the Initial Contact interfaces for the Framework; this may be obtained through a URL, a Naming or Trading Service or an equivalent service, a stringified object reference, etc. At this stage, the client has no guarantee that this is a Framework interface reference, but it to initiate the authentication process with the Framework. The Initial Contact interface supports only the initiateAuthenticationWithVersion method to allow the authentication process to take place.

Once the client has been authenticated by the Framework, it can gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method, by which the client requests a certain type of access SCF.

Independently, the client could decide to authenticate the Framework, before deciding to continue using the interfaces provided by the Framework.


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthenticationWithVersion( )
2: selectAuthenticationMechanism( )
3: challenge( )
4: authenticationSucceeded( )
5: challenge( )
6: authenticationSucceeded( )
7: requestAccess( )
8: selectSigningAlgorithm( )
9: obtainInterface( )


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthenticationWithVersion( )
2: selectAuthenticationMechanism( )
3: challenge( )
4: authenticationSucceeded( )
5: challenge( )
6: authenticationSucceeded( )
7: requestAccess( )
8: selectSigningAlgorithm( )
9: obtainInterface( )

1: Initiate Authentication

The client invokes initiateAuthenticationWithVersion 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: Select Authentication Mechanism

The client invokes selectAuthenticationMechanism on the Framework's API Level Authentication interface, identifying the authentication algorithm it supports for use with CHAP authentication. The Framework prescribes the method to be used. OSA authentication is based on CHAP, which prescribes the MD5 hashing algorithm as the minimum to be supported. Note however that the framework need not accept this algorithm.

3: The client authenticates the Framework, issuing a challenge in the challenge() method.

4: The client provides an indication if authentication succeeded.

5: The Framework authenticates the client. The sequence diagram illustrates one of a series of one or more invocations of the challenge method on the client's API Level Authentication interface. In each invocation, the Framework supplies a challenge and the client returns the correct response. The Framework could authenticate the client before the client authenticates the Framework, or afterwards, or the two authentication processes could be interleaved. However, the client shall respond immediately to any challenge issued by the Framework, as the Framework might not respond to any challenge issued by the client until the Framework has successfully authenticated the client.

6: The Framework provides an indication if authentication succeeded.

7: Request Access

Upon successful authentication of the client by the Framework, the client is permitted to invoke requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The

Framework returns a reference to its access interface. The success or failure of the client's authentication of the Framework does not affect the client's right to invoke requestAccess.

8: The client and framework negotiate the signing algorithm to be used for any signed exchanges.

9: The client invokes obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-021070**
Meeting #21, Dublin, IRELAND, 28 – 31 October 2002

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**062**	⌘ **rev**	**-**	⌘ Current version:	**4.6.0** ⌘

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

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network [✓ X]

Title:	⌘	Correction to Initial Access Sequence Diagram		
Source:	⌘	N5		
Work item code:	⌘	OSA1	***Date:*** ⌘	31/10/2002
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)*
Rel-6	*(Release 6)*

Reason for change:	⌘	A developer has reported the following error to ETSI Secretariat: In the Initial Access sequence diagram in Release-4 of the Framework, the requestAccess() method is shown as being invoked on IpInitial interface, when it should be invoked on IpAPILevelAuthentication.
Summary of change:	⌘	Correct the sequence diagram to show requestAccess() being invoked on IpAPILevelAuthentication
Consequences if not approved:	⌘	Developers use these sequence diagrams as examples of how OSA/Parlay really behaves. Since they consider that these examples are provided by the real experts, they consider they must be right and should be followed. If we don't correct such errors, we are deliberately misleading developers, and can only expect interoperability problems at later stages. If we don't correct the errors which are reported by developers, they might believe that this specification is dead and should not be used.

Clauses affected: ⌘ 6.1.1.2

Other specs affected: ⌘

Y	N		
	X	Other core specifications	⌘
	X	Test specifications	
	X	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/specs/CR.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.

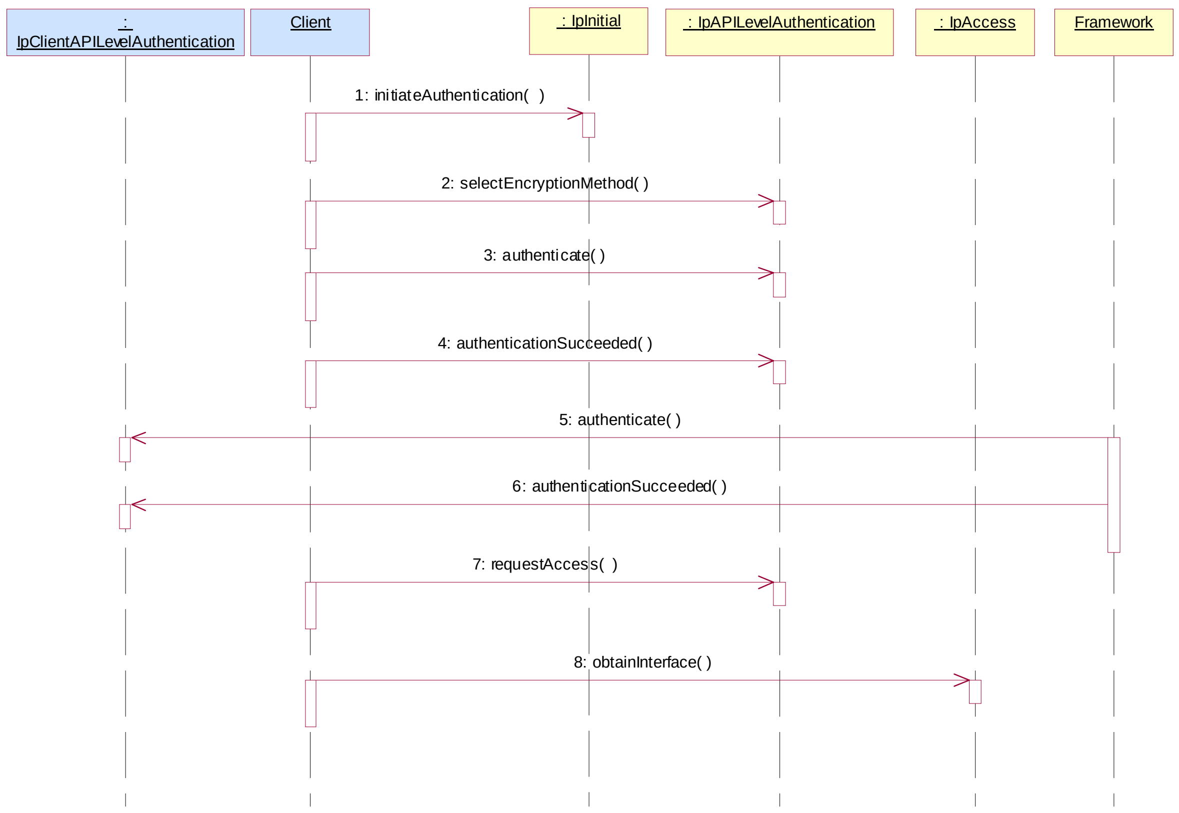
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

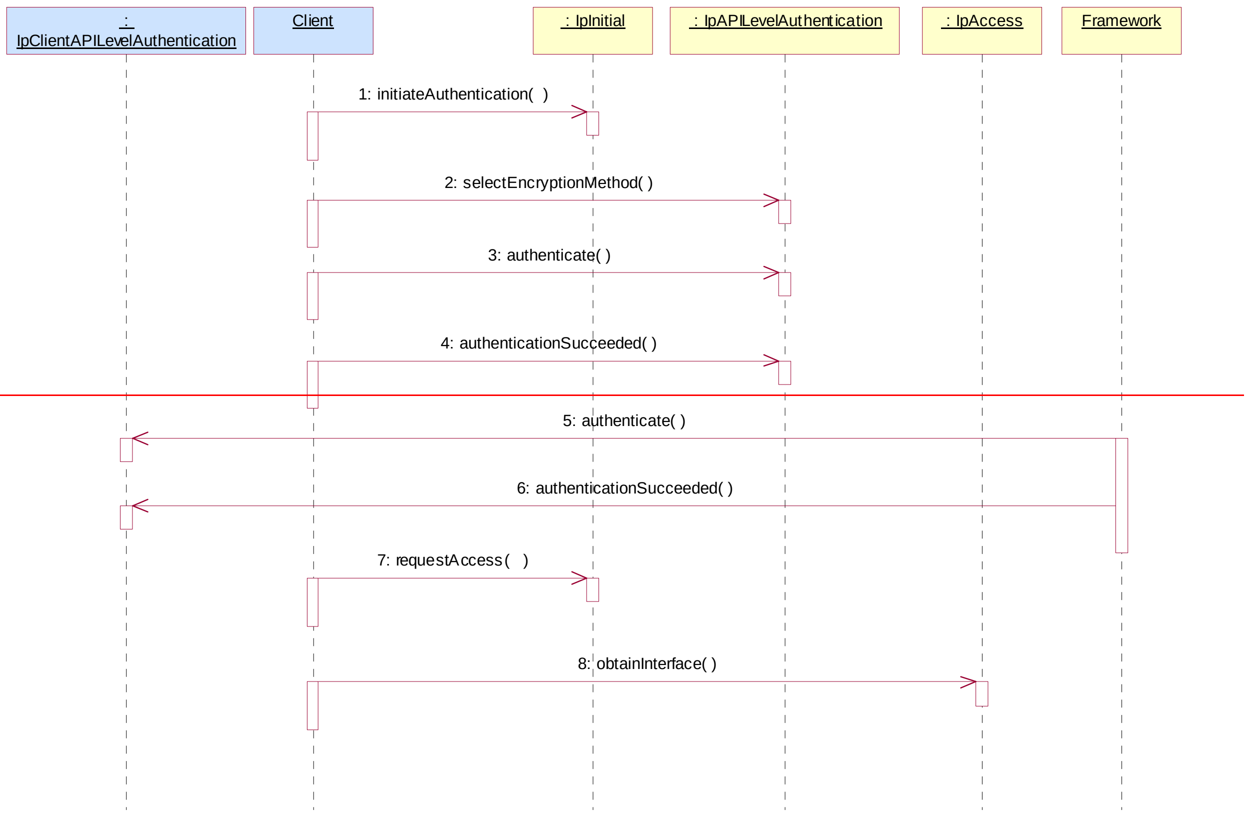
6.1.1.2 Initial Access

The following figure shows a client accessing the OSA Framework for the first time.

Before being authorized to use the OSA SCFs, the client must first of all authenticate itself with the Framework. For this purpose the client needs a reference to the Initial Contact interfaces for the Framework; this may be obtained through a URL, a Naming or Trading Service or an equivalent service, a stringified object reference, etc. At this stage, the client has no guarantee that this is a Framework interface reference, but it to initiate the authentication process with the Framework. The Initial Contact interface supports only the initiateAuthentication method to allow the authentication process to take place.

Once the client has authenticated with the Framework, it can gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method, by which the client requests a certain type of access SCF.


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: selectEncryptionMethod( )
3: authenticate( )
4: authenticationSucceeded( )
5: authenticate( )
6: authenticationSucceeded( )
7: requestAccess( )
8: obtainInterface( )


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
1: initiateAuthentication( )
2: selectEncryptionMethod( )
3: authenticate( )
4: authenticationSucceeded( )
5: authenticate( )
6: authenticationSucceeded( )
7: requestAccess( )
8: obtainInterface( )

1: Initiate Authentication

The client 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: Select Encryption Method

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

3: Authenticate

4: The client provides an indication if authentication succeeded.

5: The client and Framework authenticate each other. 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 client supplies a challenge and the Framework returns the correct response. Alternatively or additionally the Framework may issue its own challenges to the client using the authenticate method on the client's API Level Authentication interface.

6: The Framework provides an indication if authentication succeeded.

7: Request Access

Upon successful (mutual) authentication, the client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

8: The client invokes obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020896**
Meeting #20, Miami/ FL, USA, 23 – 27 September 2002

CR-Form-v7

CHANGE REQUEST

⌘	**29.198-03**	**CR**	**061**	⌘ **rev**	**-**	⌘	Current version:	**4.6.0** ⌘

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

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network [✓ X]

Title:	⌘	Correction of status of methods to interfaces in clause 8.3
Source:	⌘	N5
Work item code:	⌘	OSA1
Date:	⌘	27/09/2002
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)*
Rel-6	*(Release 6)*

Reason for change:	⌘	There is no requirement in the standard about the necessity to implement all or only some of the methods defined for an interface.
Summary of change:	⌘	Clarify which methods are mandatory and which are optional.
Consequences if not approved:	⌘	Application developers will not know which methods will actually be available.

Clauses affected:	⌘	8.3 Interface Classes				
		Y	**N**			
Other specs affected:	⌘		**X**	Other core specifications	⌘	
			X	Test specifications		
			X	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/specs/CR.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.

8.3 Interface Classes

8.3.1 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 clause 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.

8.3.1.1 Interface Class IpFwServiceRegistration

Inherits from: IpInterface.

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

This interface and at least the methods registerService(), announceServiceAvailability(), unregisterService() and unannounceService() shall be implemented by a Framework.

<<Interface>> IpFwServiceRegistration
registerService (serviceTypeName : in TpServiceTypeName, servicePropertyList : in TpServicePropertyList) : TpServiceID announceServiceAvailability (serviceID : in TpServiceID, serviceInstanceLifecycleManagerRef : in service_lifecycle::IpServiceInstanceLifecycleManagerRef) : void unregisterService (serviceID : in TpServiceID) : void describeService (serviceID : in TpServiceID) : TpServiceDescription unannounceService (serviceID : in TpServiceID) : void

Method
registerService()

The registerService() operation is the means by which a service is registered in the Framework, for subsequent discovery by the enterprise applications. Registration can only succeed when the Service type of the service is known to the Framework (ServiceType is 'available'). A service-ID is returned to the service supplier when a service is registered in the Framework. When the service is not registered because the ServiceType is 'unavailable', a P_SERVICE_TYPE_UNAVAILABLE is raised. 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.

Returns <serviceID> : 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.

Parameters

`serviceTypeName : in TpServiceTypeName`

The "serviceTypeName" parameter identifies the service type. 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 behavioural, 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. Examples 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.

Returns

`TpServiceID`

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, P_SERVICE_TYPE_UNAVAILABLE`

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 instance lifecycle manager is instantiated at a particular interface. This method informs the framework of the availability of "service instance lifecycle manager" 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 IpServiceInstanceLifecycleManager interface. The IpServiceInstanceLifecycleManager interface supports a method called the createServiceManager(application: in TpClientAppID, serviceProperties : in TpServicePropertyList, serviceInstanceID : in TpServiceInstanceID) : IpServiceRef. 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.

serviceInstanceLifecycleManagerRef : in service_lifecycle::IpServiceInstanceLifecycleManagerRef

The interface reference at which the service instance lifecycle manager 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.

Returns <serviceDescription> : 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.

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.

Returns

`TpServiceDescription`

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`

8.3.2 Service Instance Lifecycle Manager Interface Classes

The 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.

8.3.2.1 Interface Class IpServiceInstanceLifecycleManager

Inherits from: IpInterface.

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

This interface and ~~all its~~the createServiceManager() and destroyServiceManager() methods shall be implemented by a Service.

<<Interface>> IpServiceInstanceLifecycleManager
createServiceManager (application : in TpClientAppID, serviceProperties : in TpServicePropertyList, serviceInstanceID : in TpServiceInstanceID) : IpServiceRef destroyServiceManager (serviceInstance : in TpServiceInstanceID) : void

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.

Returns <serviceManager> : Specifies the service manager interface reference for the specified application ID.

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.

Returns

IpServiceRef

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`

8.3.3 Service Discovery Interface Classes

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

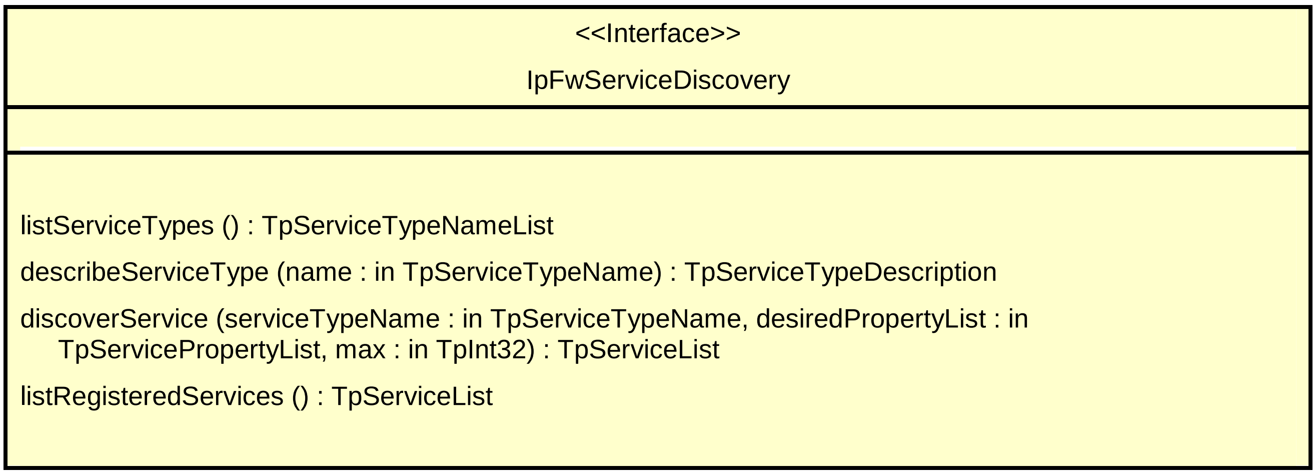
Before a service can be registered in the framework, the service supplier must know what "types" of services the Framework supports and what service "properties" are applicable to each service type. The "listServiceType()" method returns a list of all "service types" that are currently supported by the 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.

8.3.3.1 Interface Class IpFwServiceDiscovery

Inherits from: IpInterface.

This interface shall be implemented by a Framework with as a minimum requirement the listServiceTypes(), describeServiceType() and discoverService() methods.



```
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 available or unavailable.

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 properties {name and value list} 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 properties {name and value list} 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 properties {name and value list} associated with the service.

Parameters
No Parameters were identified for this method

Returns

`TpServiceList`

Raises

`TpCommonExceptions`

8.3.4 Integrity Management Interface Classes

8.3.4.1 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 IpAccess interface.

If the IpFwFaultManager interface is implemented by a Framework, at least one of these methods shall be implemented. If the Framework is capable of invoking the IpSvcFaultManager.svcActivityTestReq() method, it shall implement svcActivityTestRes() and svcActivityTestErr() in this interface. If the Framework is capable of invoking IpSvcFaultManager.genFaultStatsRecordReq(), it shall implement genFaultStatsRecordRes() and genFaultStatsRecordErr() in this interface.

<<Interface>> IpFwFaultManager
activityTestReq (activityTestID : in TpActivityTestID, testSubject : in TpSubjectType) : void svcActivityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void appUnavailableInd () : void genFaultStatsRecordReq (timePeriod : in TpTimeInterval, recordSubject : in TpSubjectType) : void svcUnavailableInd (reason : in TpSvcUnavailReason) : void svcActivityTestErr (activityTestID : in TpActivityTestID) : void genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord, serviceIDs : in TpServiceIDList) : void genFaultStatsRecordErr (faultStatisticsError : in TpFaultStatisticsError, serviceIDs : in TpServiceIDList) : 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

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. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the framework.

recordSubject : 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

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

genFaultStatsRecordRes()

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

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

`serviceIDs : in TpServiceIDList`

Specifies the services that are included in the general fault statistics record. The serviceIDs parameter is not allowed to be an empty list.

Raises

`TpCommonExceptions`

Method

`genFaultStatsRecordErr()`

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

Parameters

`faultStatisticsError : in TpFaultStatisticsError`

The fault statistics error.

`serviceIDs : in TpServiceIDList`

Specifies the services that were included in the general fault statistics record request. The serviceIDs parameter is not allowed to be an empty list.

Raises

`TpCommonExceptions`

8.3.4.2 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 IpAccess interface

If the IpSvcFaultManager interface is implemented by a Service, at least one of these methods shall be implemented. If the Service is capable of invoking the IpFwFaultManager.activityTestReq() method, it shall implement activityTestRes() and activityTestErr() in this interface. If the Service is capable of invoking IpFwFaultManager.genFaultStatsRecordReq(), it shall implement genFaultStatsRecordRes() and genFaultStatsRecordErr() in this interface.

<<Interface>>

IpSvcFaultManager

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

svcActivityTestReq (activityTestID : in TpActivityTestID) : void

fwFaultReportInd (fault : in TpInterfaceFault) : void

fwFaultRecoveryInd (fault : in TpInterfaceFault) : void

fwUnavailableInd (reason : in TpFwUnavailReason) : void

svcUnavailableInd () : void

appUnavailableInd () : void

genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord, recordSubject : in TpSubjectType) : void

activityTestErr (activityTestID : in TpActivityTestID) : void

genFaultStatsRecordErr (faultStatisticsError : in TpFaultStatisticsError, recordSubject : in TpSubjectType) : void

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

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 TpInterfaceFault`

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 should assume that the client application is leaving the service session and the service 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

No Parameters were identified for this method

Raises

`TpCommonExceptions`

Method

`genFaultStatsRecordRes()`

This method is used by the framework to provide fault statistics to a service instance 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 application) whose fault statistics record has been provided.

Raises

`TpCommonExceptions`

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

`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 TpFaultStatisticsError`

The fault statistics error.

`recordSubject : in TpSubjectType`

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

Raises

`TpCommonExceptions`

Method

`genFaultStatsRecordReq()`

This method is used by the framework to solicit fault statistics from the service, for example when the framework was asked for these statistics by the client application using the genFaultStatsRecordReq operation on the IpFaultManager interface. On receipt of this request the service must produce a fault statistics record, for either the framework or for the client's instances of the specified services during the specified time interval, which is returned to the framework using the genFaultStatsRecordRes operation on the IpFwFaultManager interface. If the framework 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. Supplying both a start time and stop time as empty strings leaves the time period to the discretion of the service.

`serviceIDs : in TpServiceIDList`

Specifies the services to be included in the general fault statistics record. This parameter is not allowed to be an empty list.

Raises

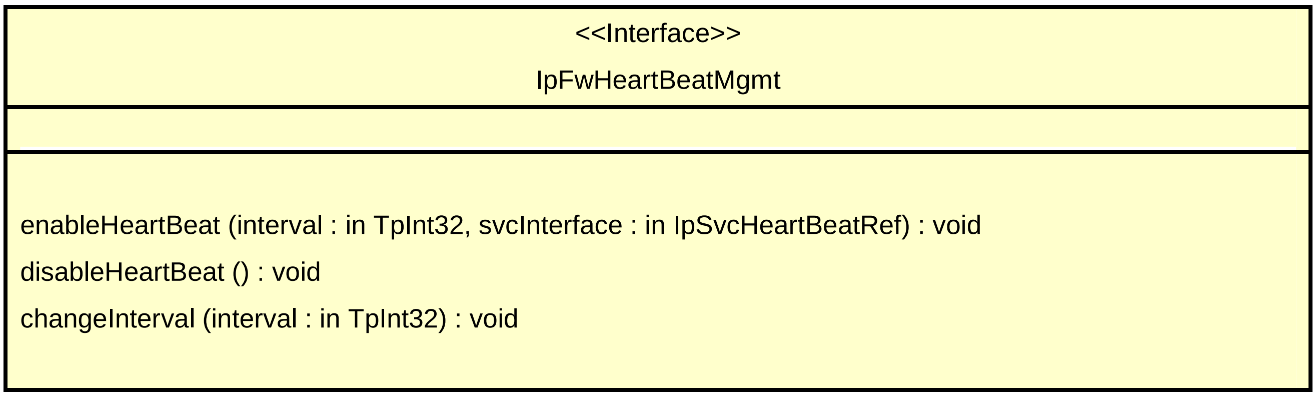
`TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE`

8.3.4.3 Interface Class IpFwHeartBeatMgmt

Inherits from: IpInterface.

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

If the IpFwHeartBeatMgmt interface is implemented by a Framework, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.



<<Interface>>

IpFwHeartBeatMgmt

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

disableHeartBeat () : void

changeInterval (interval : in TpInt32) : 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 time interval 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
No Parameters were identified for this method

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

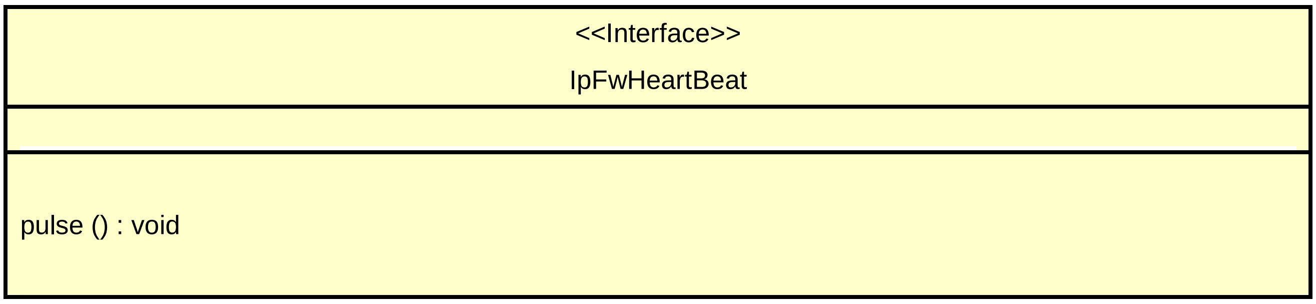
`TpCommonExceptions`

8.3.4.4 Interface Class IpFwHeartBeat

Inherits from: IpInterface.

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

If a Framework is capable of invoking IpSvcHeartBeatMgmt.enableHeartBeat(), it shall implement IpFwHeartBeat and the pulse() method.



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
No Parameters were identified for this method

Raises

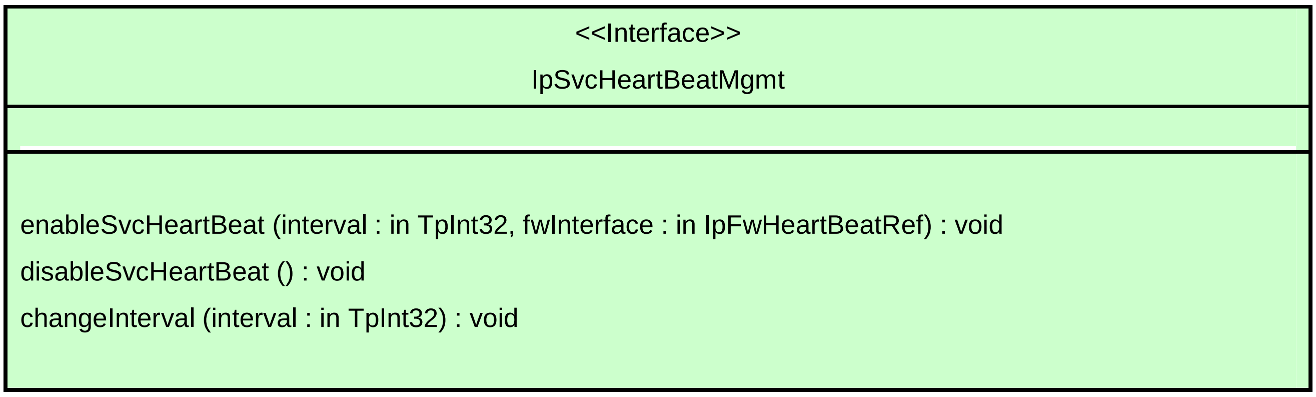
`TpCommonExceptions`

8.3.4.5 Interface Class IpSvcHeartBeatMgmt

Inherits from: IpInterface.

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

If the IpSvcHeartBeatMgmt interface is implemented by a Service, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.


<<Interface>>
IpSvcHeartBeatMgmt
enableSvcHeartBeat (interval : in TpInt32, fwInterface : in IpFwHeartBeatRef) : void
disableSvcHeartBeat () : void
changeInterval (interval : in TpInt32) : 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

No Parameters were identified for this method

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

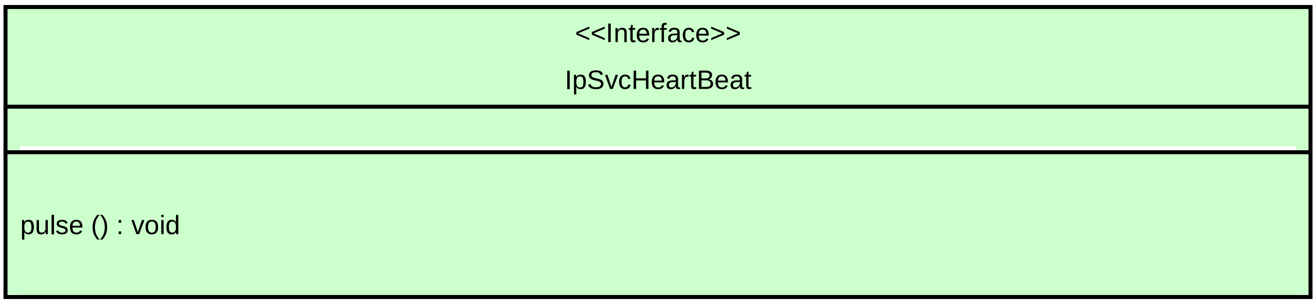
`TpCommonExceptions`

8.3.4.6 Interface Class IpSvcHeartBeat

Inherits from: IpInterface.

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

If a Service is capable of invoking IpFwHeartBeatMgmt.enableHeartBeat(), it shall implement IpSvcHeartBeat and the pulse() method.



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
No Parameters were identified for this method

Raises

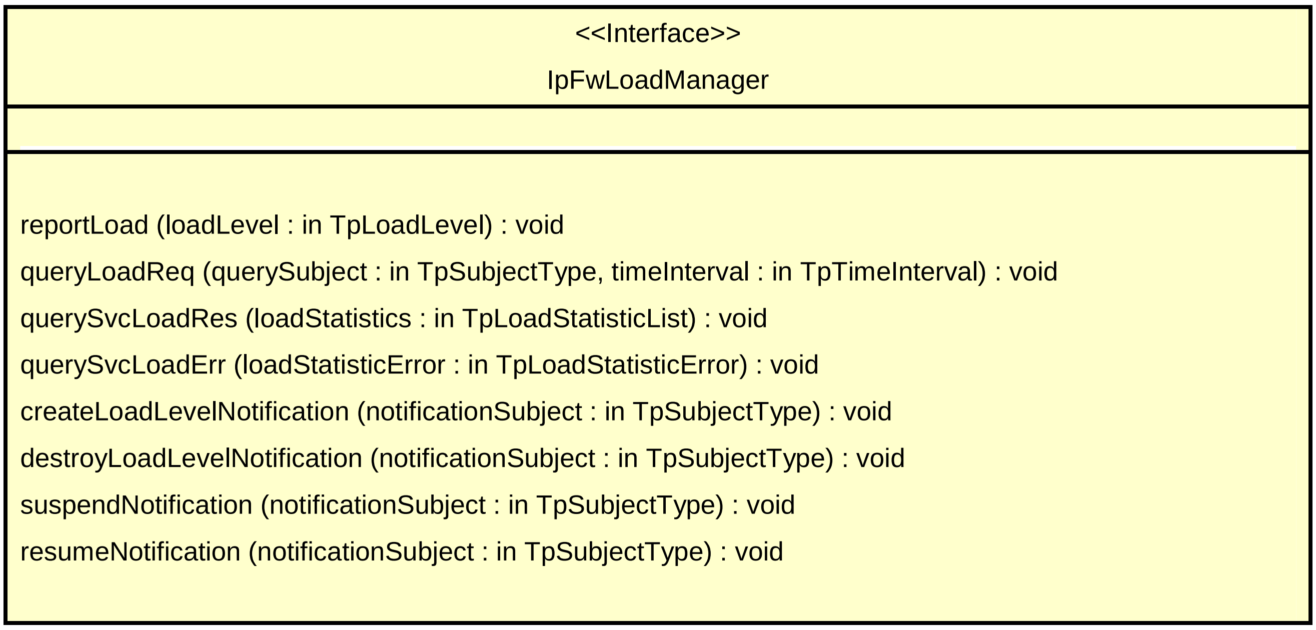
TpCommonExceptions

8.3.4.7 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.

If the IpFwLoadManager interface is implemented by a Framework, at least one of the methods shall be implemented as a minimum requirement. If load level notifications are supported, the createLoadLevelNotification() and destroyLoadLevelNotification() methods shall be implemented. If suspendNotification() is implemented, then resumeNotification() shall be implemented also. If a Framework is capable of invoking the IpSvcLoadManager.querySvcLoadReq() method, then it shall implement querySvcLoadRes() and querySvcLoadErr() methods in this interface.



<<Interface>>

IpFwLoadManager

reportLoad (loadLevel : in TpLoadLevel) : void

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

querySvcLoadRes (loadStatistics : in TpLoadStatisticList) : void

querySvcLoadErr (loadStatisticError : in TpLoadStatisticError) : void

createLoadLevelNotification (notificationSubject : in TpSubjectType) : void

destroyLoadLevelNotification (notificationSubject : in TpSubjectType) : void

suspendNotification (notificationSubject : in TpSubjectType) : void

resumeNotification (notificationSubject : 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 application) 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 instance 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

`notificationSubject : in TpSubjectType`

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

Raises

`TpCommonExceptions`

Method

`destroyLoadLevelNotification()`

The service instance 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

`notificationSubject : 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

`notificationSubject : 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

notificationSubject : 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

8.3.4.8 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 IpAccess interface.

If the IpSvcLoadManager interface is implemented by a Service, at least one of the methods shall be implemented as a minimum requirement. If load level notifications are supported, then loadLevelNotification() shall be implemented. If a the Service is capable of invoking the IpFwLoadManager.queryLoadReq() method, then it shall implement queryLoadRes() and queryLoadErr() methods in this 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 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 or framework which has 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

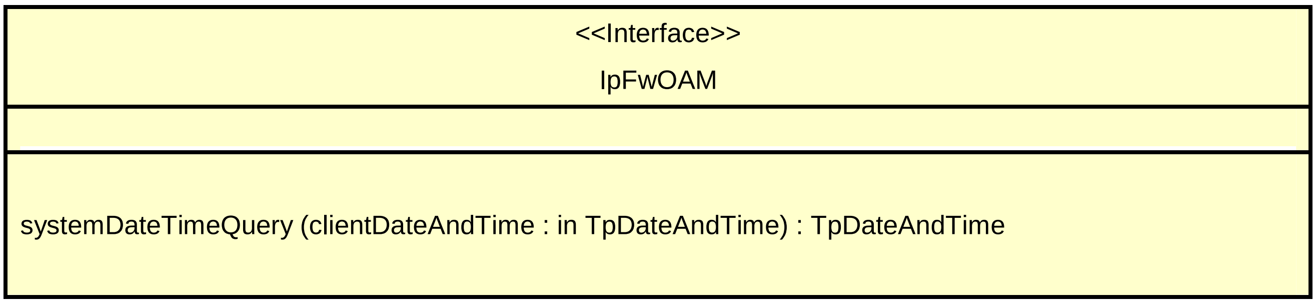
`TpCommonExceptions`

8.3.4.9 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.

This interface and ~~its~~the systemDateTimeQuery() method are optional.


<<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 error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

TpDateAndTime

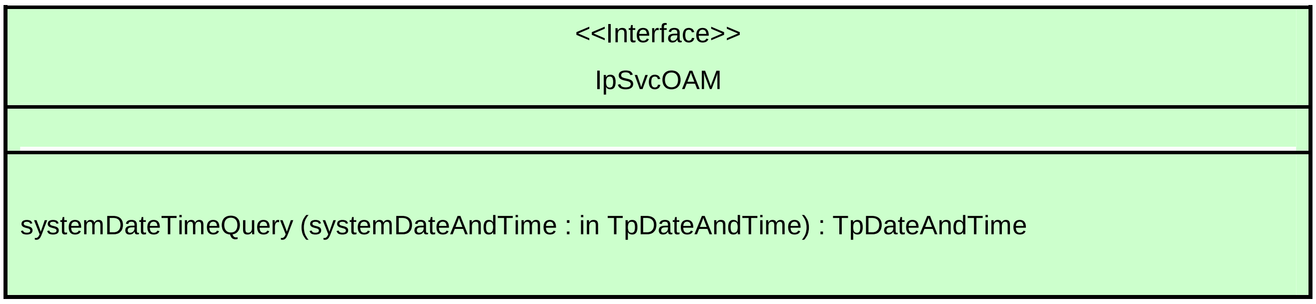
Raises

TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT

8.3.4.10 Interface Class IpSvcOAM

Inherits from: IpInterface.

This interface and ~~its~~the systemDateTimeQuery() method are optional.


<<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 error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

`TpDateAndTime`

Raises

`TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT`

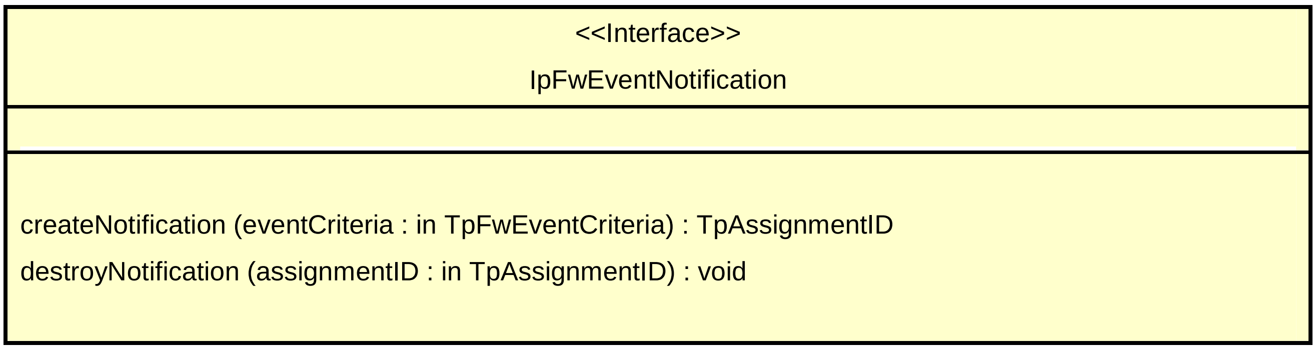
8.3.5 Event Notification Interface Classes

8.3.5.1 Interface Class IpFwEventNotification

Inherits from: IpInterface.

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

If Event Notifications are supported by a Framework, this interface and ~~all its~~the createNotification() and destroyNotification() methods shall be supported.


<<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 return the error code P_INVALID_ASSIGNMENT_ID.

Raises

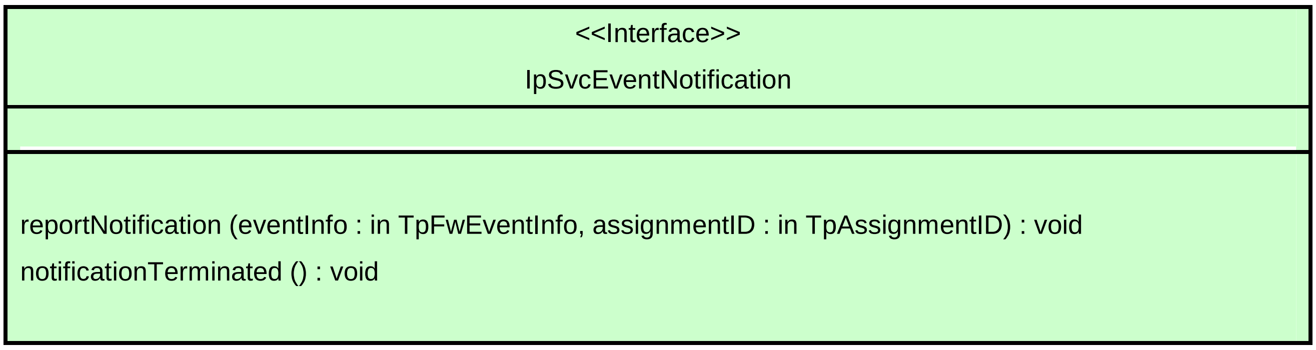
TpCommonExceptions,P_INVALID_ASSIGNMENT_ID

8.3.5.2 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.

If Event Notifications are supported by a Service, this interface and ~~all its~~the reportNotification() and notificationTerminated() methods shall be supported.


<<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`