3GPP TSG CN Plenary Meeting #14   **NP-010596**
Kyoto, Japan, 12-14 December 2001

Source: **CN5 (OSA)**

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

Agenda item: **8.5**

Document for: **Decision**

Doc-1st-Level	Spec	CR	F	Pha	Subject	Cat	Ver Cur	Ver -New	Doc-2nd-Level	Workit em
NP-010596	29.198-03	021		Rel-4	**Replace Out Parameters with Return Types**	F	4.2.0	4.3.0	N5-010563	OSA1
NP-010596	29.198-03	022		Rel-4	**Correctionto Framework (FW)**	F	4.2.0	4.3.0	N5-011241	OSA1

CR-Form-v4

CHANGE REQUEST

| ⌘ | **29.198-03** CR **021** | ⌘ | ev | **-** | ⌘ | Current version: | **4.2.0** | ⌘ |

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

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

Title:	⌘	Replacing Out Parameters with Return Types

Source:	⌘	CN5

Work item code: ⌘	OSA1		*Date:* ⌘	19/07/2001

Category:	⌘	**F**		*Release:* ⌘	REL-4

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

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

Reason for change:	⌘	At CN5 and CN it was agreed that Out-parameters should be removed from methods as a means of returning information, to be replaced by Return Types, in line with commonly used programming practice

Summary of change: ⌘	For each method, replace the return parameter TpResult with: 'void' if the method has no out-parameter; or the type of the out-parameter if the method has an out-parameter, and delete the out-parameter from the method.

Consequences if not approved:	⌘	If this particular CR is not agreed, TS 29.198-3 is out of sync. with the other parts of TS 29.198. If the related batch of CRs is not agreed, OSA will have a limited acceptance among the application development community, since it will be more difficult to implement. This presents a risk to the return on investment in development of OSA.

Clauses affected:	⌘	5, 8, 12, Annex B

Other specs affected:	⌘	**X** Other core specifications	⌘	All other parts of TS 29.198 Rel-4
		☐ Test specifications		
		☐ O&M Specifications		

Other comments:	⌘	

How to create CRs using this form:

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

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

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

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

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

5 The Base Interface Specification

5.1 Interface Specification Format

This section defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

5.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for client applications are denoted by classes with name Ip<name>. The callback interfaces to the applications are denoted by classes with name IpApp<name>. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>

5.1.2 Method descriptions

Each method (API method "call") is described. All methods in the API return a value of type TpResult, indicating, amongst other things, if the method invocation was sucessfully executed or not.

Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a 'Req' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a 'Res' or 'Err' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant IpApp<name> or IpSvc<name> interfaces to provide the callback mechanism.

5.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.
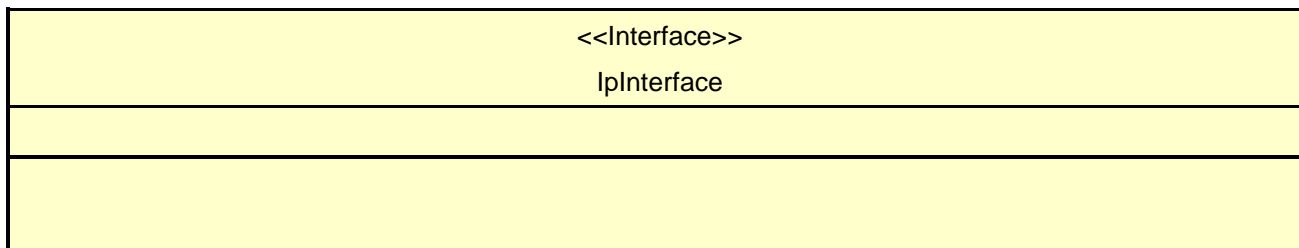
5.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

5.2 Base Interface

5.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.

<<Interface>>
IpInterface

5.3　Service Interfaces

5.3.1　Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.
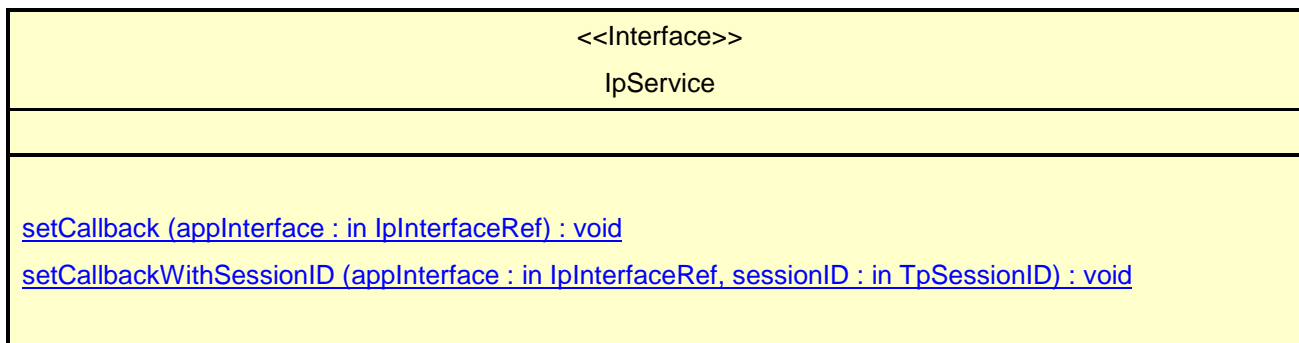
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

5.4　Generic Service Interface

5.4.1　Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.

<<Interface>>
IpService
setCallback (appInterface : in IpInterfaceRef) : void setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : void

Method
setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application. It is not allowed to invoke this method on an interface that uses SessionID's.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

Raises

TpCommonExceptions

Method
setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg. It is not allowed to invoke this method on an interface that does not uses SessionID's.

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

Specifies the session for which the service can invoke the application's callback interface.

Raises

TpCommonExceptions, P_INVALID_SESSION_ID

Parameters

appInterface : in IpInterfaceRef

Specifies a reference to the application interface, which is used for callbacks

sessionID : in TpSessionID

===========================**Next changed section**===============================

8   Framework-to-Application Interface Classes

8.1   Trust and Security Management Interface Classes

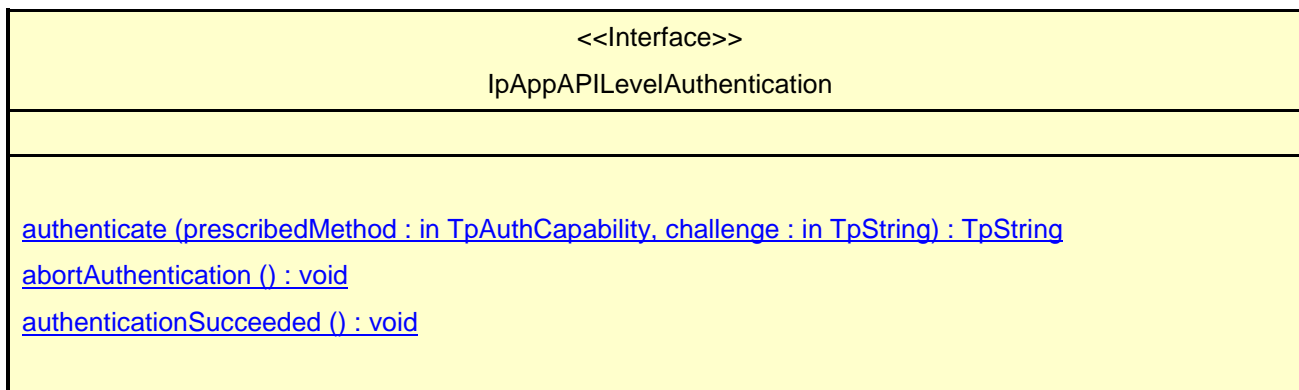The Trust and Security Management Interfaces provide:

-    the first point of contact for an application to access a Home Environment;

-    the authentication methods for the application and Home Environment to perform an authentication protocol;

-    the application with the ability to select a service capability feature to make use of;

-    the application with a portal to access other Framework interfaces.

The process by which the application accesses the Home Environment 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.

8.1.1   Interface Class IpAppAPILevelAuthentication

Inherits from: IpInterface.

<<Interface>>
IpAppAPILevelAuthentication
authenticate (prescribedMethod : in TpAuthCapability, challenge : in TpString) : TpString   abortAuthentication () : void   authenticationSucceeded () : void

Method
authenticate()

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

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

prescribedMethod : in TpAuthCapability

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

challenge : in TpString

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

Returns

TpString

Method
abortAuthentication()

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

Parameters
No Parameters were identified for this method

Method
authenticationSucceeded()

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

Parameters
No Parameters were identified for this method

8.1.2   Interface Class IpAppAccess

Inherits from: IpInterface.

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

<<Interface>>
IpAppAccess

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

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

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

Method
signServiceAgreement()

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

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

Parameters

serviceToken : in TpServiceToken

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

agreementText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

Returns

TpString

Method
terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpString

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

Method
terminateAccess()

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

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

Parameters

terminationText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

digitalSignature : in TpString

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

8.1.3   Interface Class IpInitial

Inherits from: IpInterface.

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

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

Method
initiateAuthentication()

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

Returns <fwDomain> : This provides the application domain 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 enterprise domain.

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 application domain uses this interface to authenticate with the framework.

Parameters

appDomain : in TpAuthDomain

This identifies the application 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). It is used to identify the enterprise 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 application. 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).
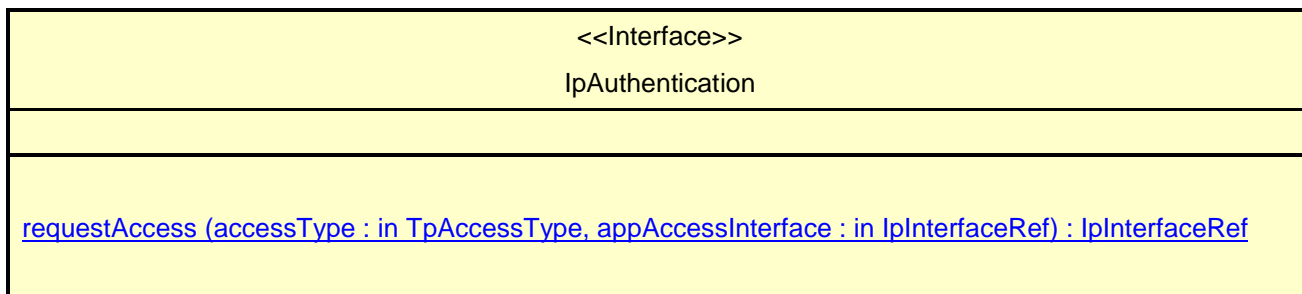
authType : in TpAuthType

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 Authentication 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 appDomain and fwDomain authInterface parameters are references to interfaces of type Ip(App)APILevelAuthentication. If P_AUTHENTICATION is selected, the authInterface parameters are refereces to interfaces of type Ip(App)Authentication which is used when an underlying distibution technology authentication mechanism is used.

Returns

TpAuthDomain

Raises

TpCommonExceptions,P_INVALID_DOMAIN_ID,P_INVALID_INTERFACE_TYPE,P_INVALID_AUTH_TYPE

8.1.4   Interface Class IpAuthentication

Inherits from: IpInterface.

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

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

Method
requestAccess()

Once application and framework are authenticated, the client application invokes the requestAccess operation on the IpAuthentication or IpAPILevelAuthentication interface. This allows the client application 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 application 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 application to call the access interface of the framework.

Parameters

accessType : in TpAccessType

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

appAccessInterface : in IpInterfaceRef

This provides the reference for the framework to call the access interface of the client application. 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_INTERF
ACE_TYPE**

8.1.5 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

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

<<Interface>>
IpAPILevelAuthentication
selectEncryptionMethod (authCaps : in TpAuthCapabilityList) : TpAuthCapability authenticate (prescribedMethod : in TpAuthCapability, challenge : in TpString) : TpString abortAuthentication () : void authenticationSucceeded () : void

Method
selectEncryptionMethod()

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

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

Parameters

authCaps : in TpAuthCapabilityList

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

Returns

TpAuthCapability

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_NO_ACCEPTABLE_AUTH_CAPABILITY

Method
authenticate()

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

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

Parameters

prescribedMethod : in TpAuthCapability

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

challenge : in TpString

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

Returns

TpString

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AUTH_CAPABILITY

Method
abortAuthentication()

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

Parameters
No Parameters were identified for this method

Raises

TpCommonExceptions,P_ACCESS_DENIED

Method
authenticationSucceeded()

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

Parameters
No Parameters were identified for this method

Raises

TpCommonExceptions,P_ACCESS_DENIED

8.1.6   Interface Class IpAccess

Inherits from: IpInterface.

<<Interface>>
IpAccess
obtainInterface (interfaceName : in TpInterfaceName) : IpInterfaceRef   obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, appInterface : in IpInterfaceRef) : IpInterfaceRef   selectService (serviceID : in TpServiceID) : TpServiceToken   signServiceAgreement (serviceToken : in TpServiceToken, agreementText : in TpString, signingAlgorithm : in TpSigningAlgorithm) : TpSignatureAndServiceMgr   terminateServiceAgreement (serviceToken : in TpServiceToken, terminationText : in TpString, digitalSignature : in TpString) : void   endAccess (endAccessProperties : in TpEndAccessProperties) : void

Method
obtainInterface()

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

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

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).

appInterface : in IpInterfaceRef

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

Returns

IpInterfaceRef

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_INTERFACE_NAME,P_INVALID_INT ERFACE_TYPE

Method
selectService()

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

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

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: TpString;
serviceMgrInterface: IpInterfaceRef;
};
The digitalSignature is the signed version of a hash of the service token and agreement text given by the client application.
The serviceMgrInterface is a reference to the service manager interface for the selected service.

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_SER
VICE_TOKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED**

Method
terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpString

This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same
as the signing algorithm given when the service agreement was signed using signServiceAgreement().The framework
uses this to 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_SIGN
ATURE**

Method
endAccess()

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

Parameters

endAccessProperties : in TpEndAccessProperties

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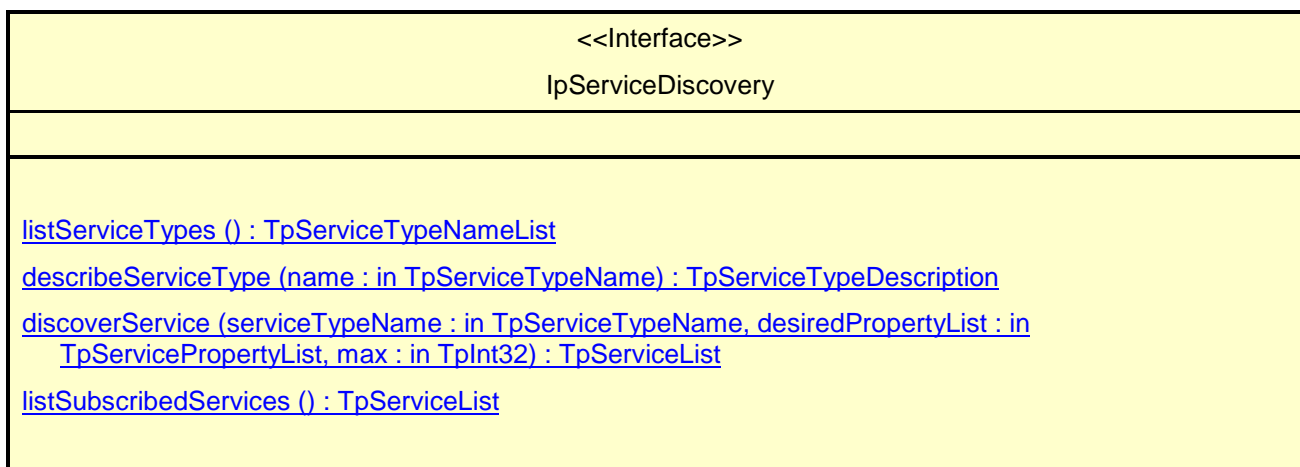error code (P_INVALID_PROPERTY) is returned.

Raises

TpCommonExceptions,P_ACCESS_DENIED, P_INVALID_PROPERTY

8.2 Service Discovery Interface Classes

8.2.1 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.

<<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 enabled or disabled.

Parameters

name : in TpServiceTypeName

The name of the service type to be described.

· If the "name" is malformed, then the P_ILLEGAL_SERVICE_TYPE exception is raised.

· If the "name" does not exist in the repository, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

Returns

TpServiceTypeDescription

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVI CE_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 property {name, mode and value list} tuples 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 property {name, mode and value list} tuples 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_SERVI
CE_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 property {name, mode and value list} tuples associated with the service.
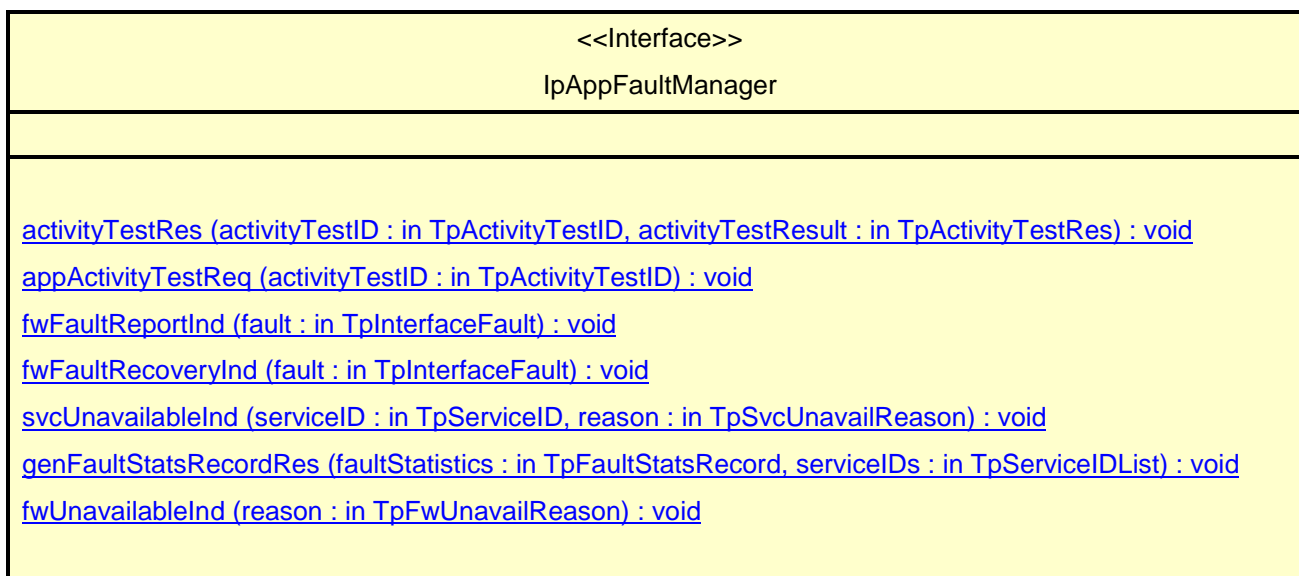
Parameters
No Parameters were identified for this method

Returns

TpServiceList

Raises

TpCommonExceptions,P_ACCESS_DENIED

8.3   Integrity Management Interface Classes

8.3.1   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

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

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

8.3.2   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.

<<Interface>>
IpFaultManager
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

Method
activityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

svcID : in TpServiceID

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID

Method
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 the indicated service (either due to a failure in the client application or in the service). On receipt of this request, the framework should take the appropriate corrective action. The framework assumes that the session between this client application and service instance is to be closed and updates its own records appropriately as well as attempting to inform the service instance and/or its administrator. Attempts by the client application to continue using this session should be rejected.

Parameters

serviceID : in TpServiceID

Identifies the service that the application can no longer use.

Raises

TpCommonExceptions ,P_INVALID_SERVICE_ID

Method
genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

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

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions ,P_INVALID_SERVICE_ID

8.3.3   Interface Class IpAppHeartBeatMgmt

Inherits from: IpInterface.

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

<<Interface>>
IpAppHeartBeatMgmt
enableAppHeartBeat (duration : in TpDuration, fwInterface : in IpHeartBeatRef, session : in TpSessionID) : void disableAppHeartBeat (session : in TpSessionID) : void changeTimePeriod (duration : in TpDuration, session : in TpSessionID) : void
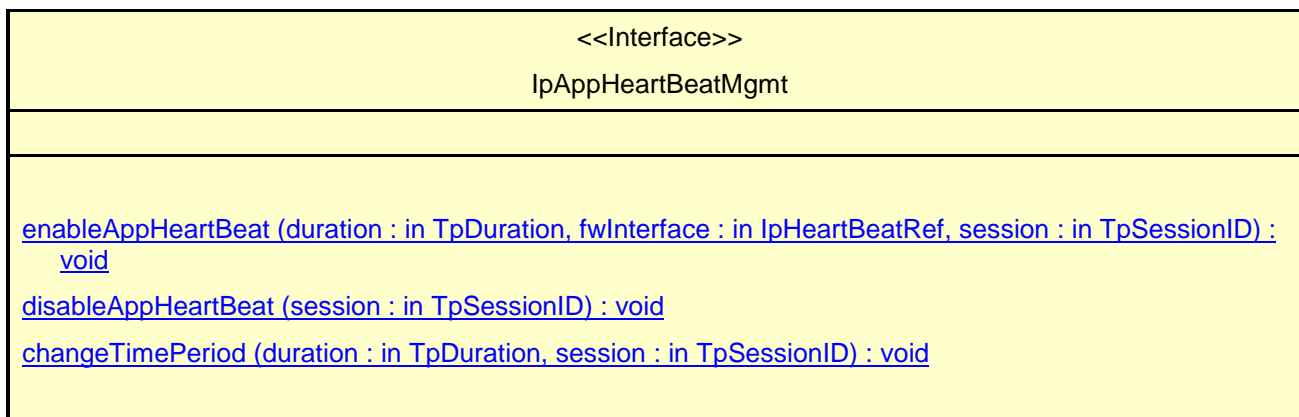
Method
enableAppHeartBeat()

With this method, the framework registers at the client application for heartbeat supervision of itself.

Parameters

duration : in TpDuration

The time interval in milliseconds between the heartbeats.

fwInterface : in IpHeartBeatRef

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

session : in TpSessionID

Identifies the heartbeat session.

Method
disableAppHeartBeat()

Allows the stop of the heartbeat supervision of the application.

Parameters

session : in TpSessionID

Identifies the heartbeat session.

Method
changeTimePeriod()

Allows the administrative change of the heartbeat period.

Parameters

duration : in TpDuration

The time interval in milliseconds between the heartbeats.

session : in TpSessionID

Identifies the heartbeat session.

8.3.4 Interface Class IpAppHeartBeat

Inherits from: IpInterface.

The Heartbeat Application interface is used by the Framework to supervise the Application. The return of the TpResult is interpreted as a heartbeat response.

<<Interface>>
IpAppHeartBeat
send (session : in TpSessionID) : void
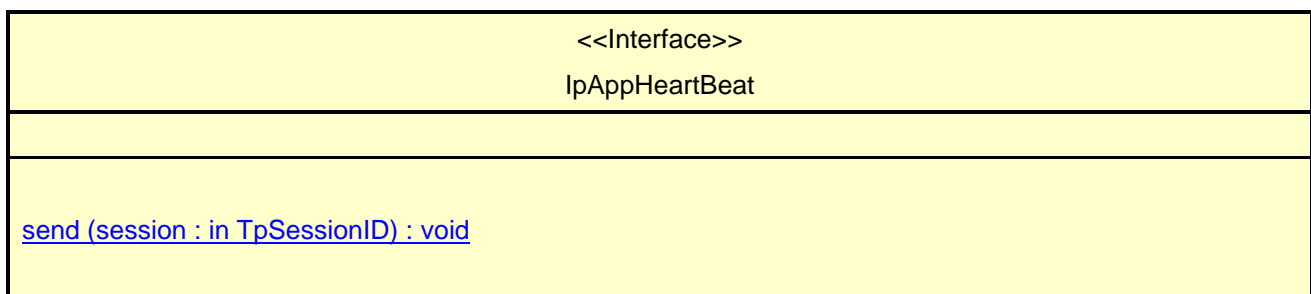
Method
send()

This is the method the framework uses in case it supervises the client application. The sender must raise an exception if no result comes back after a certain, user-defined time..
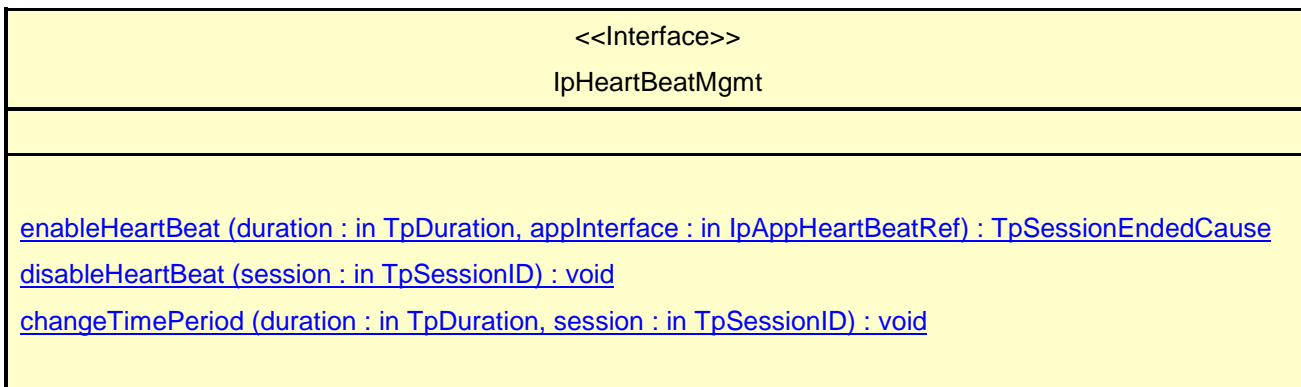
Parameters

session : in TpSessionID

Identifies the heartbeat session.

8.3.5   Interface Class IpHeartBeatMgmt

Inherits from: IpInterface.

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

<<Interface>>
IpHeartBeatMgmt
enableHeartBeat (duration : in TpDuration, appInterface : in IpAppHeartBeatRef) : TpSessionEndedCause disableHeartBeat (session : in TpSessionID) : void changeTimePeriod (duration : in TpDuration, session : in TpSessionID) : void

Method
enableHeartBeat()

With this method, the client application registers at the framework for heartbeat supervision of itself.

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

Parameters

duration : in TpDuration

The duration in milliseconds between the heartbeats.

appInterface : in IpAppHeartBeatRef

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

Returns

TpSessionEndedCause

Raises

TpCommonExceptions,P_INVALID_SESSION_ID

Method
disableHeartBeat()

Allows the stop of the heartbeat supervision of the application.

Parameters

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpCommonExceptions,P_INVALID_SESSION_ID

Method
changeTimePeriod()

Allows the administrative change of the heartbeat period.

Parameters

duration : in TpDuration

The time interval in milliseconds between the heartbeats.

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpCommonExceptions,P_INVALID_SESSION_ID

8.3.6   Interface Class IpHeartBeat

Inherits from: IpInterface.

The Heartbeat Framework interface is used by the client application to supervise the Framework.

<<Interface>>
IpHeartBeat
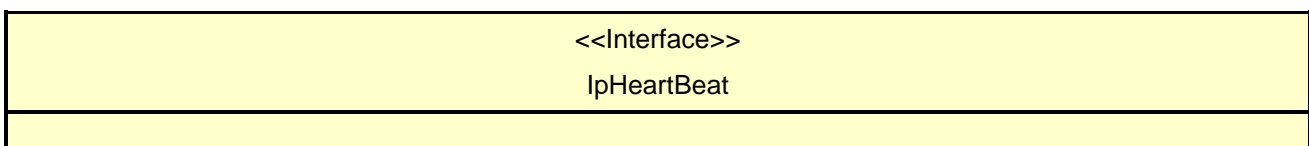
send (session : in TpSessionID) : void

Method
send()

This is the method the client application uses in case it supervises the framework. The sender must raise an exception if no result comes back after a certain, user-defined time.

Parameters

session : in TpSessionID

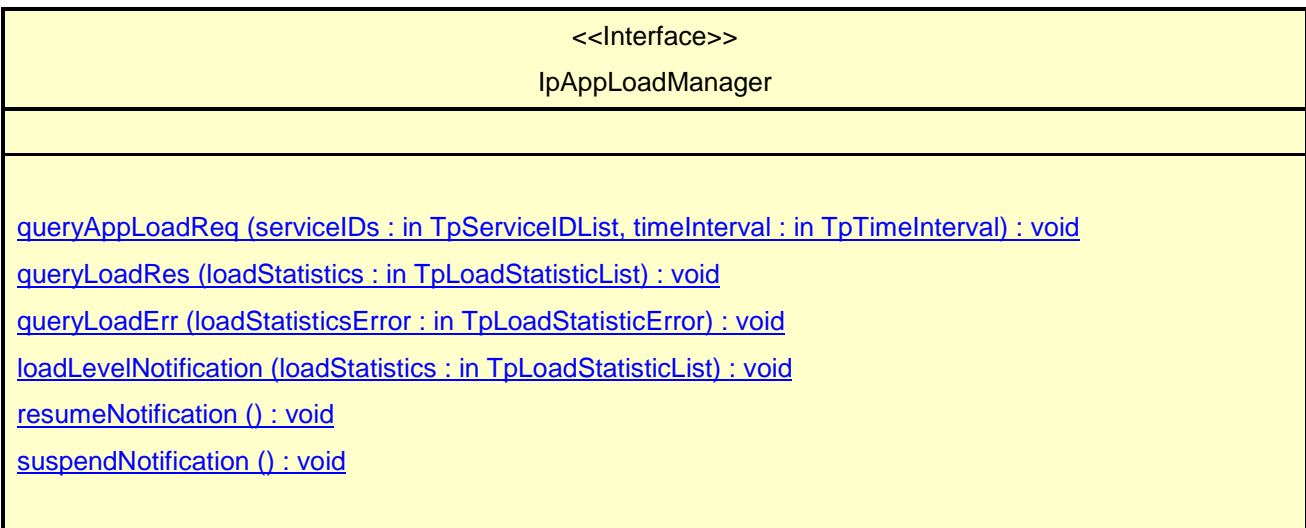Identifies the heartbeat session. In general, the application has only one session.

Raises

TpCommonExceptions,P_INVALID_SESSION_ID

8.3.7 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 (serviceIDs : in TpServiceIDList, 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 or for individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

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

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

8.3.8   Interface Class IpLoadManager

Inherits from: IpInterface.

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

<<Interface>>
IpLoadManager
reportLoad (loadLevel : in TpLoadLevel) : void queryLoadReq (serviceIDs : in TpServiceIDList, timeInterval : in TpTimeInterval) : void queryAppLoadRes (loadStatistics : in TpLoadStatisticList) : void queryAppLoadErr (loadStatisticsError : in TpLoadStatisticError) : void registerLoadController (serviceIDs : in TpServiceIDList) : void unregisterLoadController (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 individual services used by the application.

Parameters

serviceIDs : in TpServiceIDList

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

timeInterval : in TpTimeInterval

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

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
registerLoadController()

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

Parameters

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID

Method
unregisterLoadController()

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

Parameters

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID

Method
resumeNotification()

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

Parameters

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

Method
suspendNotification()

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

Parameters

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_SERVICE_NOT_ENABLED

8.3.9   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 of the OSA APIs.

<<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

8.3.10   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
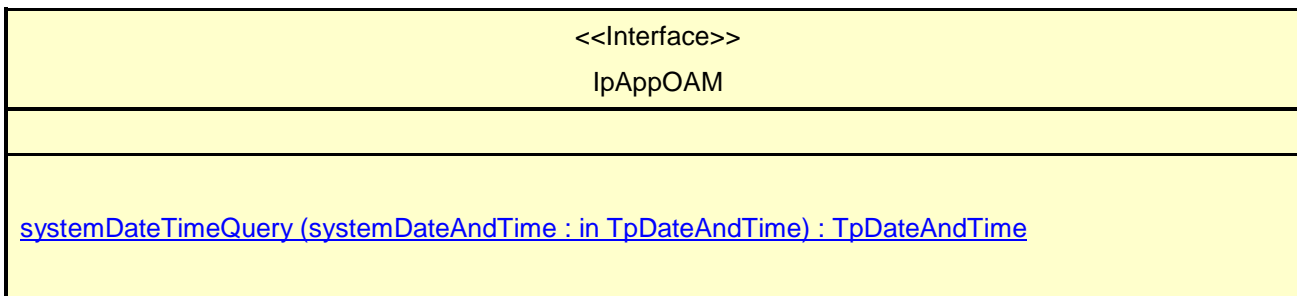
Method
systemDateTimeQuery()

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

Parameters

systemDateAndTime : in TpDateAndTime

This is the system date and time of the framework.

Returns

TpDateAndTime

8.4 Event Notification Interface Classes

8.4.1 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

8.4.2 Interface Class IpEventNotification

Inherits from: IpInterface.

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

<<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

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_ASSIGNMENT_ID

=========================Next changed section==============================

12 Framework-to-Service Interface Classes

12.1 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 section enables the service supplier to obtain a list of all the service types supported by the framework and their associated sets of service property values.

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

12.1.1 Interface Class IpFwServiceRegistration

Inherits from: IpInterface.

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

<<Interface>>
IpFwServiceRegistration
registerService (serviceTypeName : in TpServiceTypeName, servicePropertyList : in TpServicePropertyList) : TpServiceID announceServiceAvailability (serviceID : in TpServiceID, serviceFactoryRef : in service_registration::IpSvcFactoryRef) : 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 . A service-ID is returned to the service supplier when a service is registered in the Framework. The service-ID is the handle with which the service supplier can identify the registered service when needed (e.g. for withdrawing it). The service-ID is only meaningful in the context of the Framework that generated it.

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

servicePropertyList : in TpServicePropertyList

The "servicePropertyList" parameter is a list of property name and property value pairs. They describe the service being registered. This description typically covers behavioral, non-functional and non-computational aspects of the service. Service properties are marked "mandatory" or "readonly". These property mode attributes have the following semantics:
　　a. mandatory - a service associated with this service type must provide an appropriate value for this property when registering.
　　b. readonly - this modifier indicates that the property is optional, but that once given a value, subsequently it may not be modified.
　　Specifying both modifiers indicates that a value must be provided and that subsequently it may not be modified. An example of such properties are those which form part of a service agreement and hence cannot be modified by service suppliers during the life time of service.
　　If the type of any of the property values is not the same as the declared type (declared in the service type), then a P_PROPERTY_TYPE_MISMATCH exception is raised. If an attempt is made to assign a dynamic property value to a readonly property, then the P_READONLY_DYNAMIC_PROPERTY exception is raised. If the "servicePropertyList" parameter omits any property declared in the service type with a mode of mandatory, then a P_MISSING_MANDATORY_PROPERTY exception is raised. If two or more properties with the same property name are included in this parameter, the P_DUPLICATE_PROPERTY_NAME exception is raised.

Returns

TpServiceID

Raises

TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_PROPERTY_T YPE_MISMATCH,P_DUPLICATE_PROPERTY_NAME,
**　P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE,P_MISSING_MANDATORY_PROP ERTY**

Method
announceServiceAvailability()

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

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

Parameters

serviceID : in TpServiceID

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

serviceFactoryRef : in service_registration::IpSvcFactoryRef

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

Raises

**TpCommonExceptions,P_ILLEGAL_SERVICE_ID,P_UNKNOWN_SERVICE_ID,P_INVALID_IN
TERFACE_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.
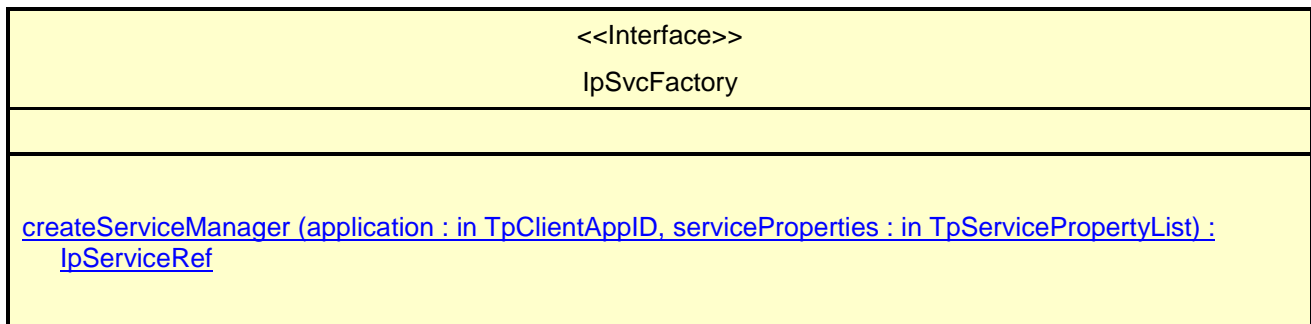
Returns

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

# 12.2	Service Factory Interface Classes

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

12.2.1   Interface Class IpSvcFactory

Inherits from: IpInterface.

<<Interface>>
IpSvcFactory
createServiceManager (application : in TpClientAppID, serviceProperties : in TpServicePropertyList) : IpServiceRef

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.

Returns

IpServiceRef

Raises

TpCommonExceptions,P_INVALID_PROPERTY

========================**Next changed section**===============================

B.9 All Interfaces

All methods on IpApp interfaces no longer throw exceptions.

All methods on the other interfaces throw TpCommonExceptions and individual, identified exceptions

All methods now return void or the former out parameter.

CR-Form-v5

CHANGE REQUEST

| ⌘ | **29.198-03** CR **022** | ⌘**rev** | **-** | ⌘ | Current version: | **4.2.0** | ⌘ |

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

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

Title:	⌘	Corrections to Framework Sequence Diagrams, Scope of Authentication, Service Registration, Service Agreement Management, and DataTypes
Source:	⌘	CN5
Work item code: ⌘	OSA1	**Date:** ⌘ 30/11/2001
Category:	⌘ **F**	**Release:** ⌘ REL-4

Use <u>one</u> 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 <u>TR 21.900</u>.

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

Reason for change: ⌘		Currently in the specification it is not clear what happens when a registerService() is used.

Clarification needed for signServiceAgreement() that if the signature is incorrect the serviceToken will be expired immediately.

The TpInterfaceName P_SERVICE_AGREEMENT_MANAGEMENT is missing.

The Event Notification Sequence diagram can't work: Framework has no chance to get the call back interface reference of IpAppEventNotification

The Heartbeat Management Sequence Diagram has an incorrectly implemented CR from a previous meeting
It needs to be made clearer that an application can have only one instance of a service
As a result of alignment with Parlay the range of domains allowed to invoke authenticate is expanded.
References to ServicePropertyMode are inappropriate and ServiceID is no longer a composite
Incorrect type defined for service instances in TpDomainID |
| **Summary of change:** ⌘ | | The method description of registerSevice() is changed to clarify the exact behaviour, the description of data type TpServiceTypeDescription is clarified.

The method description of signServiceAgreement() is changed to clarify that if the signature is incorrect the serviceToken will be expired immediately.

Method description of selectService updated to clarify that an application can have only one instance of a service;

Add P_SERVICE_AGREEMENT_MANAGEMENT in the defined values for TpInterfaceName. |

		Event Notification Sequence Diagram corrected. Heartbeat Management Sequence Diagram updated. clientAppID is replaced with domainID, thus allowing the client to be any one from client application, service instance, enterprise operator etc. In other words all domains defined in TpDomainID. ServicePropertyMode references removed and description of ServiceID corrected Change the type defined for service instances in TpDomainID to TpServiceInstanceID Removed Tp..Ref and Ip..RefRef types which are misleading and not used. Various editorial spelling errors corrected.
Consequences if not approved:	⌘	Interfaces with incomplete method descriptions will lead to interworking problems, or even to difficulties in implementing the specificaiton. Sequence Diagrams which show incorrect examples are misleading to developers. Incorrect data types make it impossible to correctly implement this specification correctly ETSI and Parlay have already corrected these errors, and 29.198-3 needs to aligned with these changes to allow development of common applications across the 3 platforms

Clauses affected:	⌘	6.1, 6.3, 7.1, .73, 8.1, 8.3, 10, Annex B

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

Other comments:	⌘	

How to create CRs using this form:

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

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

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

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

6 Framework Access Session API

6.1 Sequence Diagrams

6.1.1 Trust and Security Management Sequence Diagrams

6.1.1.1 Initial Access for trusted parties

The following figure shows a trusted party, typically within the same domain as the Framework, accessing the OSA Framework for the first time. Trusted parties don't need to be authenticated and after contacting the Initial interface the Framework will indicate that no further authentication is needed and that the application can immediately gain access to other framework interfaces and SCFs. This is done by invoking the requestAccess method.
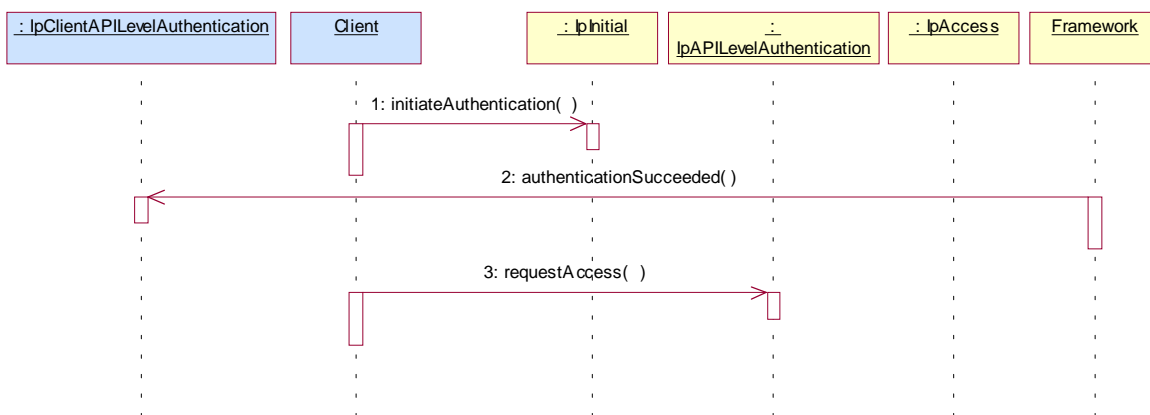
1: The Client invokes initiateAuthentication on the Framework's "public" (initial contact) interface to initiate the authentication process. It provides in turn a reference to its own authentication interface. The Framework returns a reference to its authentication interface.

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

3: 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.

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.
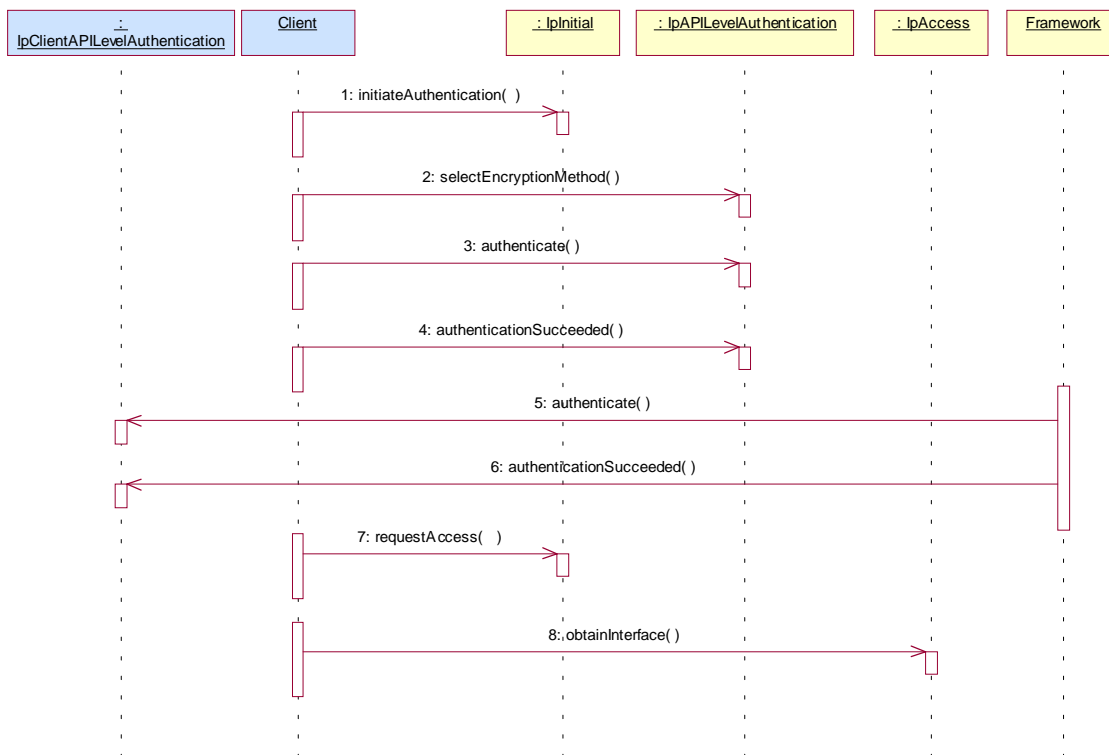
1: Initiate Authentication

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 Authenticaiton 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.

6.1.1.3 Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another using an underlying distribution technology mechanism.
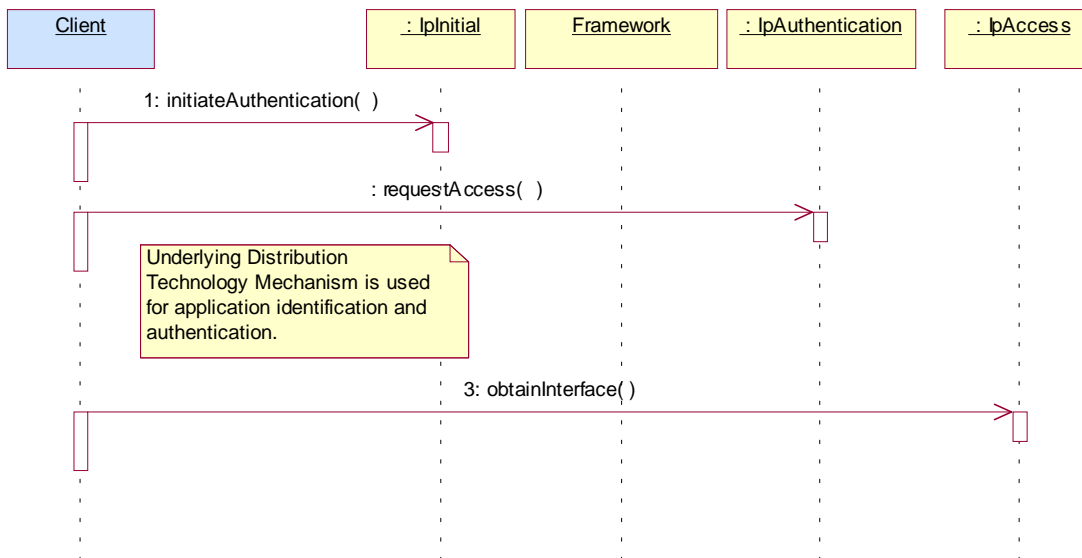
1: The client calls initiateAuthentication on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. In this case, the client selects to use the underlying distribution technology mechanism for identification and authentication.

2: The client invokes the requestAccess method on the Framework's Authentication interface. The Framework now uses the underlying distribution technology mechanism for identification and authentication of the client.

3: If the authentication was successful, the client can now invoke obtainInterface on the framework's Access interface to obtain a reference to its service discovery interface.

6.1.1.4 API Level Authentication

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

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

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

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

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

3) The application and Framework interact to authenticate each other. For an authentication method of P_OSA_ACCESS, this procedure consists of a number of challenge/ response exchanges. This authentication protocol is performed using the authenticate method on the API Level Authentication interface. P_OSA_ACCESS is based on CHAP, which is primarily a one-way protocol. Mutual authentication is achieved by the framework invoking the authenticate method on the client's APILevelAuthentication interface.

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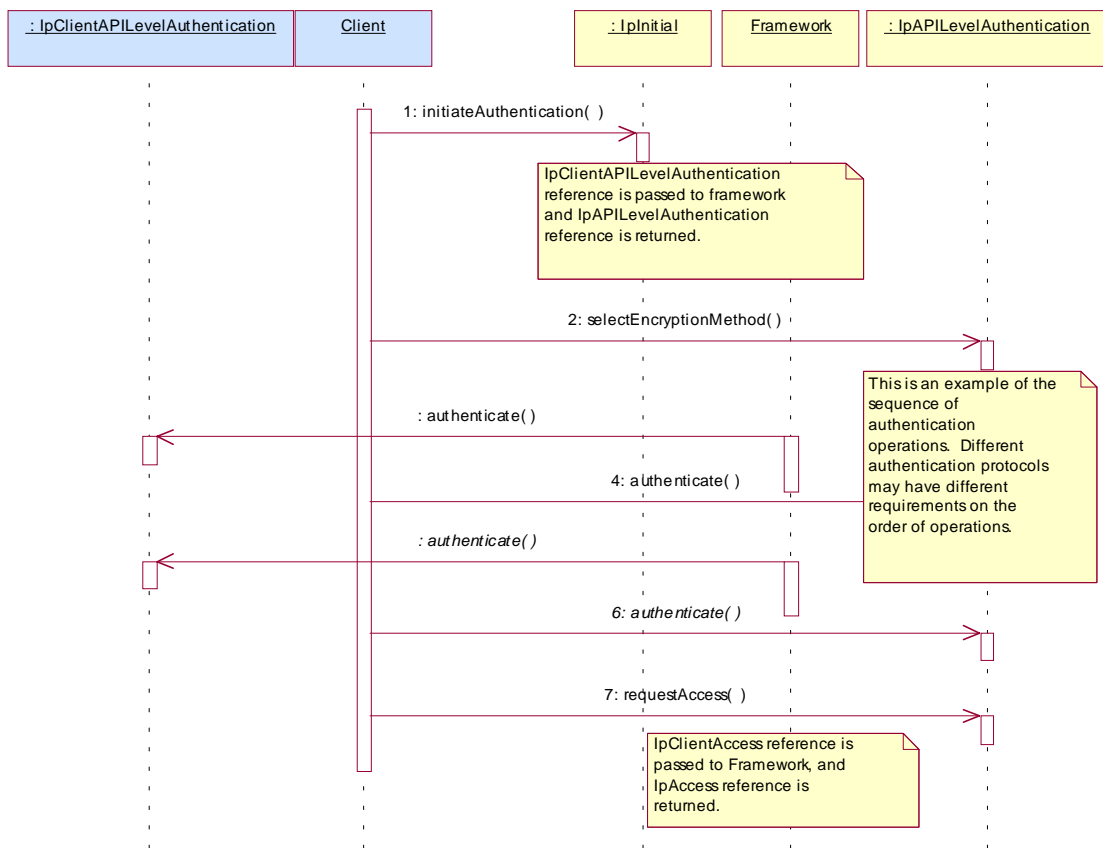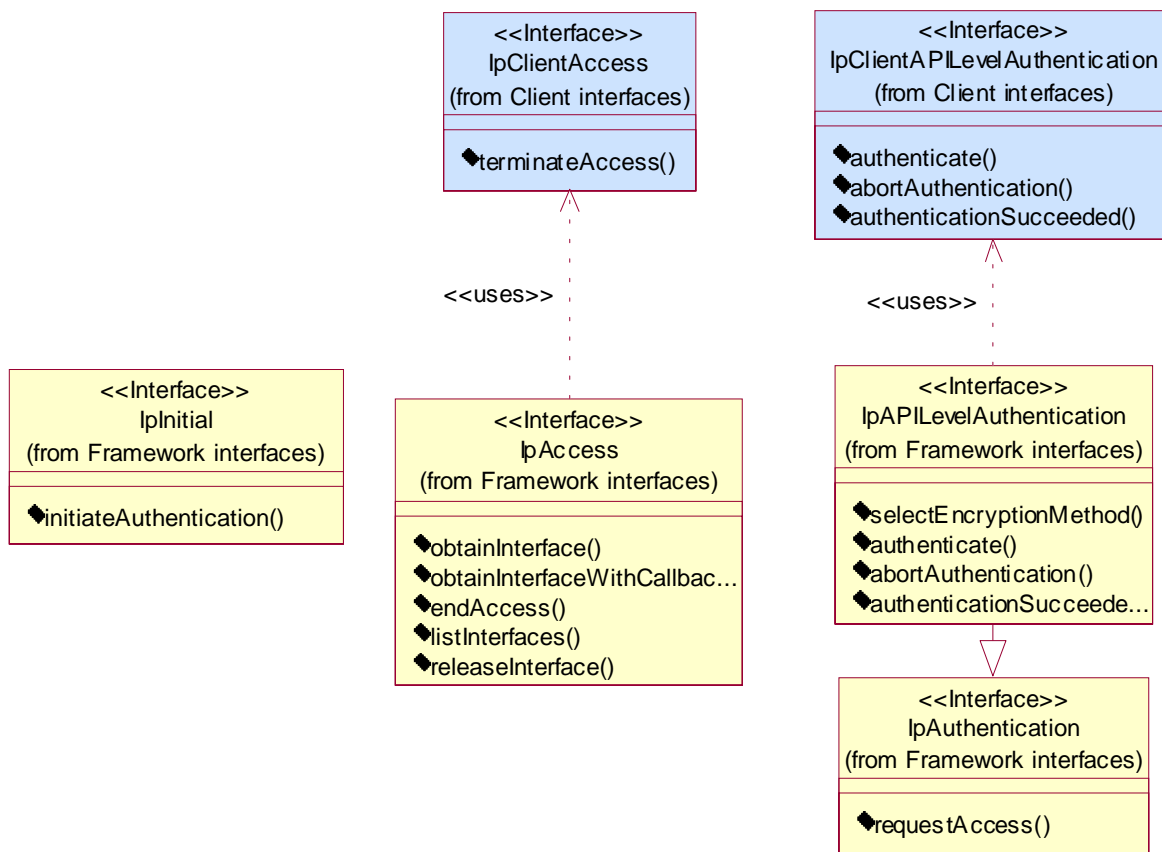

6.2 Class Diagrams

Figure: Trust and Security Management Package Overview

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.

<<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.
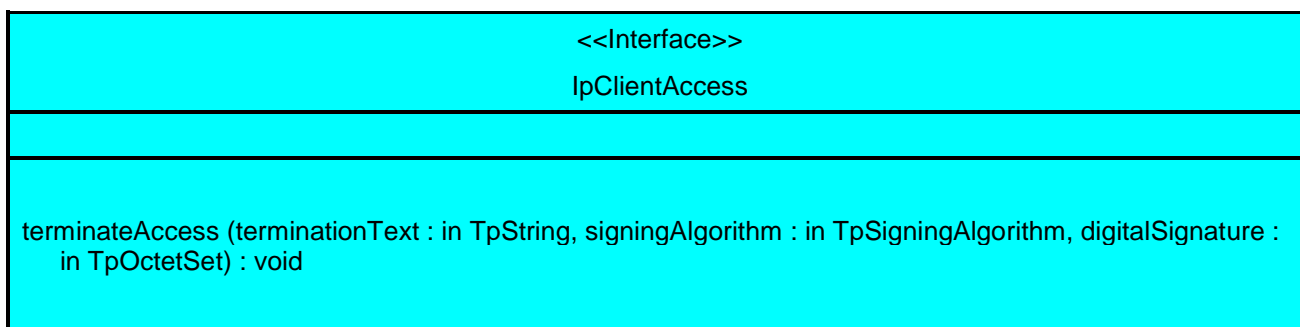
Parameters
No Parameters were identified for this method

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.

<<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.
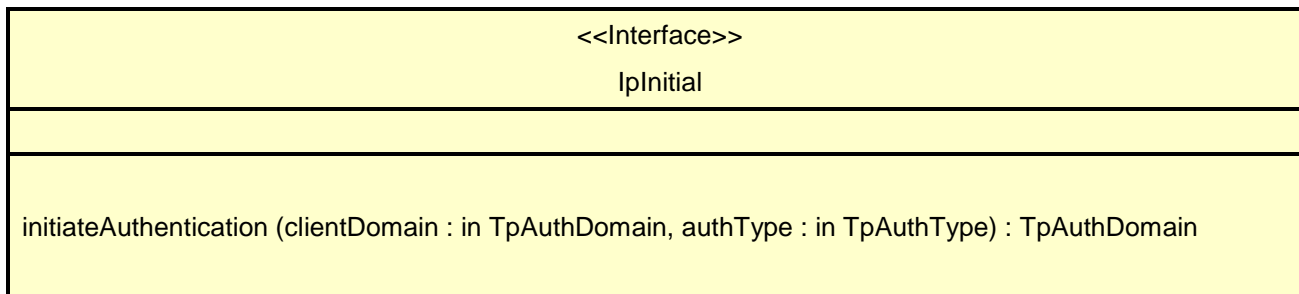
digitalSignature : in TpOctetSet

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.

<<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.

<<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.

<<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.

<<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 obtainInterfacesWithCallback 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_INT ERFACE_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

6.4 State Transition Diagrams

This section contains the State Transition Diagrams for the objects that implement the Framework interfaces on the gateway side. The State Transition Diagrams show the behaviour of these objects. For each state the methods that can be invoked by the client are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the client also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

6.4.1 Trust and Security Management State Transition Diagrams
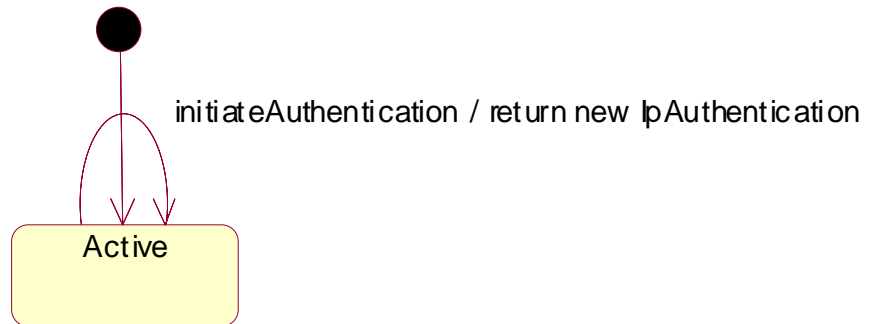
6.4.1.1 State Transition Diagrams for IpInitial

initiateAuthentication / return new IpAuthentication

Active

Figure : State Transition Diagram for IpInitial

6.4.1.1.1 Active State

6.4.1.2 State Transition Diagrams for IpAPILevelAuthentication
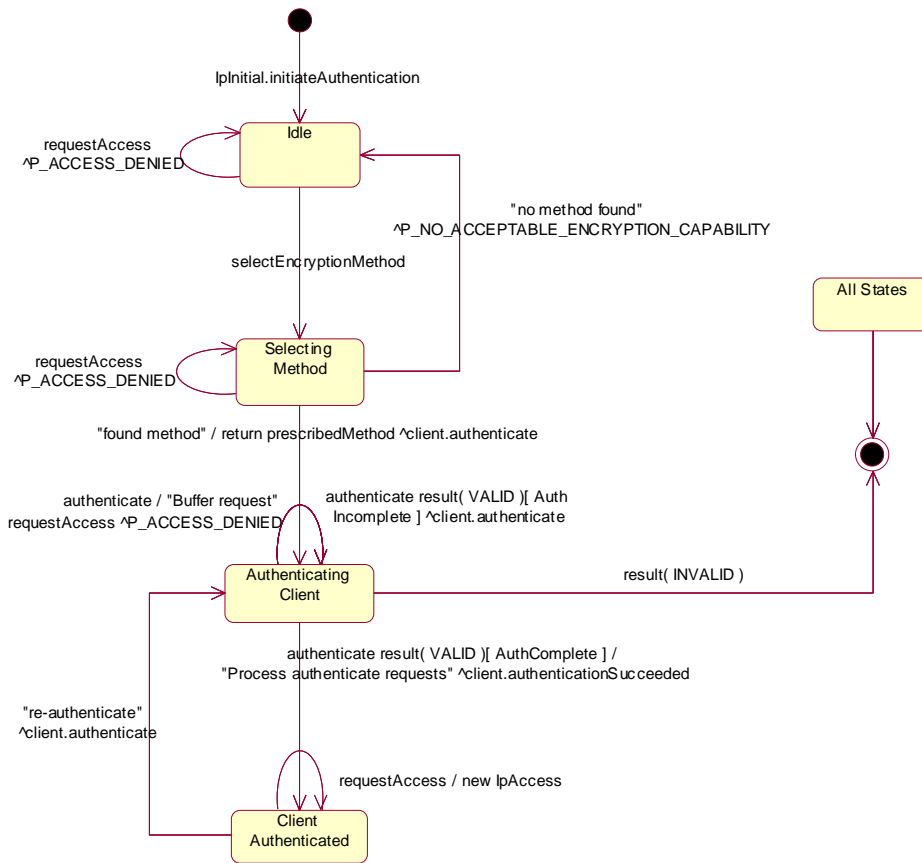
Figure : State Transition Diagram for IpAPILevelAuthentication

6.4.1.2.1   Idle State

When the client has invoked the IpInitial initiateAuthentication method, an object implementing the IpAPILevelAuthentication interface is created. The client now has to provide its encryption capabilities by invoking selectEncryptionMethod.

6.4.1.2.2   Selecting Method State

In this state the Framework selects the preferred encryption mechanism within the capability of the client. It is a policy of the framework (perhaps agreed off-line with the enterprise operator) whether the client has to be authenticated or not. In case no mechanism can be found the P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY exception is thrown and the Authentication object moves back to the IDLE state The client can now revisit its list of supported capabilities to identify whether it is complete. If it has no more encryption capabilities to use, then it must invoke abortAuthentication.

6.4.1.2.3   Authenticating Client State

When entering this state, the Framework requests the client to authenticate itself by invoking the Authenticate method on the client. In case the client requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework will either buffer the requests and respond when the client has been authenticated, or respond immediately, depending on policy. When the Framework has processed the response from the Authenticate request on the client, the response is analysed. If the response is valid but the authentication process is not yet complete, then another Authenticate request is sent to the client. If the response is valid and the authentication process has been completed, then a transition to the state ClientAuthenticated is made, the client is informed of its success by invoking authenticationSucceeded, then the framework begins to process any buffered authenticate requests. In case the response is not valid, the Authentication object is destroyed. This implies that the

client has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface.

6.4.1.2.4 Client Authenticated State

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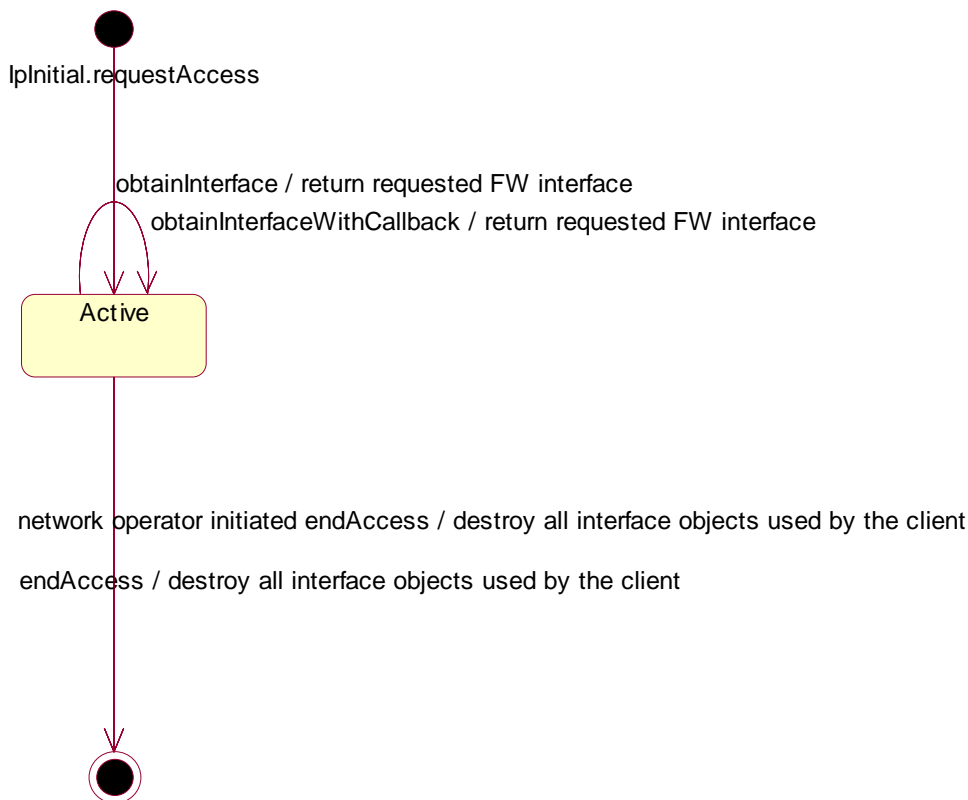
6.4.1.3 State Transition Diagrams for IpAccess

IpInitial.requestAccess

obtainInterface / return requested FW interface

obtainInterfaceWithCallback / return requested FW interface

Active

network operator initiated endAccess / destroy all interface objects used by the client

endAccess / destroy all interface objects used by the client

Figure : State Transition Diagram for IpAccess

6.4.1.3.1 Active State

When the client requests access to the Framework on the IpInitial interface, an object implementing the IpAccess interface is created. The client can now request other Framework interfaces, including Service Discovery. When the client is no longer interested in using the interfaces it calls the endAccess method. This results in the destruction of all interface objects used by the client. In case the network operator decides that the client has no longer access to the interfaces the same will happen.

7 Framework-to-Application API

7.1 Sequence Diagrams

7.1.1 Event Notification Sequence Diagrams

7.1.1.1 Enable Event Notification
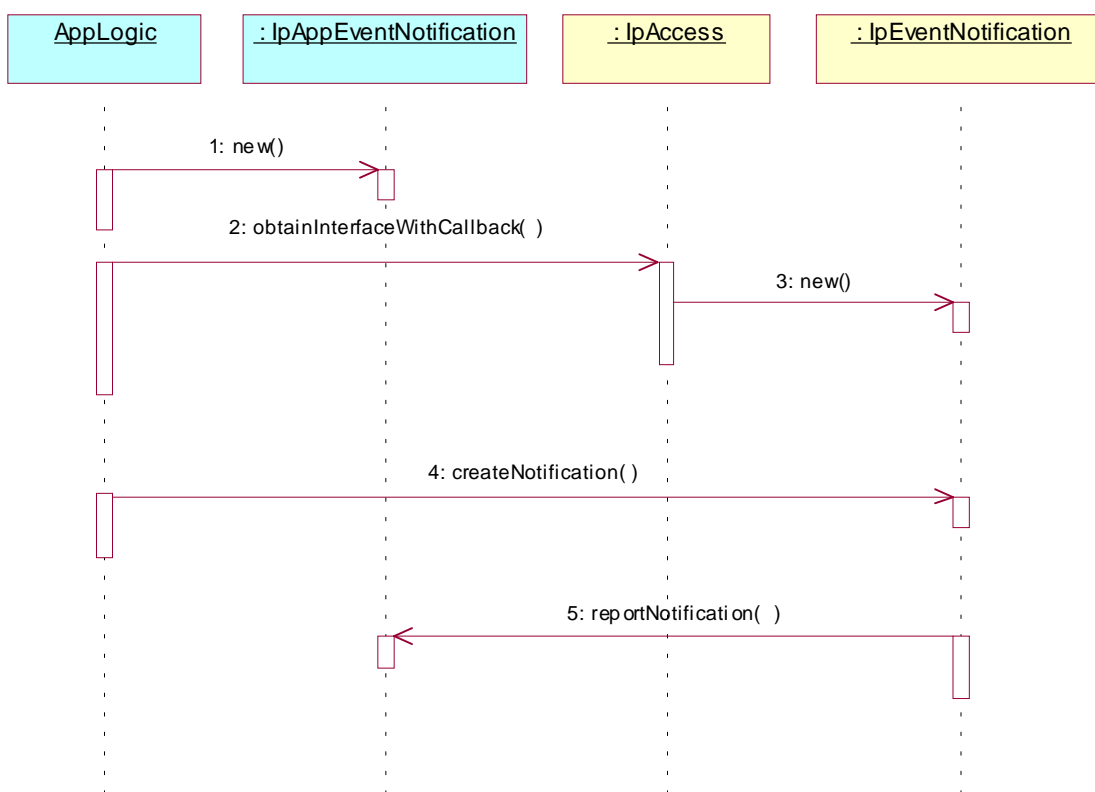
2: This message is used to receive a reference to the object implementing the IpEventNotification interface and set the callback interface for the framework.

3: If there is currently no object implementing the IpEventNotification interface, then one is created using this message.

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

4: createNotification(eventCriteria : in TpFwEventCriteria, assignmentID : out TpAssignmentIDRef) : TpResult

This message is used to enable the notification mechanism so that subsequent framework events can be sent to the application. The framework event the application requests to be informed of is the availability of new SCFs.

Newly installed SCFs become available after the invocation of registerService and announceServiceAvailability on the Framework. The application uses the input parameter eventCriteria to specify the SCFs of whose availability it wants to be notified: those specified in ServiceTypeNameList.

The result of this invocation has many similarities with the result of invoking listServiceTypes: in both cases the application is informed of the availability of a list of SCFs. The differences are:

· in the case of invoking listServiceTypes, the application has to take the initiative, but it is informed of ALL SCFs available

· in the case of using the event notification mechanism, the application needs not take the initiative to ask about the availability of SCFs, but it is only informed of the ones that are newly available.

Alternatively, or additionally, the application can request to be informed of SCFs becoming unavailable.

5: The application is notified of the availability of new SCFs of the requested type(s).

## 7.1.2   Integrity Management Sequence Diagrams

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.
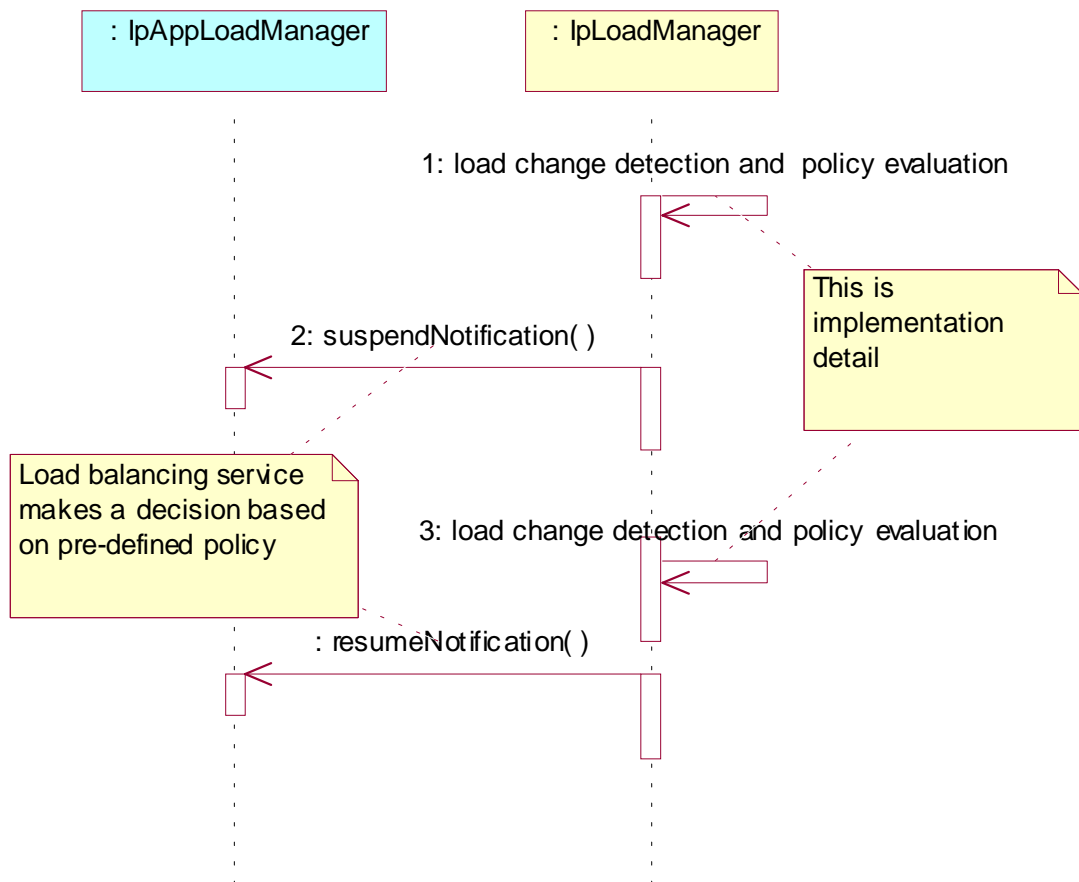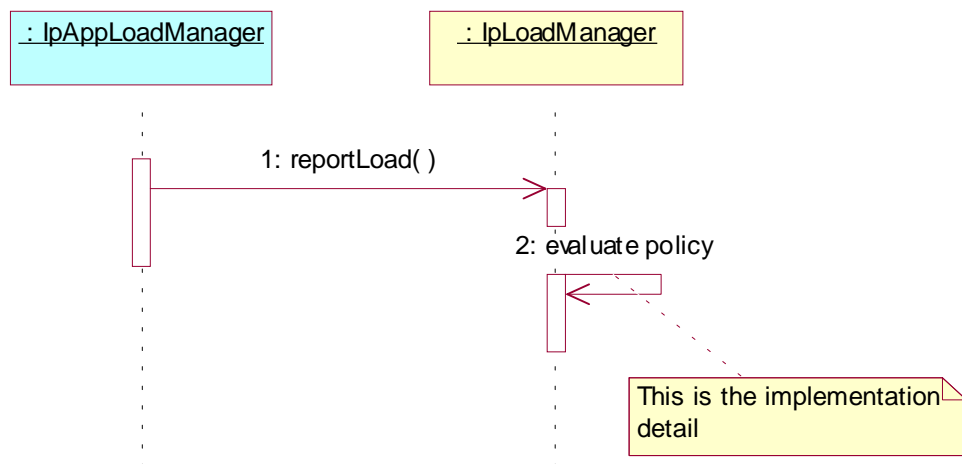
7.1.2.2 Load Management: Framework queries load statistics

This sequence diagram shows how the framework requests load statistics for an application.
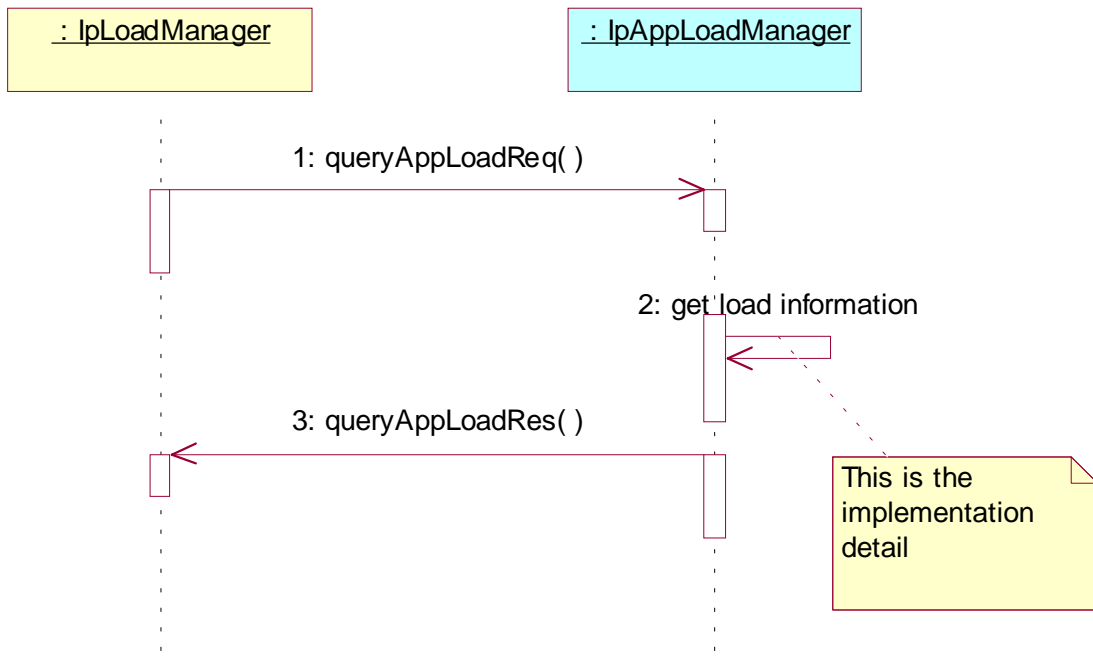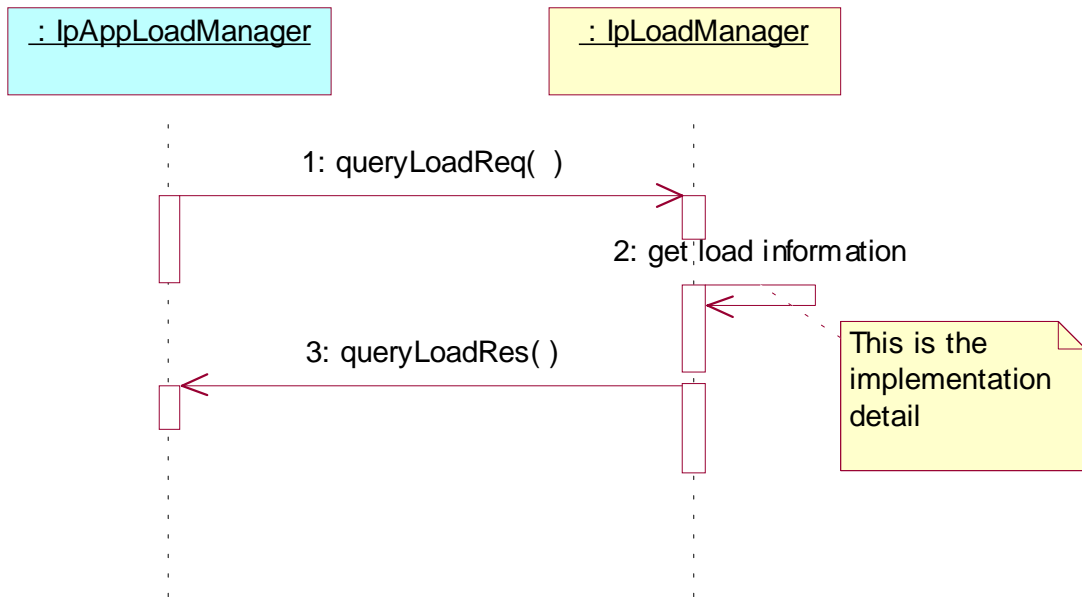

7.1.2.3 Load Management: Application reports current load condition

This sequence diagram shows how an application reports its load condition to the framework load manager.
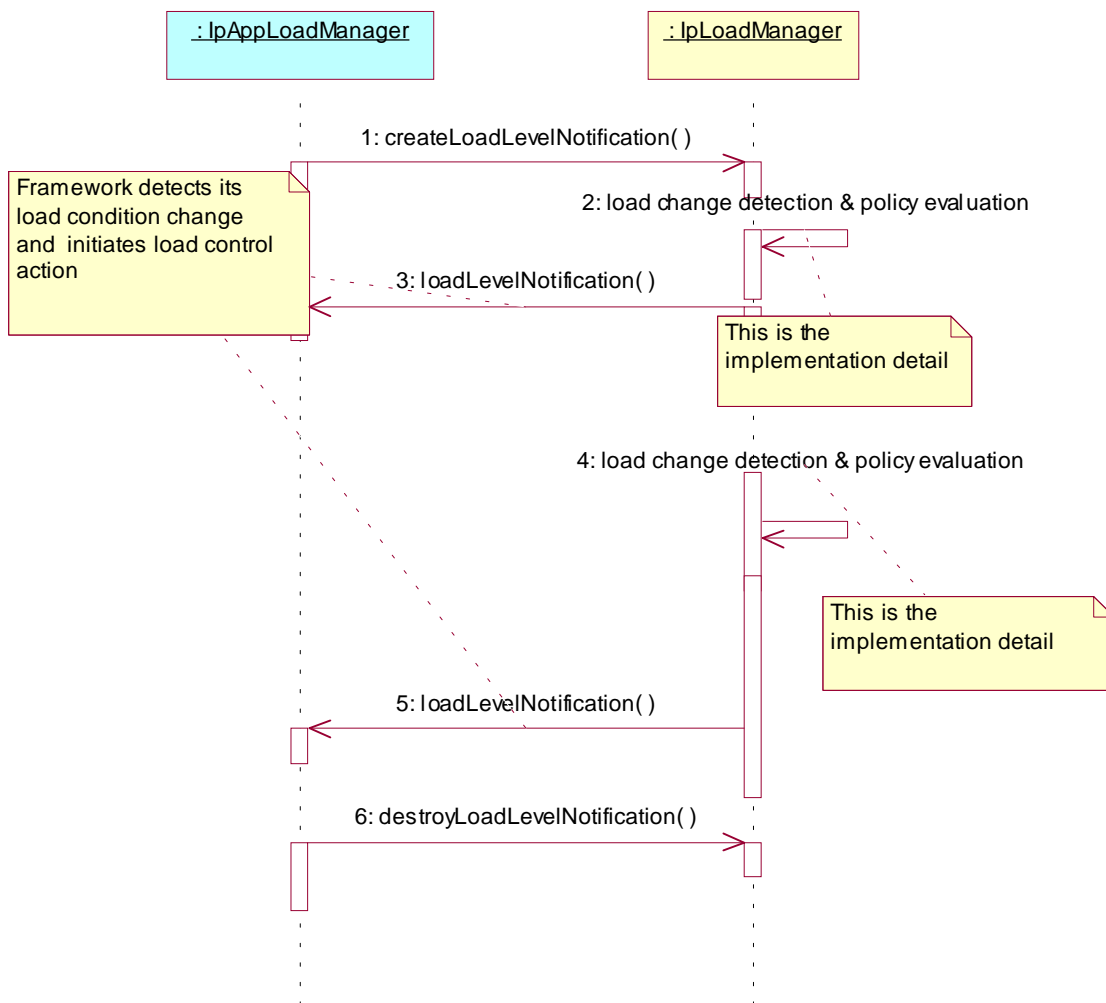
### 7.1.2.4	Load Management: Application queries load statistics

This sequence diagram shows how an application requests load statistics for the framework.


### 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( )

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

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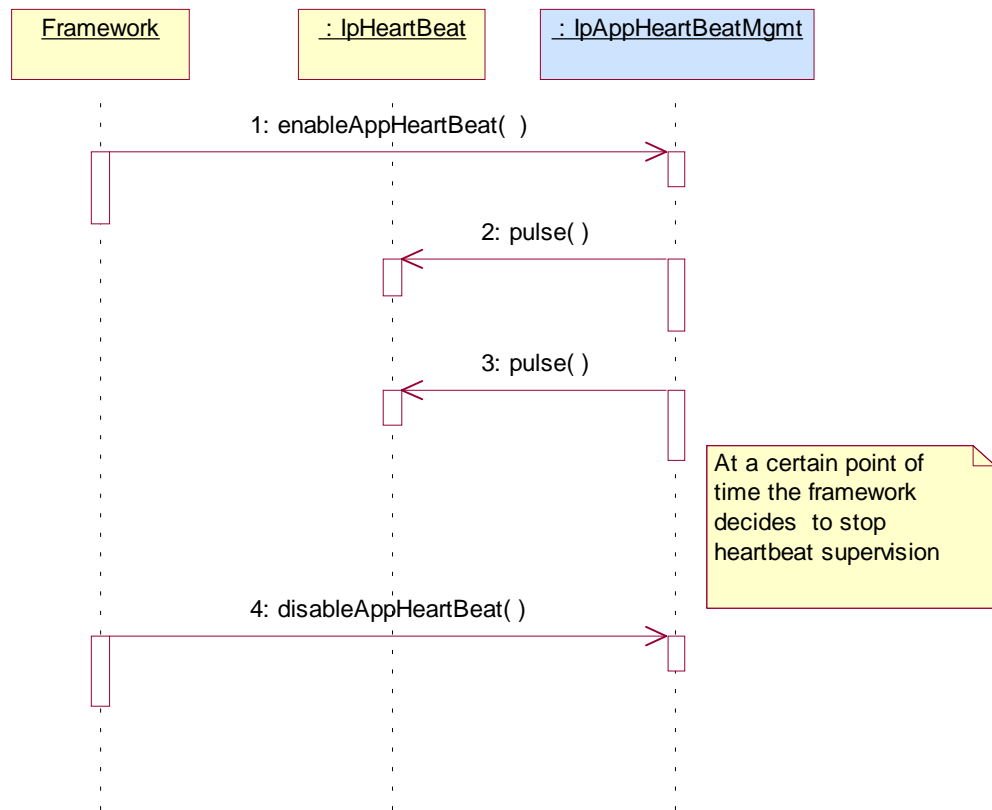
7.1.2.7   Fault Management: Framework detects a Service failure

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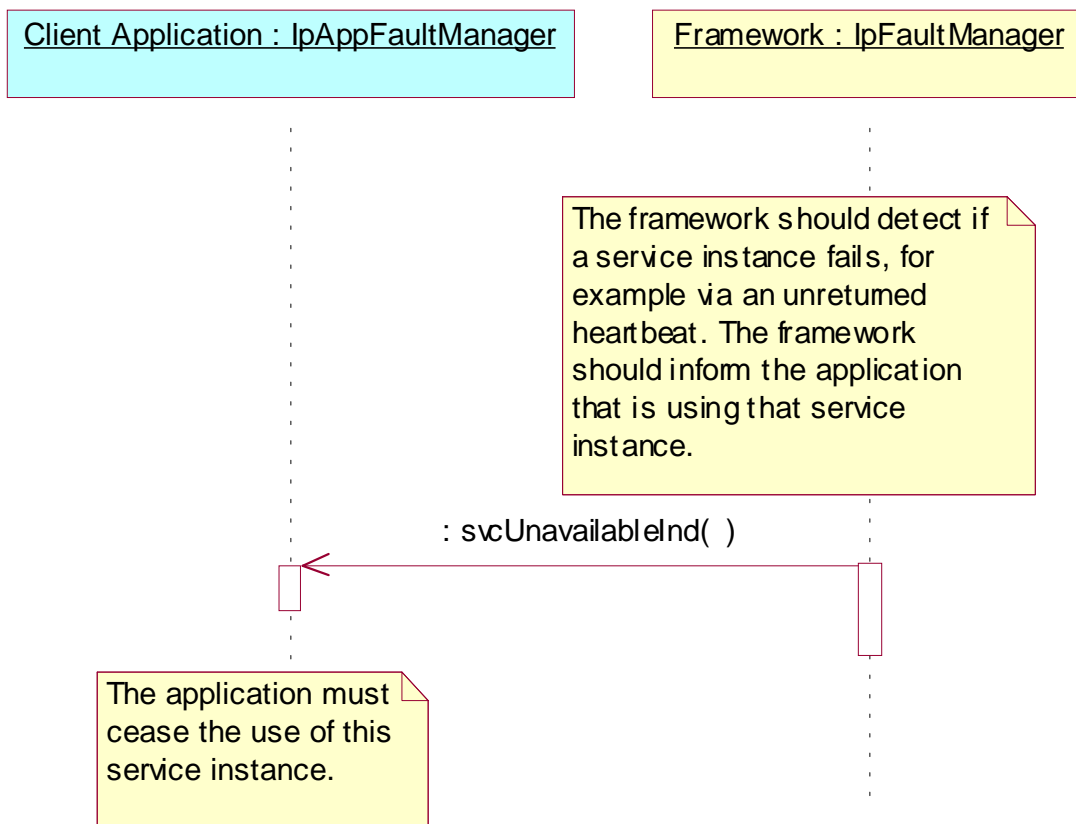
1: The framework informs the client application that is using the service instance that the service is unavailable. The client application is then expected to abandon use of this service instance and access a different service instance via the usual means (e.g. discovery, selectService etc.). The client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

7.1.2.8   Fault Management: Application requests a Framework activity test
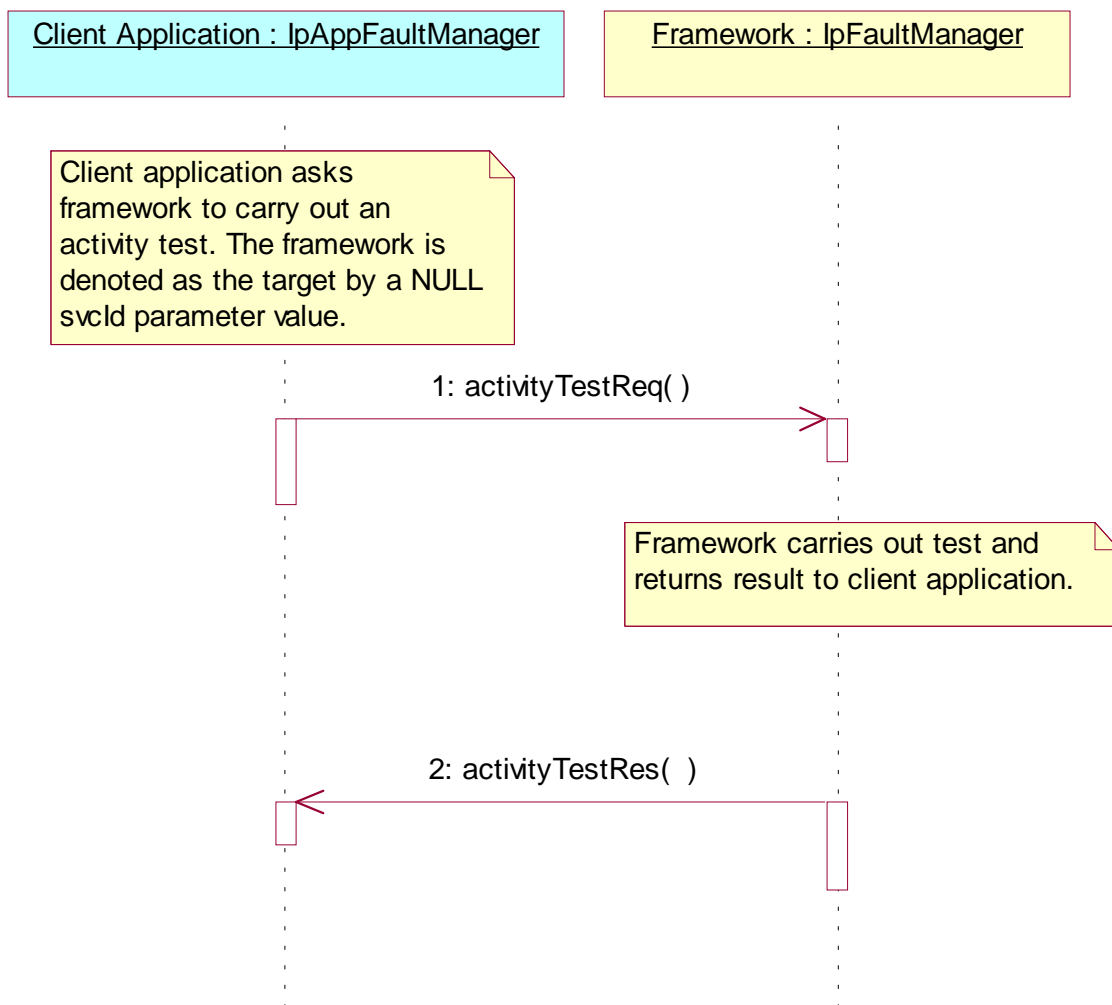
1: The client application asks the framework to do an activity test. The client identifies that it would like the activity test done for the framework, rather then a service, by supplying a NULL value for the svcId parameter.

2: The framework does the requested activity test and sends the result to the client application.

7.1.3 Service Discovery Sequence Diagrams

7.1.3.1 Service Discovery

The following figure shows how Applications discover a new Service Capability Feature in the network. Even applications that have already used the OSA API of a certain network know that the operator may upgrade it any time; this is why they use the Service Discovery interfaces.

Before the discovery process can start, the Application needs a reference to the Framework's Service Discovery interface; this is done via an invocation the method obtainInterface on the Framework's Access interface.

Discovery can be a three-step process. The first two steps have to be performed initially, but can subsequently be skipped (if the service type and its properties are already known, the application can invoke discoverService() without having to re-invoke the list/discoverServiceType methods):
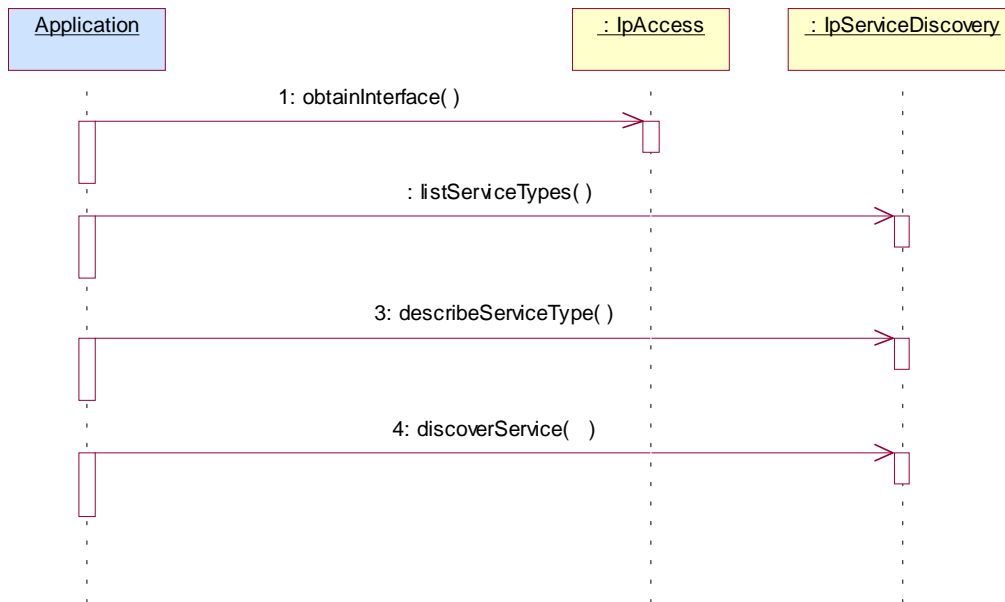
2: Discovery: first step - list service types

In this first step the application asks the Framework what service types that are available from this network. Service types are standardized or non-standardised SCF names, and thus this first step allows the Application to know what SCFs are supported by the network.

The following output is the result of this first discovery step:

·   out listTypes

This is a list of service type names, i.e., a list of strings, each of them the name of a SCF or a SCF specialization (e.g. "P_MPCC").

3: Discovery: second step - describe service type

In this second step the application requests what are the properties that describe a certain service type that it is interested in, among those listed in the first step.

The following input is necessary:

·   in name

This is a service type name: a string that contains the name of the SCF whose description the Application is interested in (e.g. "P_MPCC") .

And the output is:

·   out serviceTypeDescription

The description of the specified SCF type. The description provides information about:

·   the property names associated with the SCF,

·   the corresponding property value types,

·   the corresponding property mode (mandatory or read only) associated with each SCF property,

· the names of the super types of this type, and

· whether the type is currently enabled or disabled.

4: Discovery: third step - discover service

In this third step the application requests for a service that matches its needs by tuning the service properties (i. e., assigning values for certain properties).

The Framework then checks whether there is a match, in which case it sends the Application the serviceID that is the identifier this network operator has assigned to the SCF version described in terms of those service properties. This is the moment where the serviceID identifier is shared with the application that is interested on the corresponding service.

This is done for either one service or more (the application specifies the maximum number of responses it wishes to accept).

Input parameters are:

· in serviceTypeName

This is a string that contains the name of the SCF whose description the Application is interested in (e.g. "P_MPCC").

· in desiredPropertyList

This is again a list like the one used for service registration, but where the value of the service properties have been fine tuned by the Application to (they will be logically interpreted as "minimum", "maximum", etc. by the Framework).

The following parameter is necessary as input:

· in max

This parameter states the maximum number of SCFs that are to be returned in the "ServiceList" result.

And the output is:

· out serviceList

This is a list of duplets: (serviceID, servicePropertyList). It provides a list of SCFs matching the requirements from the Application, and about each: the identifier that has been assigned to it in this network (serviceID), and once again the service property list.

# 7.1.4 Service Agreement Management Sequence Diagrams

7.1.4.1 Service Selection

The following figure shows the process of selecting an SCF.

After discovery the Application gets a list of one or more SCF versions that match its required description. It now needs to decide which service it is going to use; it also needs to actually get a way to use it.

This is achieved by the following two steps:
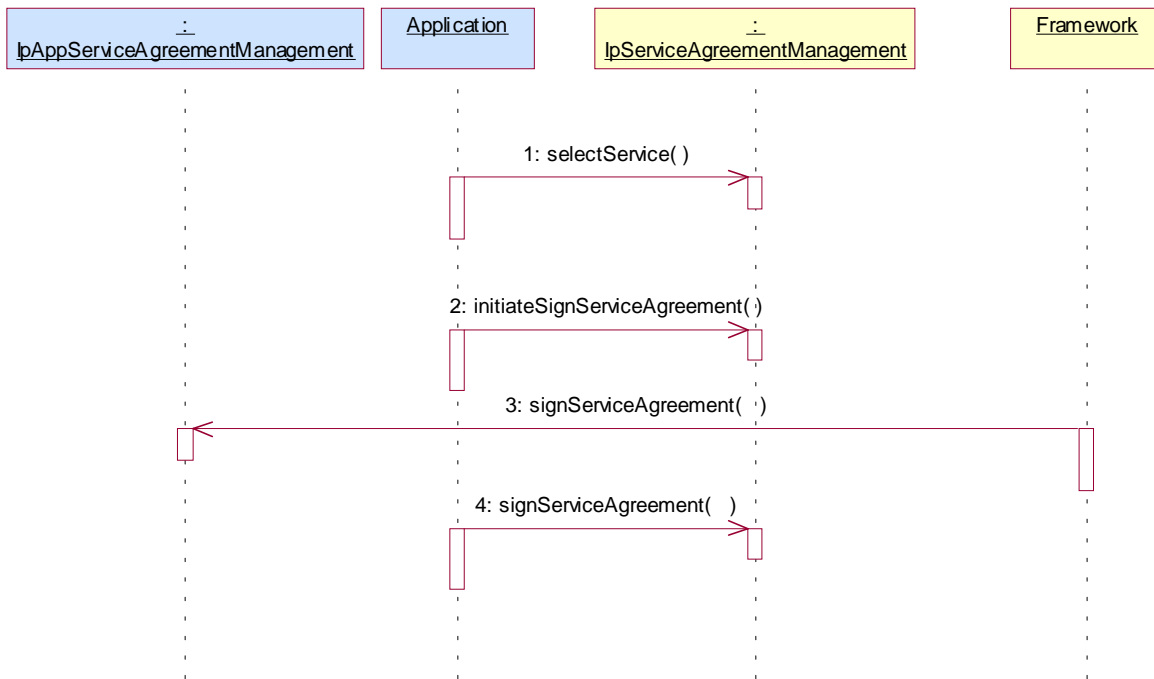
1: Service Selection: first step - selectService

In this first step the Application identifies the SCF version it has finally decided to use. This is done by means of the serviceID, which is the agreed identifier for SCF versions. The Framework acknowledges this selection by returning to the Application a new identifier for the service chosen: a service token, that is a private identifier for this service between this Application and this network, and is used for the process of signing the service agreement.

Input is:

·   in serviceID

This identifies the SCF required.

And output:

·   out serviceToken

This is a free format text token returned by the framework, which can be signed as part of a service agreement. It contains operator specific information relating to the service level agreement.

2: Service Selection: second step - signServiceAgreement

In this second step an agreement is signed that allows the Application to use the chosen SCF version. And once this contractual details have been agreed, then the Application can be given the means to actually use it. The means are a reference to the manager interface of the SCF version (remember that a manager is an entry point to any SCF). By calling the createServiceManager operation on the lifecycle manager the Framework retrieves this interface and returns it to the Application. The service properties suitable for this application are also fed to the SCF (via the lifecycle manager interface) in order for the SCS to instantiate an SCF version that is suitable for this application.

Input:

·   in serviceToken

This is the identifier that the network and Application have agreed to privately use for a certain version of SCF.

·   in agreementText

This is the agreement text that is to be signed by the Framework using the private key of the Framework.

· in signingAlgorithm

This is the algorithm used to compute the digital signature.

Output:

· out signatureAndServiceMgr

This is a reference to a structure containing the digital signature of the Framework for the service agreement, and a reference to the manager interface of the SCF.

7.2 Class Diagrams
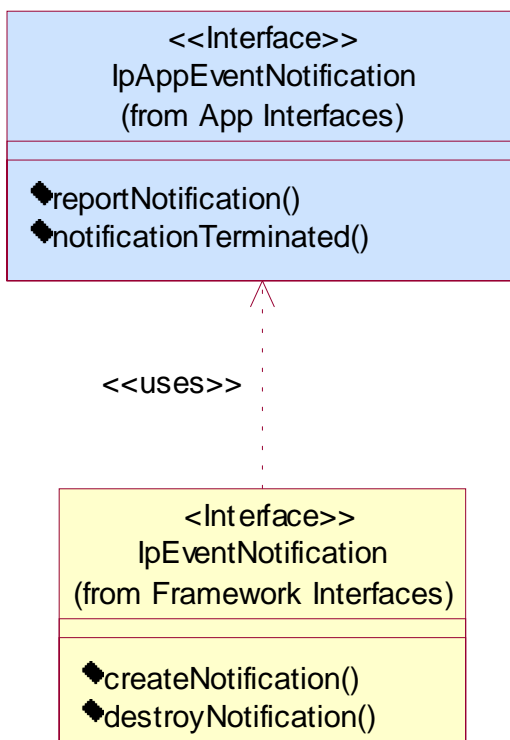
Figure: Event Notification Class Diagram
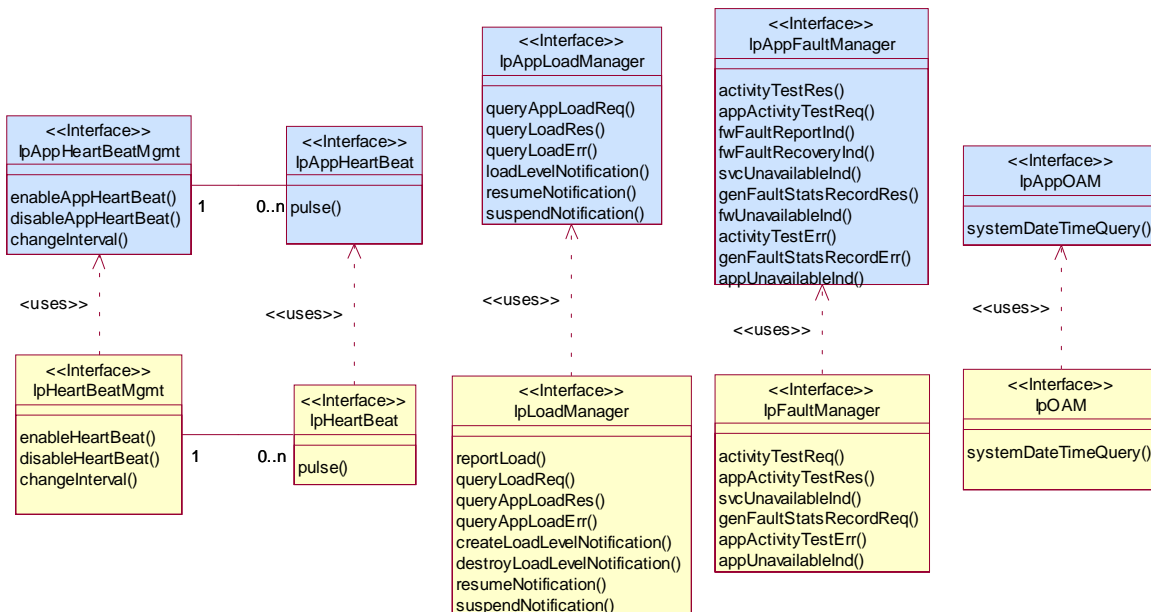
Figure: Integrity Management Package Overview
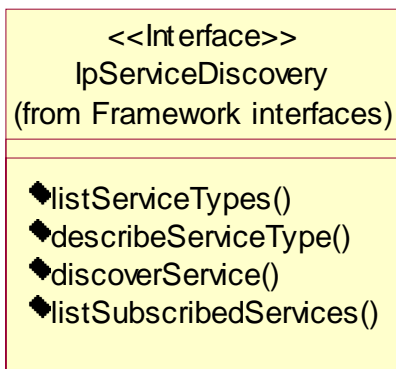
Figure: Service Discovery Package Overview
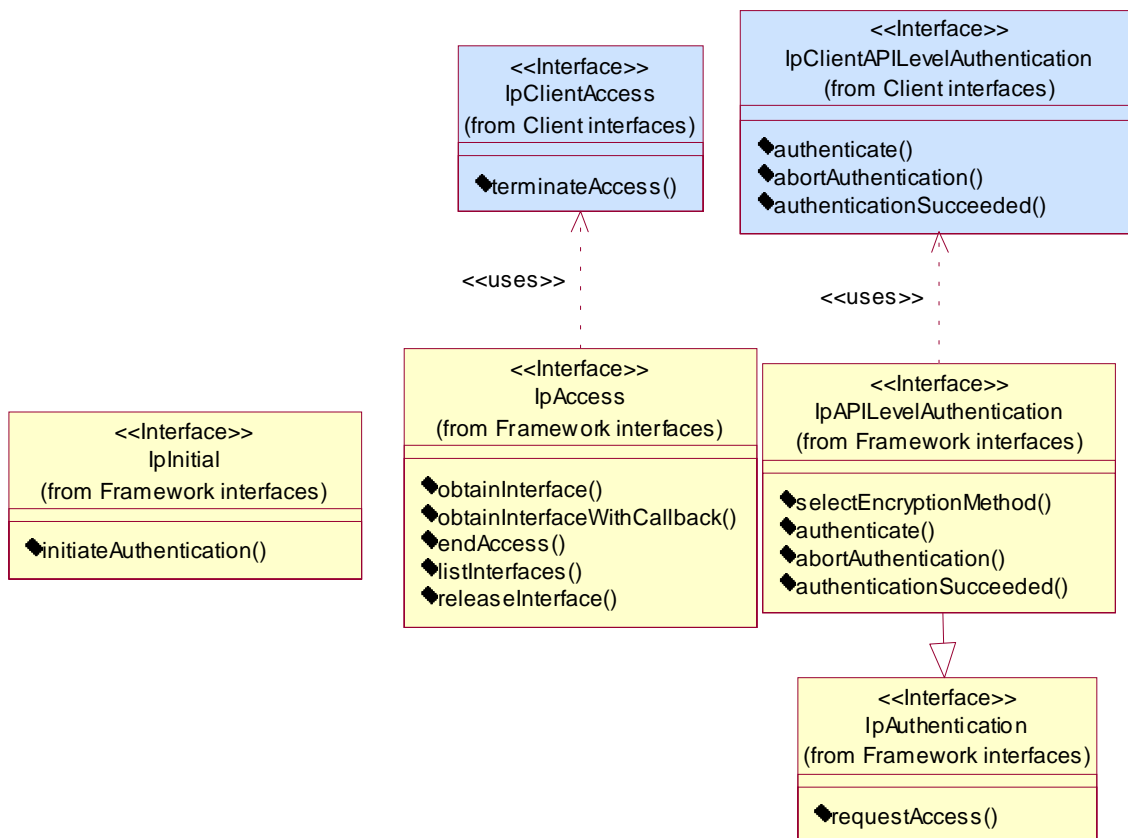
Figure: Trust and Security Management Package Overview
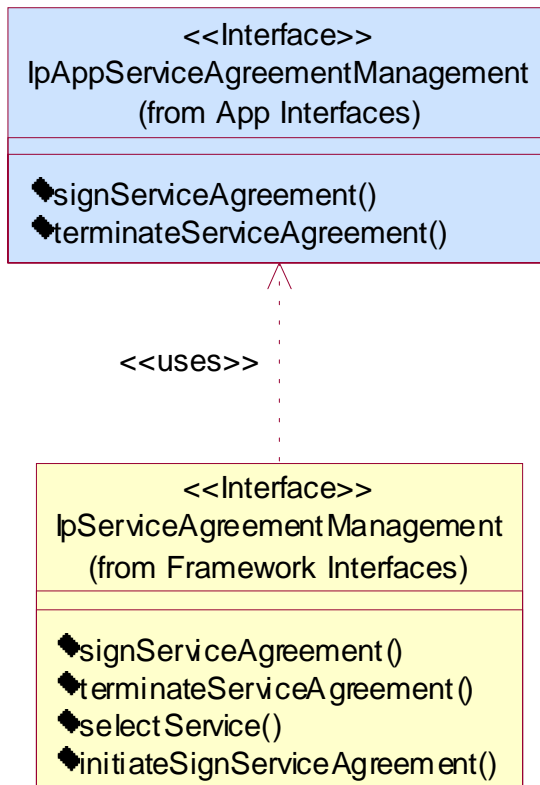
Figure: Service Agreement Management Package Overview

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.

<<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 enabled or disabled.

Parameters

name : in TpServiceTypeName
The name of the service type to be described.

· If the "name" is malformed, then the P_ILLEGAL_SERVICE_TYPE exception is raised.

· If the "name" does not exist in the repository, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

Returns

```
TpServiceTypeDescription
```

Raises

```
TpCommonExceptions,P_ACCESS_DENIED,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVI
CE_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 property {name, mode and value list} tuples 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 property {name, mode and value list} tuples 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_SERVI
CE_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 property {name, mode and value list} tuples 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:

<<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:

<<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:  IpInterfaceRef;
                    };
```

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

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

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_SER VICE_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 () : 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
No Parameters were identified for this method

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.

<<Interface>>
IpFaultManager
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

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.

Parameters

activityTestID : in TpActivityTestID

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

svcID : in TpServiceID

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method
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. A null value leaves this 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

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.

<<Interface>>
IpAppHeartBeat
pulse () : void
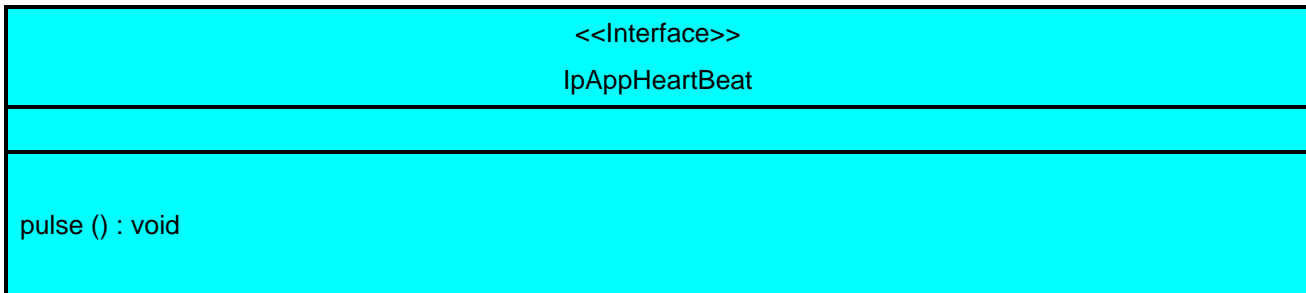
Method
pulse()

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.
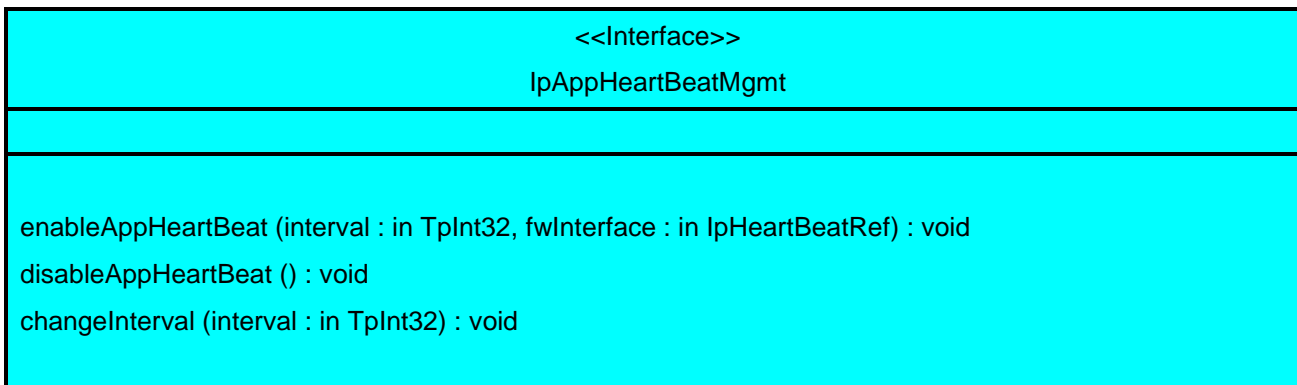
Parameters
No Parameters were identified for this method

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.

<<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.

<<Interface>>
IpHeartBeat
pulse () : void

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.

<<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 of the OSA APIs.

<<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
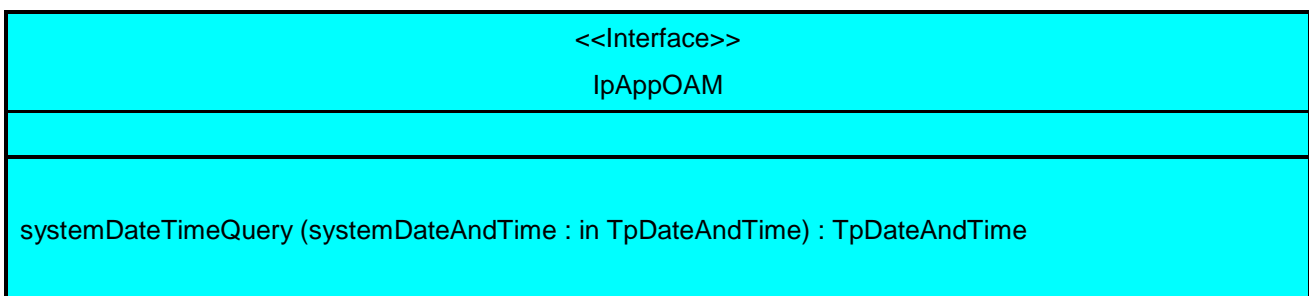
Method
systemDateTimeQuery()

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.

<<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

7.4 State Transition Diagrams

This section contains the State Transition Diagrams for the objects that implement the Framework interfaces on the gateway side. The State Transition Diagrams show the behaviour of these objects. For each state the methods that can be invoked by the application are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the application also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

7.4.1 Service Discovery State Transition Diagrams

7.4.1.1 State Transition Diagrams for IpServiceDiscovery

obtainFrameworkInterface(discoveryService)

obtainInterfaceWithCallback(discoveryService)

istServiceTypes

describeServiceType

listSubscribedServices

discoverService

Active

IpAccess.endAccess
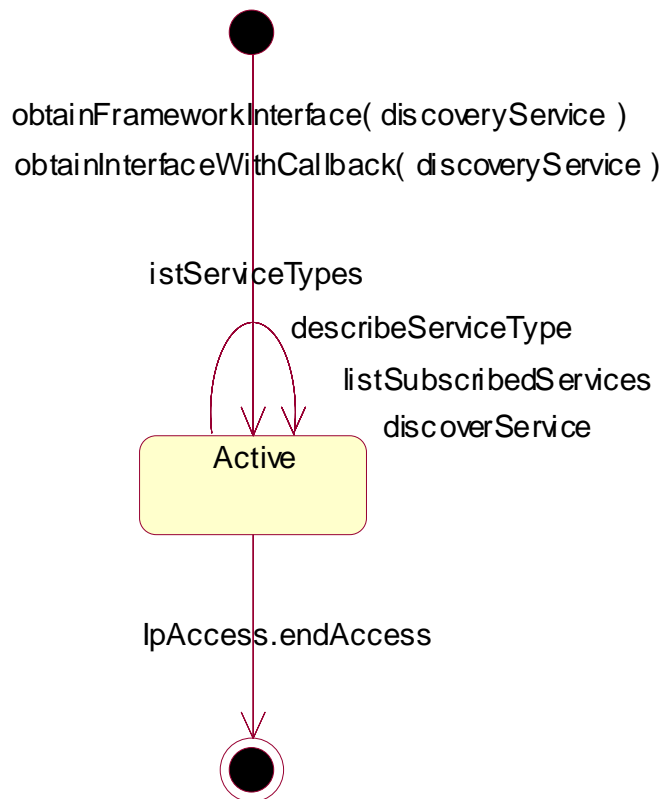
Figure : State Transition Diagram for IpServiceDiscovery

7.4.1.1.1 Active State

When the application requests Service Discovery by invoking the obtainInterface or the obtainInterfaceWithCallback methods on the IpAccess interface, an instance of the IpServiceDiscovery will be created. Next the application is allowed to request a list of the provided SCFs and to obtain a reference to interfaces of SCFs.

7.4.2   Service Agreement Management State Transition Diagrams

There are no State Transition Diagrams defined for Service Agreement Management

7.4.3   Integrity Management State Transition Diagrams

7.4.3.1   State Transition Diagrams for IpLoadManager

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
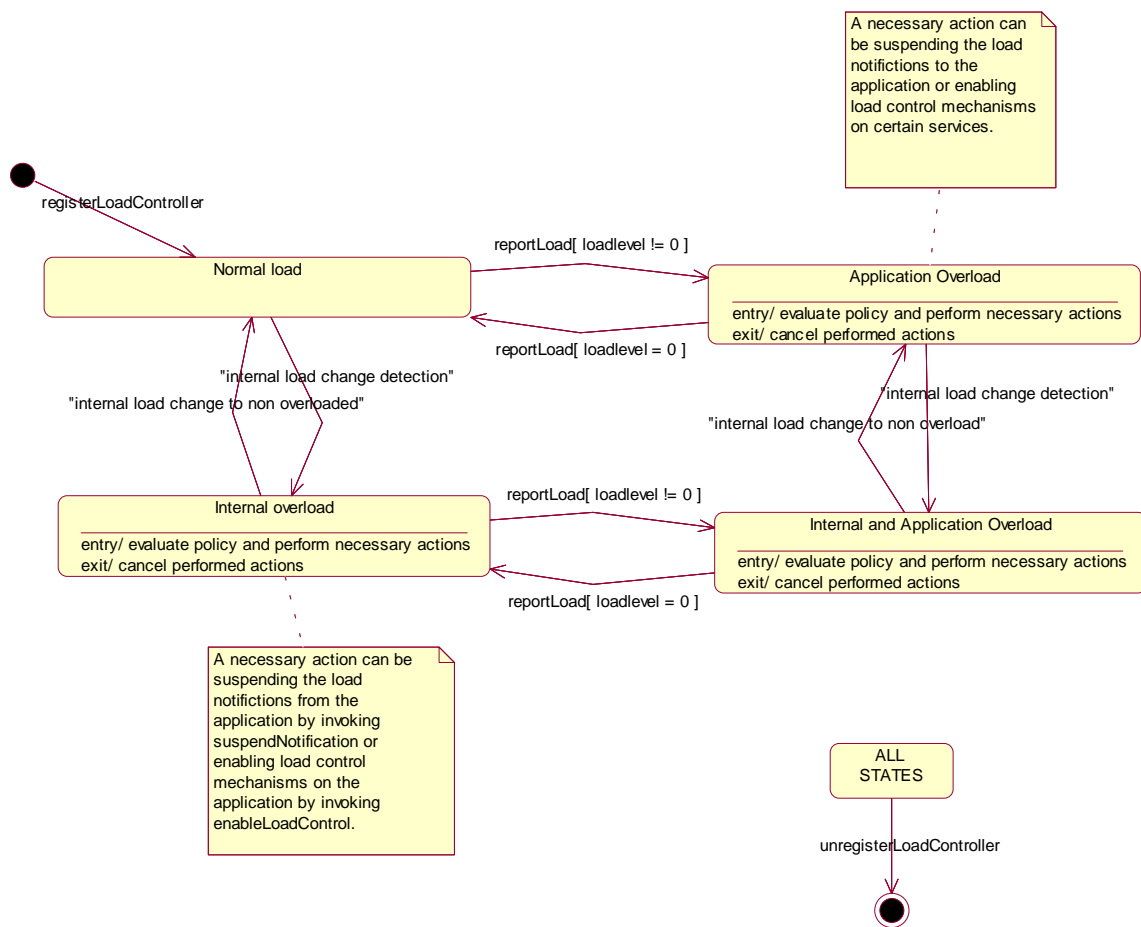
Figure : State Transition Diagram for LoadManagerInternal

7.4.3.2.1 Normal load State

In this state the none of the entities defined in the load balancing policy between the application and the framework / SCFs is overloaded.

7.4.3.2.2 Application Overload State

In this state the application has indicated it is overloaded. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

7.4.3.2.3 Internal overload State

In this state the Framework or one or more of the SCFs within the specific load policy is overloaded. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

7.4.3.2.4 Internal and Application Overload State

In this state the application is overloaded as well as the Framework or one or more of the SCFs within the specific load policy. When entering this state the load policy is consulted and the appropriate actions are taken by the LoadManager.

7.4.3.3 State Transition Diagrams for IpOAM

IpAccess.obtainInterface

IpAccess.obtainInterfaceWithCallback

systemDateTimeQuery

Active

IpAccess.endAccess

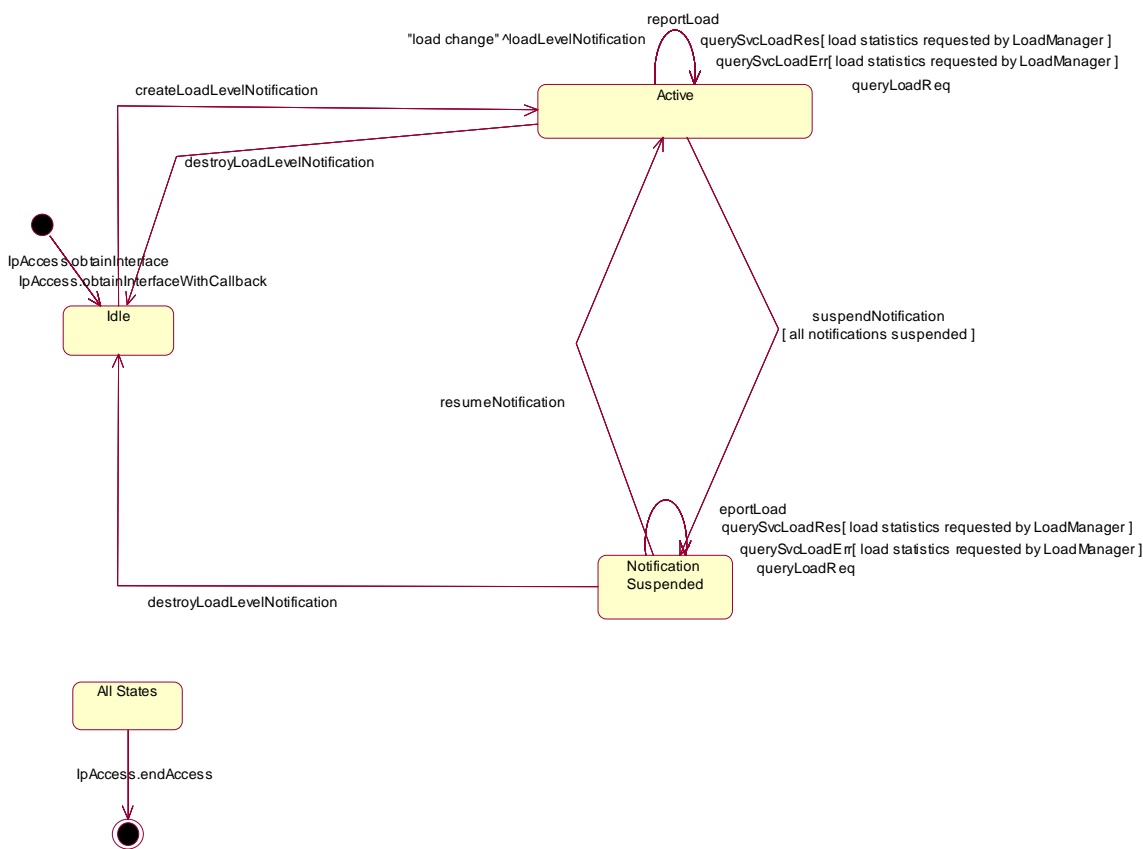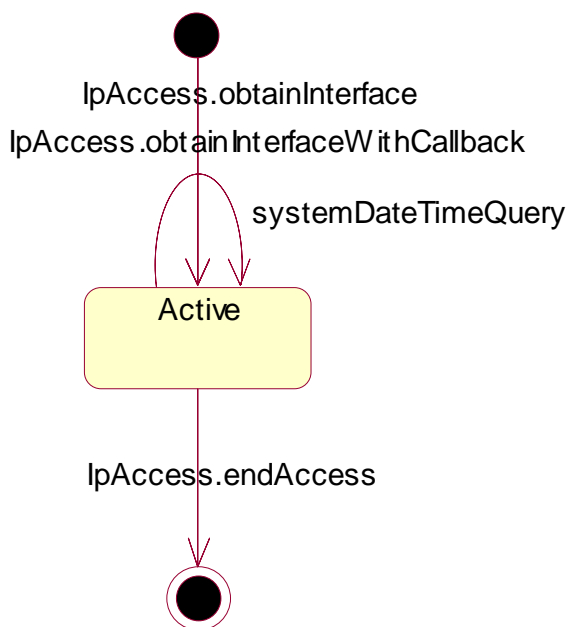**Figure : State Transition Diagram for IpOAM**

7.4.3.3.1 Active State

In this state the application has obtained a reference to the IpOAM interface. The application is now able to request the date / time of the Framework.

7.4.3.4 State Transition Diagrams for IpFaultManager

Figure : State Transition Diagram for IpFaultManager

7.4.3.4.1 Framework Active State

This is the normal state of the framework, which is fully functional and able to handle requests from both applications and services capability features.

7.4.3.4.2 Framework Faulty State

In this state, the framework has detected an internal problem with itself such that application and services capability features cannot communicate with it anymore; attempts to invoke any methods that belong to any SCFs of the framework return an error. If the framework ever recovers, applications with fault management callbacks will be notified via a fwFaultRecoveryInd message.

7.4.3.4.3 Framework Activity Test State

In this state, the framework is performing self-diagnostic test. If a problem is diagnosed, all applications with fault management callbacks are notified through a fwFaultReportInd message.

7.4.3.4.4 Service Activity Test State

In this state, the framework is performing a test on one service capability feature. If the SCF is faulty, applications with fault management callbacks are notified accordingly through a svcUnavailableInd message.

7.4.4 Event Notification State Transition Diagrams

7.4.4.1 State Transition Diagrams for IpEventNotification

Figure : State Transition Diagram for IpEventNotification

7.4.4.1.1 Idle State

7.4.4.1.2 Notification Active State

8 Framework-to-Service API

8.1 Sequence Diagrams

8.1.1 Service Discovery Sequence Diagrams

No Sequence Diagrams exist for Service Discovery

8.1.2 Service Registration Sequence Diagrams

8.1.2.1 New SCF Registration

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

```
┌─────────────────┐                              ┌─────────────────────┐
│      SCS        │                              │         :           │
│                 │                              │ IpFwServiceRegistration │
└─────────────────┘                              └─────────────────────┘
         ┊                                                  ┊
         ┊            1: registerService( )                 ┊
        ┌─┐──────────────────────────────────────────────▶┌─┐
        │ │                                                │ │
        └─┘                                                └─┘
         ┊                                                  ┊
         ┊         2: announceServiceAvailability( )        ┊
        ┌─┐──────────────────────────────────────────────▶┌─┐
        │ │                                                │ │
        │ │                                                └─┘
        └─┘                                                  ┊
         ┊                                                  ┊
```

1: Registration: first step - register service

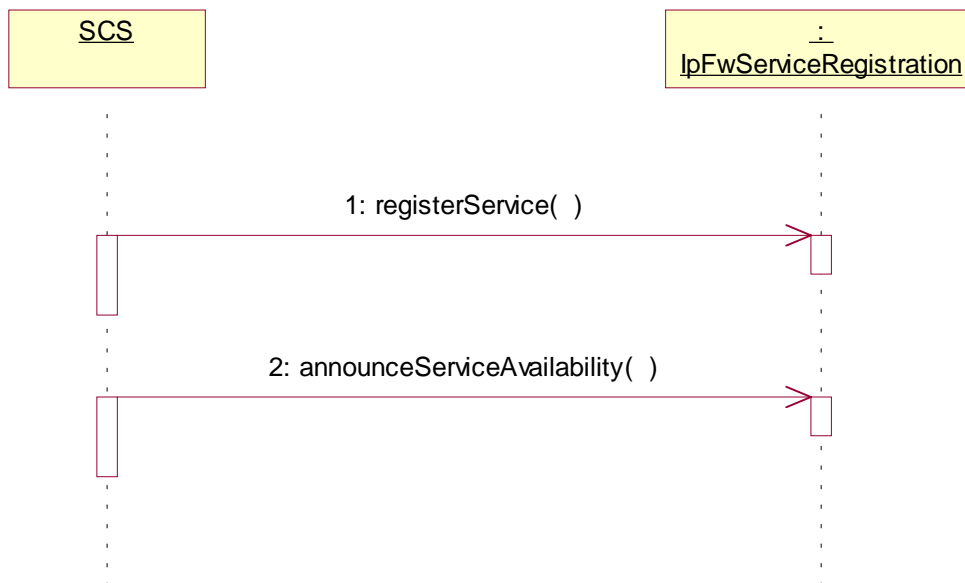The purpose of this first step in the process of registration is to agree, within the network, on a name to call, internally, a newly installed SCF version. It is necessary because the OSA Framework and SCF in the same network may come from different vendors. The goal is to make an association between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

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

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

·   in serviceTypeName

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

·   in servicePropertyList

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

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

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

The following output parameter results from service registration:

·   out serviceID

This is a string, automatically generated by the Framework and unique within the Framework.

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

2: Registration: second step - announce service availability

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

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

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

·   in serviceID

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

·   in serviceInstanceLifecycleManagerRef

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

8.1.3   Service Instance Lifecycle Manager Sequence Diagrams

8.1.3.1   Sign Service Agreement

This sequence illustrates how the application can get access to a specified service. It only illustrates the last part: the signing of the service agreement and the corresponding actions towards the service. For more information on accessing the framework, authentication and discovery of services, see the corresponding sections.
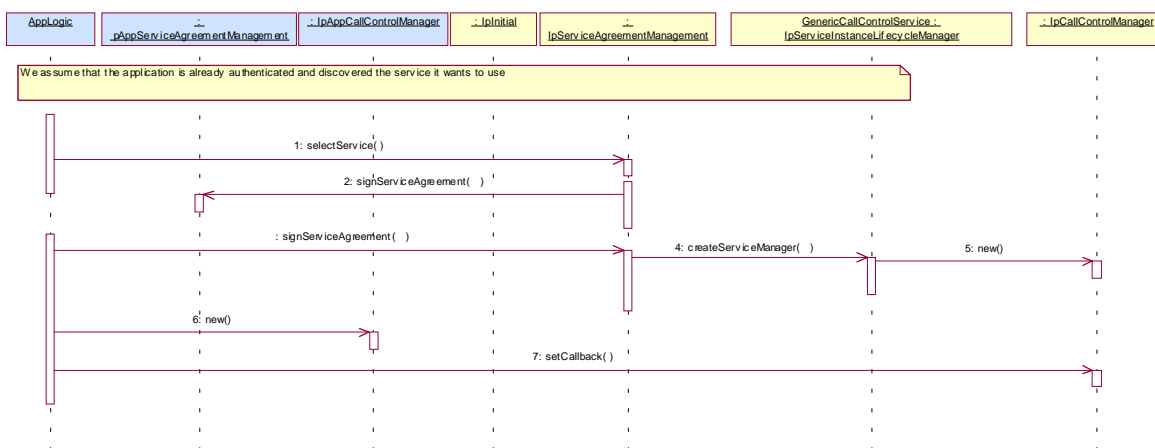
1: The application selects the service, using a serviceID for the generic call control service. The serviceID could have been obtained via the discovery interface. A ServiceToken is returned to the application.

2: The framework signs the service agreement.

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

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

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

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

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

8.1.4 Integrity Management Sequence Diagrams

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
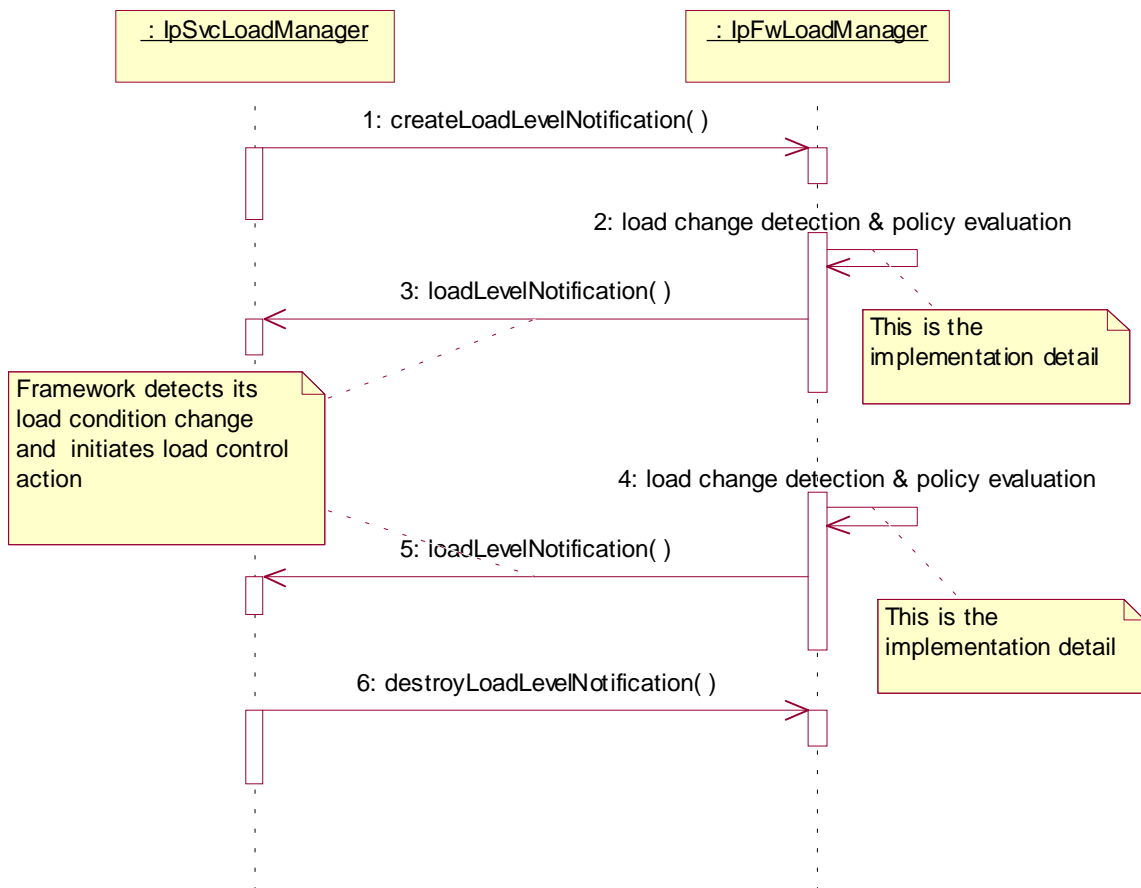
8.1.4.2 Load Management: Client and Service Load Balancing
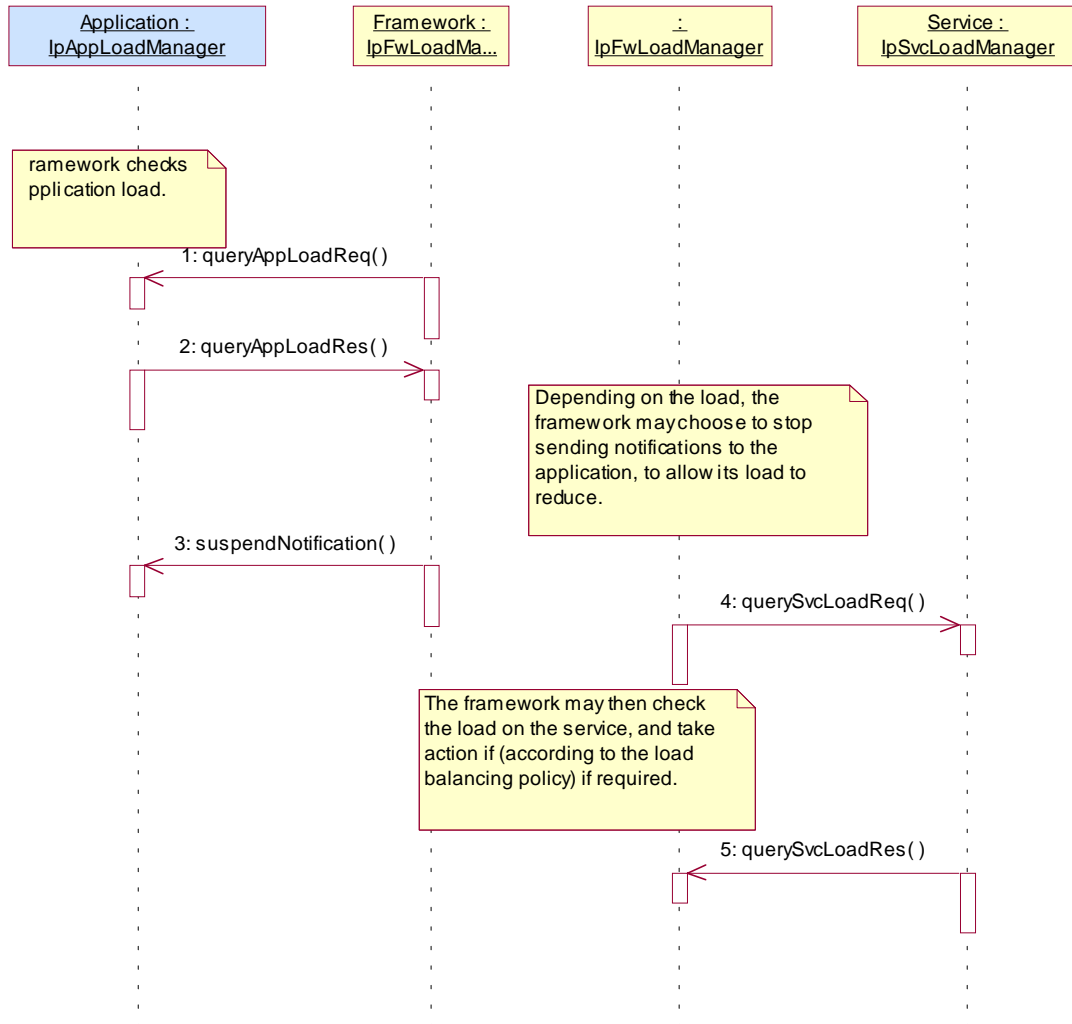
8.1.4.3 Heartbeat Management: Start/perform/end heartbeat supervision of the service

In this sequence diagram, the framework has decided that it wishes to monitor the service, and has therefore requested the service to commence sending its heartbeat. The service responds by sending its heartbeat at the specified interval. The framework then decides that it is satisfied with the service's health and disables the heartbeat mechanism. If the heartbeat was not received from the service within the specified interval, the framework can decide that the service has failed the heartbeat and can then perform some recovery action.
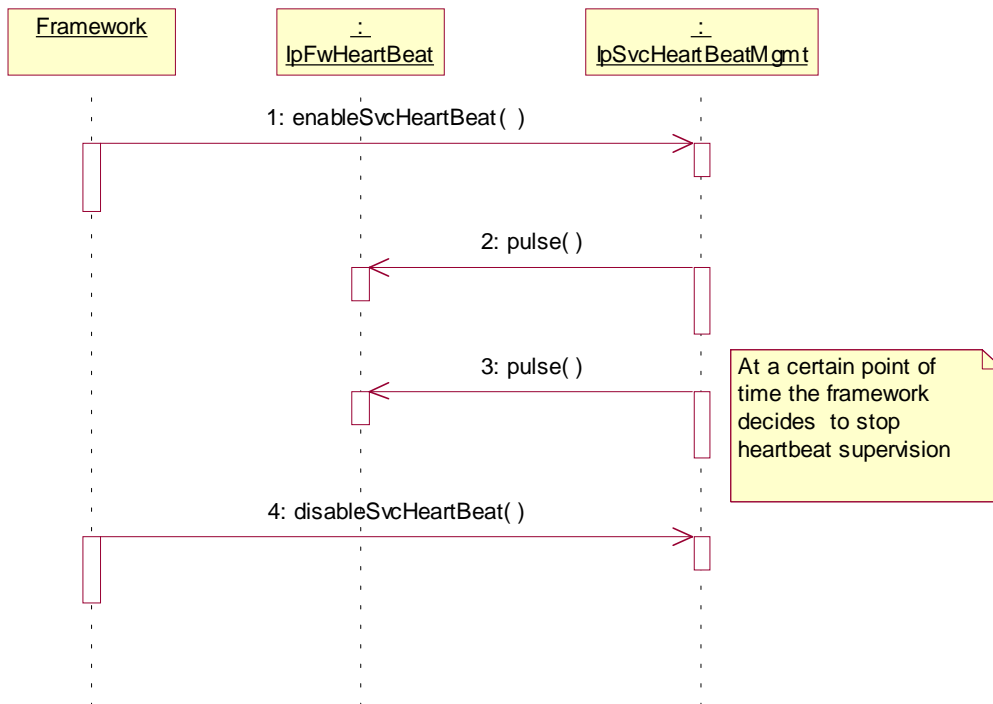
8.1.4.4　Fault Management: Service requests Framework activity test
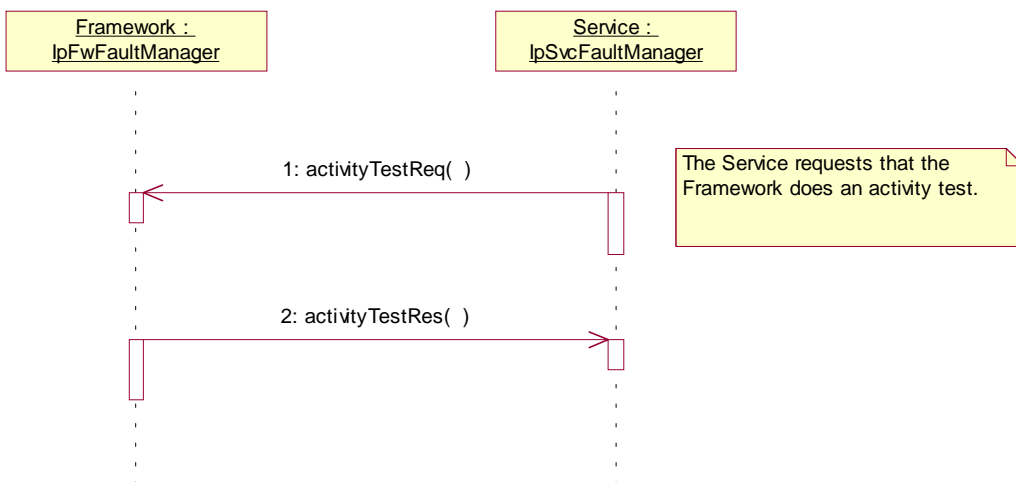
1:　The service asks the framework to carry out its activity test. The service denotes that it requires the activity test done for the framework, rather than an application, by supplying an appropriate parameter.

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

8.1.4.5 Fault Management: Service requests Application activity test
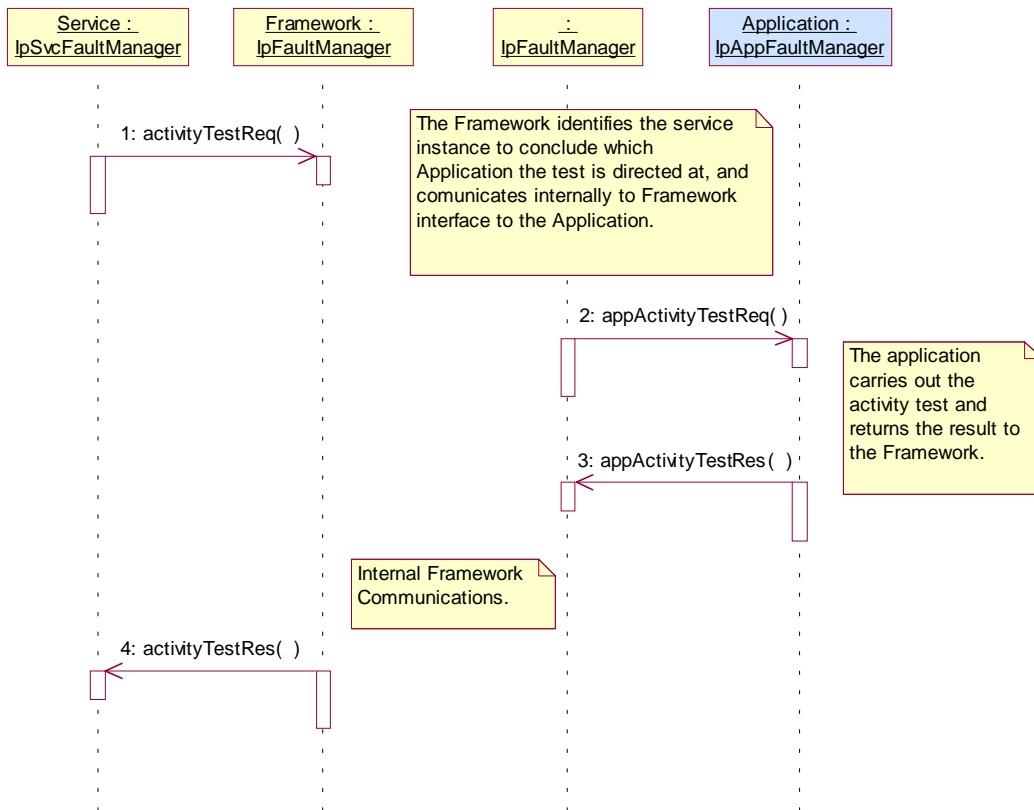
1: The service asks the framework to invoke an activity test on a client application, the application is identified by the appId parameter.

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

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

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

8.1.4.6 Fault Management: Application requests Service activity test
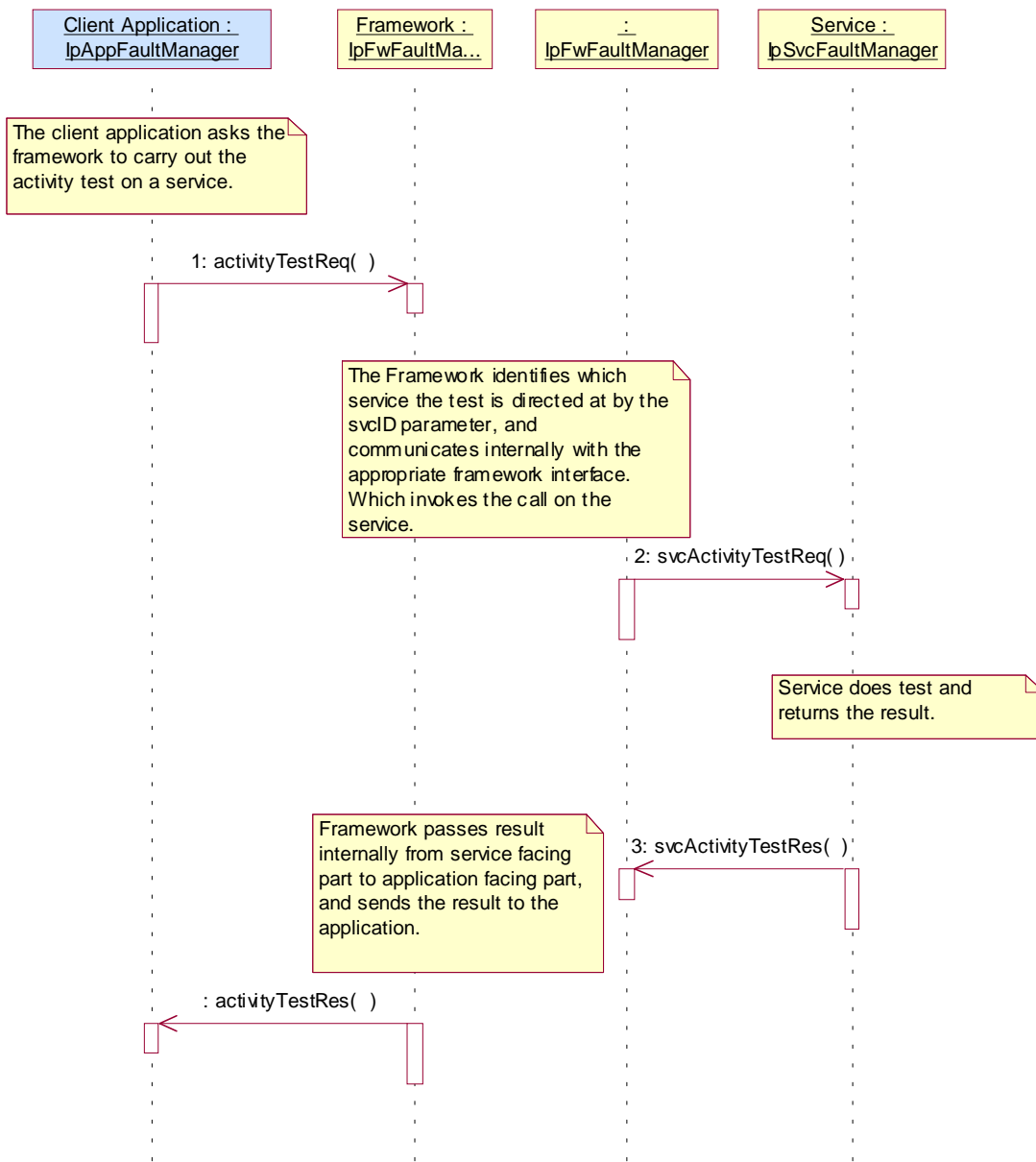
1: The client application asks the framework to invoke an activity test on a service, the service is identified by the svcId parameter.

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

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

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

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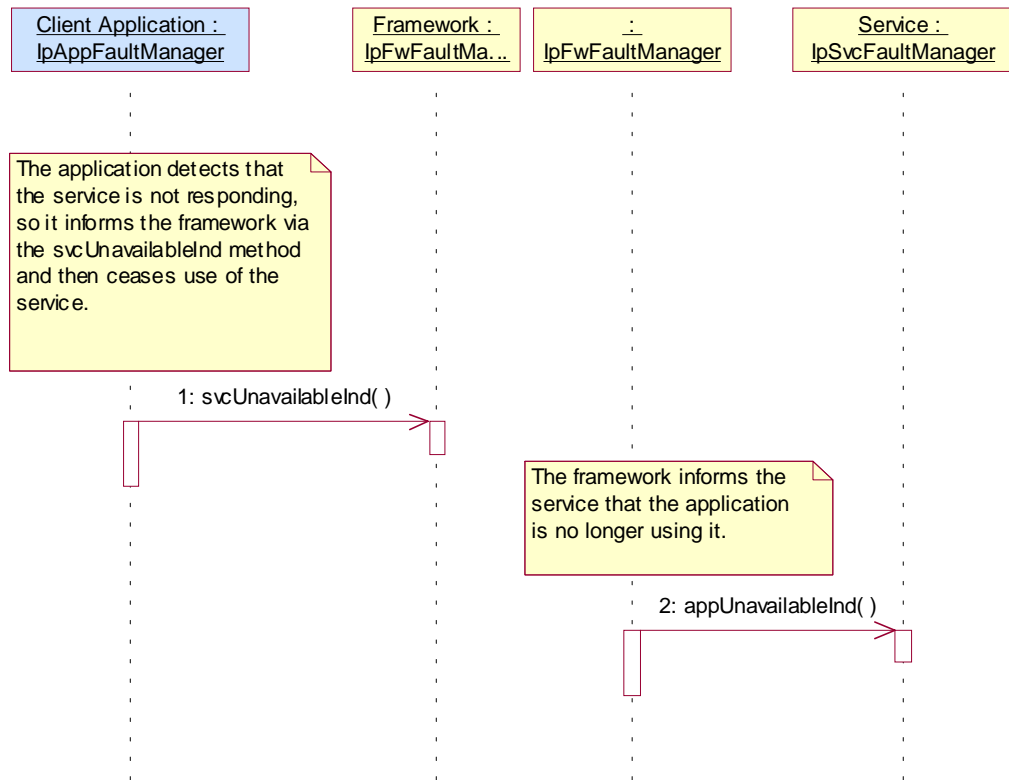
1: The client application detects that the service instance is currently not available, i.e. the service instance is not responding to the client application in the normal way, so it informs the framework and takes action to stop using this service instance and change to a different one (via the usual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

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

8.1.5　Event Notification Sequence Diagrams

No Sequence Diagrams exist for Event Notification

8.2　Class Diagrams

```
          <Interface>>
        IpFwServiceDiscovery
      (from Framework interfaces)

       ◆listServiceTypes()
       ◆describeServiceType()
       ◆discoverService()
       ◆listRegisteredServices()
```

Figure: Service Discovery Package Overview

```
          <<Interface>>
        IpFwServiceRegistration
      (from Framework interfaces)

       ◆registerService()
       ◆announceServiceAvailability()
       ◆unregisterService()
       ◆describeService()
       ◆unannounceService()
```
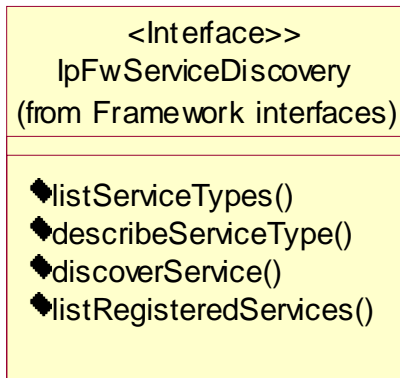
Figure: Service Registration Package Overview

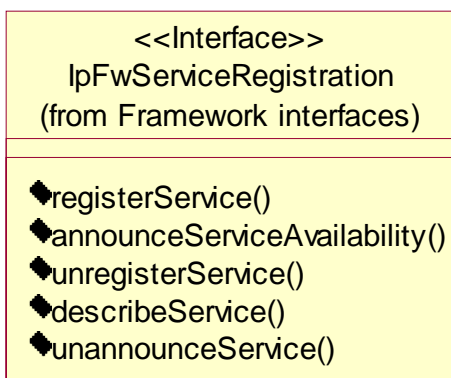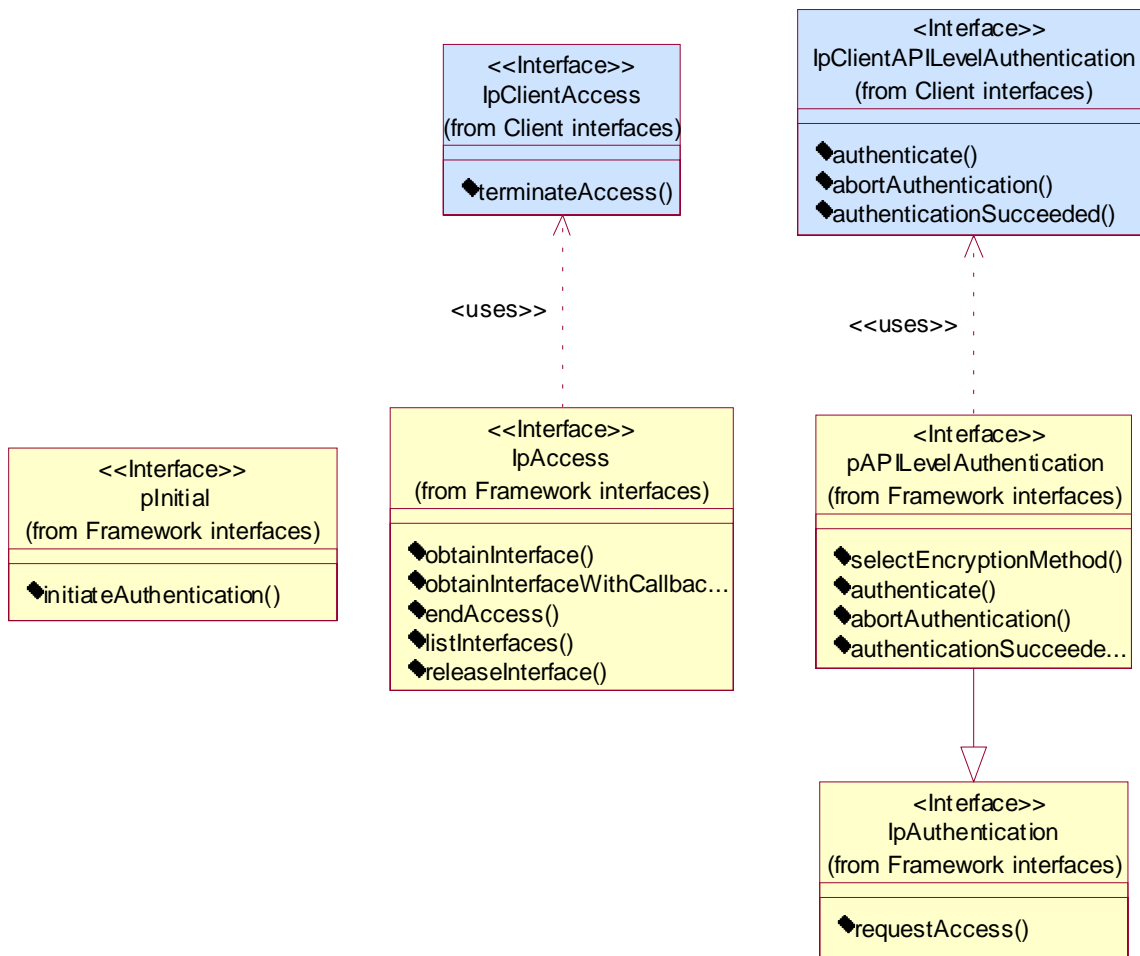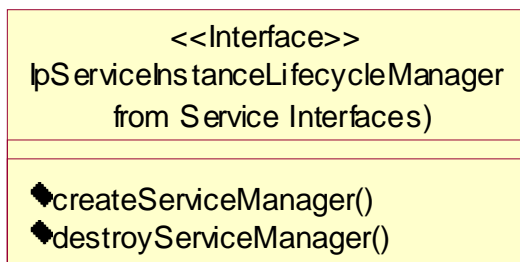**Figure: Trust and Security Management Package Overview**

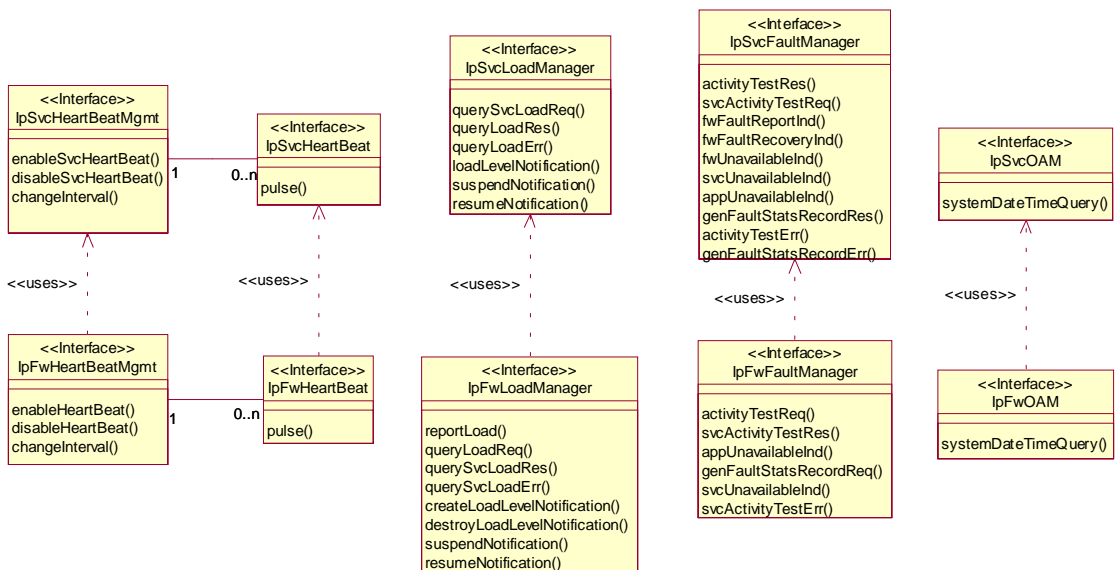**Figure: Service Instance Lifecycle Manager Package Overview**

Figure: Integrity Management Package Overview
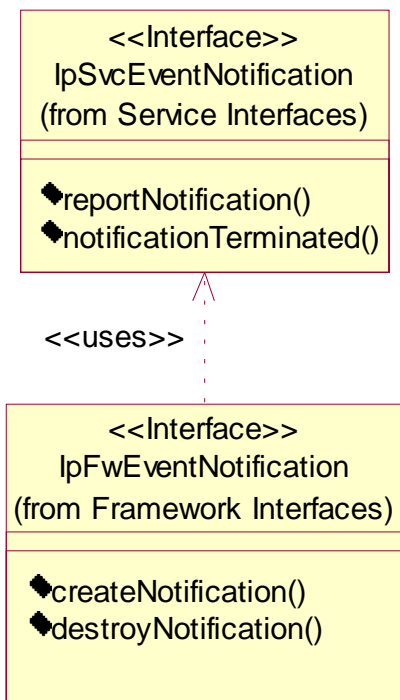
Figure: Event Notification Package Overview

# 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 section enables the service supplier to obtain a list of all the service types supported by the framework and their associated sets of service property values.

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

### 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.

<<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 behavioral, non-functional and non-computational aspects of the service. Service properties are marked "mandatory" or "readonly". These property mode attributes have the following semantics:

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

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

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

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

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, serviceManager: out IpServiceRefRef). When the service agreement is signed for some serviceID (using signServiceAgreement()), the framework calls the createServiceManager() for this service, gets a serviceManager and returns this to the client application.

Parameters

serviceID : in TpServiceID

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

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_IN TERFACE_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.

```
                          <<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 section.

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

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

8.3.3.1   Interface Class IpFwServiceDiscovery

Inherits from: IpInterface.

<<Interface>>
IpFwServiceDiscovery
listServiceTypes () : TpServiceTypeNameList   describeServiceType (name : in TpServiceTypeName) : TpServiceTypeDescription   discoverService (serviceTypeName : in TpServiceTypeName, desiredPropertyList : in TpServicePropertyList, max : in TpInt32) : TpServiceList   listRegisteredServices () : TpServiceList

Method
listServiceTypes()

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

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

Parameters
No Parameters were identified for this method

Returns

TpServiceTypeNameList

Raises

TpCommonExceptions

Method
describeServiceType()

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

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

Parameters

name : in TpServiceTypeName

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

Returns

TpServiceTypeDescription

Raises

TpCommonExceptions,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE

Method
discoverService()

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

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

Parameters

serviceTypeName : in TpServiceTypeName

The name of the required service type. If the string representation of the "type" does not obey the rules for service type identifiers, then the P_ILLEGAL_SERVICE_TYPE exception is raised. If the "type" is correct syntactically but is not

recognised as a service type within the Framework, then the P_UNKNOWN_SERVICE_TYPE exception is raised. The framework may return a service of a subtype of the "type" requested. A service sub-type can be described by the properties of its supertypes.

desiredPropertyList : in TpServicePropertyList

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

max : in TpInt32

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

Returns

TpServiceList

Raises

TpCommonExceptions,P_ILLEGAL_SERVICE_TYPE,P_UNKNOWN_SERVICE_TYPE,P_INVALI D_PROPERTY

Method
listRegisteredServices()

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

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

Parameters
No Parameters were identified for this method

Returns

TpServiceList

Raises

TpCommonExceptions

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.

<<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

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. A null value leaves this 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

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

```
┌─────────────────────────────────────────────────────────────────────────────────┐
│                               <<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                                                                           │
│                                                                                    │
└─────────────────────────────────────────────────────────────────────────────────┘
```

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 : <u>in TpInterfaceFault</u>

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

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.

<<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.

<<Interface>>
IpFwHeartBeat
pulse () : void

Method
pulse()

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

Parameters
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.

```
                         <<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.

<<Interface>>
IpSvcHeartBeat
pulse () : void
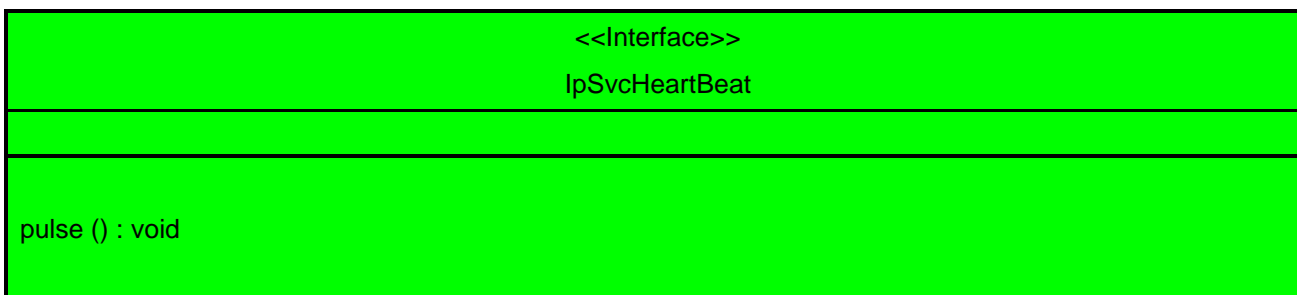
Method
pulse()

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.

<<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.

<<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
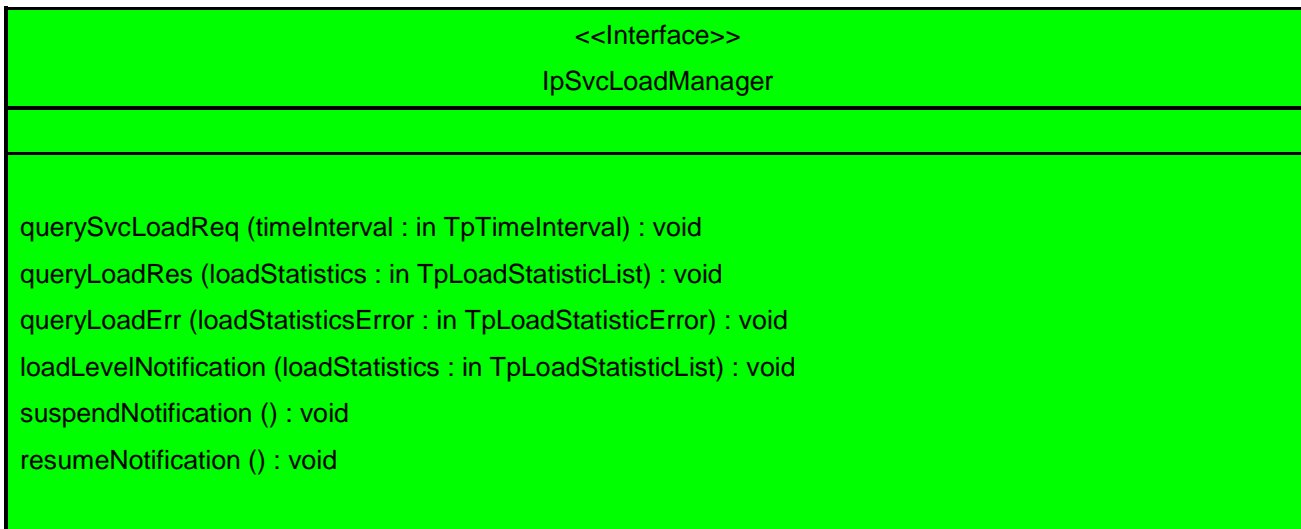
Method
querySvcLoadReq()

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.

<<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.

<<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.

<<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.

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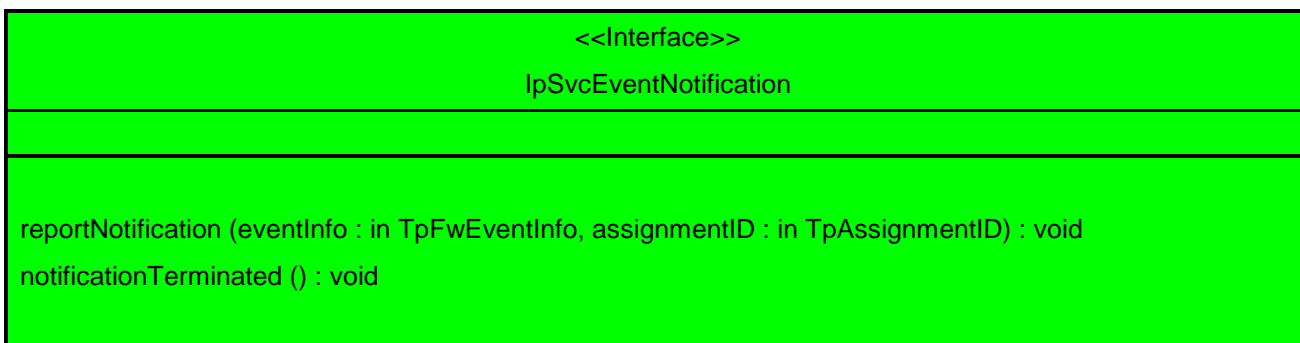
Method
reportNotification()

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

# 8.4		State Transition Diagrams

This section contains the State Transition Diagrams for the objects that implement the Framework interfaces on the gateway side. The State Transition Diagrams show the behaviour of these objects. For each state the methods that can be invoked by the client are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the client also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

## 8.4.1		Service Registration State Transition Diagrams

### 8.4.1.1		State Transition Diagrams for IpFwServiceRegistration
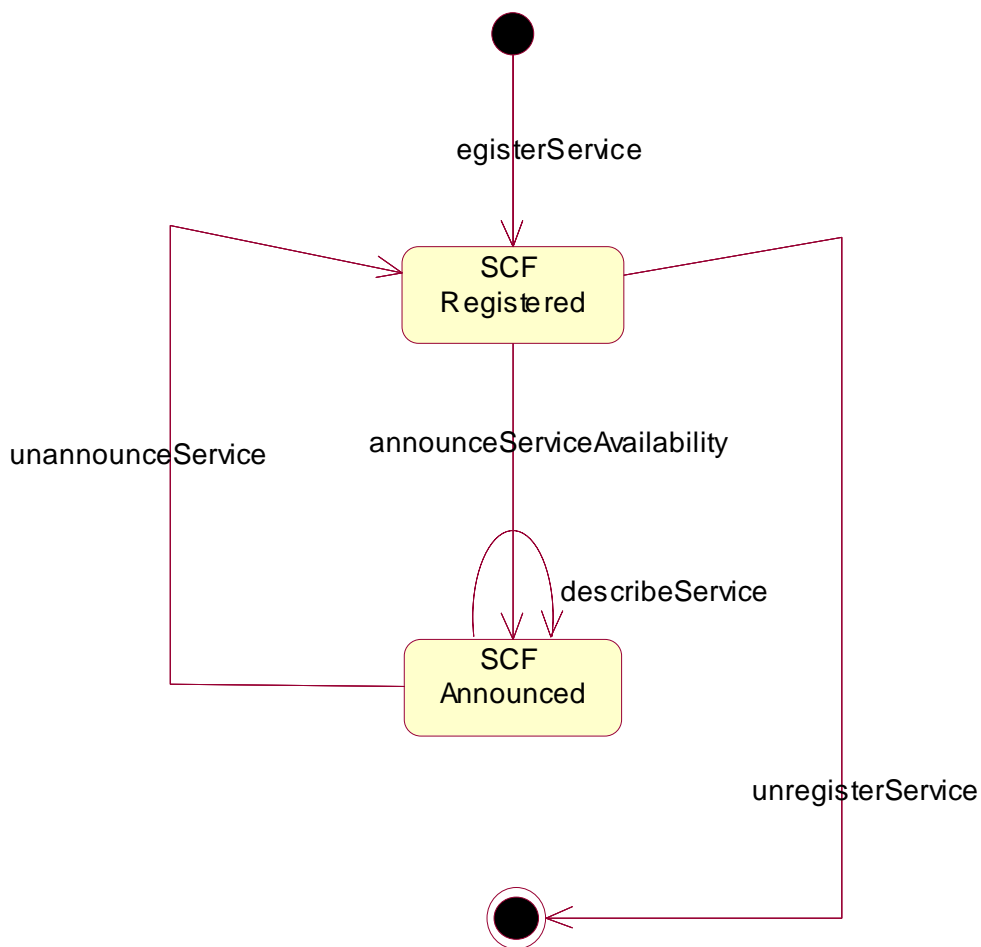
Figure : State Transition Diagram for IpFwServiceRegistration

8.4.1.1.1   SCF Registered State

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

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

8.4.1.1.2   SCF Announced State

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

8.4.2   Service Instance Lifecycle Manager State Transition Diagrams

There are no State Transition Diagrams defined for Service Instance Lifecycle Manager

8.4.3 Service Discovery State Transition Diagrams

There are no State Transition Diagrams defined for Service Discovery

8.4.4 Integrity Management State Transition Diagrams

8.4.4.1 State Transition Diagrams for IpFwLoadManager

Figure : State Transition Diagram for IpFwLoadManager

8.4.4.1.1 Idle State

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

8.4.4.1.2 Notification Suspended State

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

8.4.4.1.3 Active State

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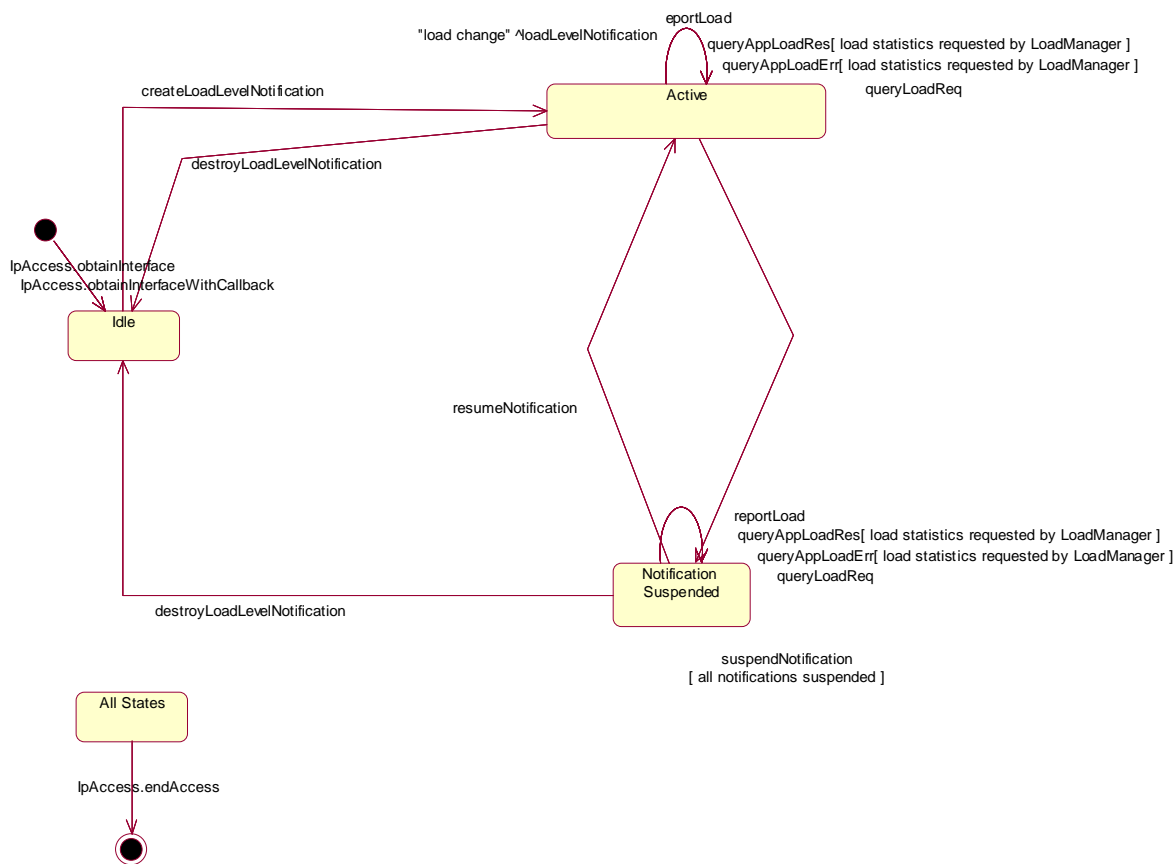
8.4.5 Event Notification State Transition Diagrams

There are no State Transition Diagrams defined for Event Notification

9 Service Properties

9.1 Service Property Types

The service type defines which properties the supplier of an SCF supplier shall provide when he registers an SCF.

At Service Registration the properties of a type shall be interpreted as the set of values that can be supported by the service. If a service type has a certain property (e.g. "CAN_DO_SOMETHING"), a service registers with a property value of {"true", "false"}. This means that the SCS is able to support Service instances where this property is used or allowed and instances where this property is not used or allowed. This clarifies why sets of values shall be used for the property values instead of primitive types.

At establishment of the Service Level Agreement the property can then be set to the value of the specific agreement. The context of the Service Level Agreement thus restricts the set of property values of the SCS and will thus lead to a sub-set of the service property values. When the correct SCF is instantiated during the discovery and selection procedure (see Note), the Service Properties shall thus be interpreted as the requested property values.

NOTE: This is achieved through the createServiceManager() operation in the Service Instance Lifecycle Manager interface.

All property values are represented by an array of strings. The following table shows all supported property types.

Property type name	Description	Example value (array of strings)	Interpretation of example value
BOOLEAN_SET	set of Booleans	{"FALSE"}	The set of Booleans consisting of the Boolean "false".
INTEGER_SET	set of integers	{"1", "2", "5", "7"}	The set of integers consisting of the integers 1, 2, 5 and 7.
STRING_SET	set of strings	{"Sophia", "Rijen"}	The set of strings consisting of the string "Sophia" and the string "Rijen"
ADDRESSRANGE_SET	set of address ranges	{"123??*", "*.ericsson.se"}	The set of address ranges consisting of ranges 123??* and *.ericsson.se.
INTEGER_INTERVAL	interval of integers	{"5", "100"}	The integers that are between or equal to 5 and 100.
STRING_INTERVAL	interval of strings	{"Rijen", "Sophia"}	The strings that are between or equal to the strings "Rijen" and "Sophia", in lexicographical order.
INTEGER_INTEGER_MAP	map from integers to integers	{"1", "10", "2", "20", "3", "30"}	The map that maps 1 to 10, 2 to 20 and 3 to 30.

The bounds of the string interval and the integer interval types may hold the reserved value "UNBOUNDED". If the left bound of the interval holds the value "UNBOUNDED", the lower bound of the interval is the smallest value supported by the type. If the right bound of the interval holds the value "UNBOUNDED", the upper bound of the interval is the largest value supported by the type.

9.2 General Service Properties

Each service instance has the following general properties:

- Service Name

- Service Version

- Service Instance ID

- Service Instance Description

- Product Name

- Product Version

- Supported Interfaces

9.2.1 Service Name

This property contains the name of the service, e.g. "UserLocation", "UserLocationCamel", "UserLocationEmergency" or "UserStatus".

9.2.2 Service Version

This property contains the version of the APIs, to which the service is compliant, e.g. "2.1".

9.2.3 Service Instance ID

This property uniquely identifies a specific instance of the service. The Framework generates this property.

9.2.4 Service Instance Description

This property contains a textual description of the service.

9.2.5 Product Name

This property contains the name of the product that provides the service, e.g. "Find It", "Locate.com".

9.2.6 Product Version

This property contains the version of the product that provides the service, e.g. "3.1.11".

9.2.7 Supported Interfaces

This property contains a list of strings with interface names that the service supports, e.g. "IpUserLocation", "IpUserStatus".

9.2.8 Operation Set

Property	Type	Description
P_OPERATION_SET	STRING_SET	Specifies set of the operations the SCS supports. The notation to be used is : {"Interface1.operation1","Interface1.operation2", "Interface2.operation1"}, e.g.: {"IpCall.createCall","IpCall.routeReq"}.

10 Data Definitions

This clause provides the Framework specific data definitions necessary to support the OSA interface specification.

The general format of a data definition specification is the following:

- − Data type, that shows the name of the data type;

- − Description, that describes the data type;

- − Tabular specification, that specifies the data types and values of the data type;

- − Example, if relevant, shown to illustrate the data type.

10.1 Common Framework Data Definitions

10.1.1 TpClientAppID

This is an identifier for the client application. It is used to identify the client to the Framework. This data type is identical to TpString and is defined as a string of characters that uniquely identifies the application. The content of this string shall be unique for each OSA API implementation (or unique for a network operator's domain). This unique identifier shall be negotiated with the OSA operator and the application shall use it to identify itself.

10.1.2 TpClientAppIDList

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

10.1.3   TpDomainID

Defines the `Tagged Choice of Data Elements` that specify either the Framework or the type of entity attempting to access the Framework.

	Tag Element Type	
	TpDomainIDType	

Tag Element Value	Choice Element Type	Choice Element Name
P_FW	TpFwID	FwID
P_CLIENT_APPLICATION	TpClientAppID	ClientAppID
P_ENT_OP	TpEntOpID	EntOpID
P_SERVICE_INSTANCE	TpServiceInstanceID	ServiceID
P_SERVICE_SUPPLIER	TpServiceSupplierID	ServiceSupplierID

10.1.4   TpDomainIDType

Defines either the Framework or the type of entity attempting to access the Framework.

Name	Value	Description
P_FW	0	The Framework
P_CLIENT_APPLICATION	1	A client application
P_ENT_OP	2	An enterprise operator
P_SERVICE_INSTANCE	3	A service instance
P_SERVICE_SUPPLIER	4	A service supplier

10.1.5   TpEntOpID

This data type is identical to TpString and is defined as a string of characters that identifies an enterprise operator. In conjunction with the application it uniquely identifies the enterprise operator which uses a particular OSA Service Capability Feature (SCF).

10.1.6   TpPropertyName

This data type is identical to `TpString`. It is the name of a generic "property".

10.1.7   TpPropertyValue

This data type is identical to `TpString`. It is the value (or the list of values) associated with a generic "property".

10.1.8   TpProperty

This data type is a `Sequence of Data Elements` which describes a generic "property". It is a structured data type consisting of the following {name,value} pair:

Sequence Element Name	Sequence Element Type
PropertyName	TpPropertyName
PropertyValue	TpPropertyValue

10.1.9   TpPropertyList

This data type defines a `Numbered List of Data Elements` of type TpProperty.

10.1.10 TpEntOpIDList

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

10.1.11 TpFwID

This data type is identical to `TpString` and identifies the Framework to a client application (or Service Capability Feature)

10.1.12 TpService

This data type is a Sequence of Data Elements which describes a registered SCFs. It is a structured type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceID	TpServiceID	
ServiceDescription	TpServiceDescription	This field contains the description of the service

10.1.13 TpServiceList

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

10.1.14 TpServiceDescription

This data type is a Sequence of Data Elements which describes a registered SCF. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypeName	TpServiceTypeName	
ServicePropertyList	TpServicePropertyList	

10.1.15 TpServiceID

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

10.1.16 TpServiceIDList

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

~~11.1.17~~ ~~TpServiceIDRef~~

~~Defines a Reference to type TpServiceId.~~

~~11.1.18~~10.1.17   TpServiceInstanceID

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

11.1.1910.1.18   TpServiceSpecString

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the name of an SCF specialization interface. Other network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF specialization
P_CALL	The Call specialization of the of the User Interaction SCF

11.1.2010.1.19   TpServiceTypeProperty

This data type is a `Sequence of Data Elements` which describes a service property associated with a service type. It defines the name and mode of the service property, and also the service property type: e.g. Boolean, integer. It is similar to, but distinct from, TpServiceProperty. The latter is associated with an actual service: it defines the service property's name and mode, but also defines the list of values assigned to it.

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServiceTtypePropertyMode	TpServiceTypePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

11.1.2110.1.20   TpServiceTypePropertyList

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

11.1.2210.1.21   TpServiceTypePropertyMode

This type defines SCF property modes.

Name	Value	Documentation
NORMAL	0	The value of the corresponding SCF property type may optionally be provided
MANDATORY	1	The value of the corresponding SCF property type shall be provided at service registration time
READONLY	2	The value of the corresponding SCF property type is optional, but once given a value it can not be modified/restricted by a service level agreement
MANDATORY_READONLY	3	The value of the corresponding SCF property type shall be provided but can not subsequently be modified/restricted by a service level agreement.

11.1.2310.1.22   TpServicePropertyTypeName

This data type is identical to TpString and describes a valid SCF property name. The valid SCF property names are listed in the SCF data definition.

11.1.2410.1.23   TpServicePropertyName

This data type is identical to TpString. It defines a valid SCF property name.

11.1.2510.1.24   TpServicePropertyNameList

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

11.1.2610.1.25 TpServicePropertyValue

This data type is identical to TpString and describes a valid value of a SCF property.

11.1.2710.1.26 TpServicePropertyValueList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyValue

11.1.2810.1.27 TpServiceProperty

This data type is a Sequence of Data Elements which describes an "SCF property". It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServicePropertyValueList	TpServicePropertyValueList	

11.1.2910.1.28 TpServicePropertyList

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

11.1.3010.1.29 TpServiceSupplierID

This is an identifier for a service supplier. It is used to identify the supplier to the Framework. This data type is identical to TpString.

11.1.3110.1.30 TpServiceTypeDescription

This data type is a Sequence_of_Data_Elements which describes an SCF type. It is a structured data type. It consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypePropertyList	TpServiceTypePropertyList	a sequence of property name and property mode tuples associated with the SCF type
ServiceTypeNameList	TpServiceTypeNameList	the names of the super types of the associated SCF type
AvailableOrUnavailableEnabledOrDisabled	TpBoolean	an indication whether the SCF type is available enabled (true) or unavailable disabled (false)

~~11.1.32~~10.1.31   TpServiceTypeName

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the type of an SCF interface. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF name
P_GENERIC_CALL_CONTROL	The name of the Generic Call Control SCF
P_MULTI_PARTY_CALL_CONTROL	The name of the MultiParty Call Control SCF
P_MULTI_MEDIA_CALL_CONTROL	The name of the MultiMedia Call Control SCF
P_CONFERENCE_CALL_CONTROL	The name of the Conference Call Control SCF
P_USER_INTERACTION	The name of the User Interaction SCFs
P_TERMINAL_CAPABILITIES	The name of the Terminal Capabilities SCF
P_USER_LOCATION	The name of the User Location SCF
P_USER_LOCATION_CAMEL	The name of the Network User Location SCF
P_USER_LOCATION_EMERGENCY	The name of the User Location Emergency SCF
P_USER_STATUS	The name of the User Status SCF
P_DATA_SESSION_CONTROL	The name of the Data Session Control SCF
P_GENERIC_MESSAGING	The name of the Generic Messaging SCF
P_CONNECTIVITY_MANAGER	The name of the Connectivity Manager SCF
P_CHARGING	The name of the Charging SCF
P_ACCOUNT_MANAGEMENT	The name of the Account Management SCF

~~11.1.33~~10.1.32   TpServiceTypeNameList

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

~~11.1.34~~10.1.33   TpSubjectType

Defines the subject of a query/notification request/result.

Name	Value	Description
P_SUBJECT_UNDEFINED	0	The subject is neither the framework nor the client application
P_SUBJECT_CLIENT_APP	1	The subject is the client application
P_SUBJECT_FW	2	The subject is the framework

10.2   Event Notification Data Definitions

10.2.1   TpFwEventName

Defines the name of event being notified.

Name	Value	Description
P_EVENT_FW_NAME_UNDEFINED	0	Undefined
P_EVENT_FW_SERVICE_AVAILABLE	1	Notification of SCS(s) available
P_EVENT_FW_SERVICE_UNAVAILABLE	2	Notification of SCS(s) becoming unavailable

10.2.2 TpFwEventCriteria

Defines the `Tagged Choice of Data Elements` that specify the criteria for an event notification to be generated.

	Tag Element Type	
	TpFwEventName	

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	TpString	EventNameUndefined
P_EVENT_FW_ SERVICE_AVAILABLE	TpServiceTypeNameList	ServiceType-Name-List
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceTypeNameList	UnavailableServiceType-Name-List

10.2.3 TpFwEventInfo

Defines the `Tagged Choice of Data Elements` that specify the information returned to the application in an event notification.

	Tag Element Type	
	TpFwEventName	

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	TpString	EventNameUndefined
P_EVENT_FW_ SERVICE_AVAILABLE	TpServiceIDList	ServiceID—List
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceIDList	UnavailableServiceID List

10.3 Trust and Security Management Data Definitions

10.3.1 TpAccessType

This data type is identical to a TpString. This identifies the type of access interface requested by the client application. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Network operators can define their own access interfaces to satisfy client requirements for different types of access. These can be selected using the TpAccessType, but should be preceded by the string "SP_". The following value is defined:

String Value	Description
P_OSA_ACCESS	Access using the OSA Access Interfaces: IpAccess and IpClientAccess

10.3.2   TpAuthType

This data type is identical to a TpString. It identifies the type of authentication mechanism requested by the client. It provides Network operators and client's with the opportunity to use an alternative to the OSA API Level Authentication interface. This can for example be an implementation specific authentication mechanism, e.g. CORBA Security, or a proprietary Authentication interface supported by the Network Operator. OSA API Level Authentication is the default authentication method. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined:

String Value	Description
P_OSA_AUTHENTICATION	Authenticate using the OSA API Level Authentication Interfaces: IpAPILevelAuthentication and IpClientAPILevelAuthentication
P_AUTHENTICATION	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.

10.3.3   TpEncryptionCapability

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

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

10.3.4   TpEncryptionCapabilityList

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

10.3.5   TpEndAccessProperties

This data type is of type TpPropertyList. It identifies the actions that the Framework should perform when an application or service capability feature entity ends its access session (e.g. existing service capability or application sessions may be stopped, or left running).

10.3.6   TpAuthDomain

This is Sequence of Data Elements containing all the data necessary to identify a domain: the domain identifier, and a reference to the authentication interface of the domain

Sequence Element Name	Sequence Element Type	Description
DomainID	TpDomainID	Identifies the domain for authentication. This identifier is assigned to the domain during the initial contractual agreements, and is valid during the lifetime of the contract.
AuthInterface	IpInterfaceRef	Identifies the authentication interface of the specific entity. This data element has the same lifetime as the domain authentication process, i.e. in principle a new interface reference can be provided each time a domain intents intends to access another.

10.3.7 TpInterfaceName

This data type is identical to a TpString, and is defined as a string of characters that identify the names of the Framework SCFs that are to be supported by the OSA API. Other Network operator specific SCFs may also be used, but should be preceded by the string "SP_". The following values are defined.

Character String Value	Description
P_DISCOVERY	The name for the Discovery interface.
P_EVENT_NOTIFICATION	The name for the Event Notification interface.
P_OAM	The name for the OA&M interface.
P_LOAD_MANAGER	The name for the Load Manager interface.
P_FAULT_MANAGER	The name for the Fault Manager interface.
P_HEARTBEAT_MANAGEMENT	The name for the Heartbeat Management interface.
P_SERVICE_AGREEMENT_MANAGEMENT	The name of the Service Agreement Management interface.
P_REGISTRATION	The name for the Service Registration interface.
P_ENT_OP_ACCOUNT_MANAGEMENT	The name for the Service Subscription: Enterprise Operator Account Management interface.
P_ENT_OP_ACCOUNT_INFO_QUERY	The name for the Service Subscription: Enterprise Operator Account Information Query interface.
P_SVC_CONTRACT_MANAGEMENT	The name for the Service Subscription: Service Contract Management interface.
P_SVC_CONTRACT_INFO_QUERY	The name for the Service Subscription: Service Contract Information Query interface.
P_CLIENT_APP_MANAGEMENT	The name for the Service Subscription: Client Application Management interface.
P_CLIENT_APP_INFO_QUERY	The name for the Service Subscription: Client Application Information Query interface.
P_SVC_PROFILE_MANAGEMENT	The name for the Service Subscription: Service Profile Management interface.
P_SVC_PROFILE_INFO_QUERY	The name for the Service Subscription: Service Profile Information Query interface.

10.3.8 TpInterfaceNameList

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

10.3.9 TpServiceToken

This data type is identical to a TpString, and identifies a selected SCF. This is a free format text token returned by the Framework, which can be signed as part of a service agreement. This will contain Network operator specific information relating to the service level agreement. The serviceToken has a limited lifetime, which is the same as the lifetime of the service agreement in normal conditions. If something goes wrong the serviceToken expires, and any method accepting the serviceToken will return an error code (P_INVALID_SERVICE_TOKEN). Service Tokens will automatically expire if the client or Framework invokes the endAccess method on the other's corresponding access interface.

10.3.10 TpSignatureAndServiceMgr

This is a Sequence of Data Elements containing the digital signature of the Framework for the service agreement, and a reference to the SCF manager interface of the SCF.

Sequence Element Name	Sequence Element Type
DigitalSignature	TpOctetSet~~TpString~~
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 SCF manager interface for the selected SCF.

10.3.11 TpSigningAlgorithm

This data type is identical to a TpString, and is defined as a string of characters that identify the signing algorithm that shall be used. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.

String Value	Description
NULL	An empty (NULL) string indicates no signing algorithm is required
P_MD5_RSA_512	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit key.
P_MD5_RSA_1024	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public- key cryptography system using a 1024-bit key

10.4   Integrity Management Data Definitions

10.4.1   TpActivityTestRes

This type is identical to TpString and is an implementation specific result. The values in this data type are "Available" or "Unavailable".

10.4.2   TpFaultStatsRecord

This defines the set of records to be returned giving fault information for the requested time period.

Sequence Element Name	Sequence Element Type
Period	TpTimeInterval
FaultStatsSet	TpFaultStatsSet

10.4.3   TpFaultStats

This defines the sequence of data elements which provide the statistics on a per fault type basis.

Sequence Element Name	Sequence Element Type	Description
Fault	TpInterfaceFault	
Occurrences	TpInt32	The number of separate instances of this fault
MaxDuration	TpInt32	The number of seconds duration of the longest fault
TotalDuration	TpInt32	The cumulative duration (all occurrences)
NumberOfClientsAffected	TpInt32	The number of clients informed of the fault by the Fw

Occurrences is the number of separate instances of this fault during the period. MaxDuration and TotalDuration are the number of seconds duration of the longest fault and the cumulative total during the period. NumberOfClientsAffected is the number of clients informed of the fault by the Framework.

10.4.4   TpFaultStatisticsError

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

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

10.4.5 TpFaultStatsSet

This data type defines a `Numbered Set of Data Elements` of type TpFaultStats

10.4.6 TpActivityTestID

This data type is identical to a TpInt32, and is used as a token to match activity test requests with their results..

10.4.7 TpInterfaceFault

Defines the cause of the interface fault detected.

Name	Value	Description
INTERFACE_FAULT_UNDEFINED	0	Undefined
INTERFACE_FAULT_LOCAL_FAILURE	1	A fault in the local API software or hardware has been detected
INTERFACE_FAULT_GATEWAY_FAILURE	2	A fault in the gateway API software or hardware has been detected
INTERFACE_FAULT_PROTOCOL_ERROR	3	An error in the protocol used on the client-gateway link has been detected

10.4.8 TpSvcUnavailReason

Defines the reason why a SCF is unavailable.

Name	Value	Description
SERVICE_UNAVAILABLE_UNDEFINED	0	Undefined
SERVICE_UNAVAILABLE_LOCAL_FAILURE	1	The Local API software or hardware has failed
SERVICE_UNAVAILABLE_GATEWAY_FAILURE	2	The gateway API software or hardware has failed
SERVICE_UNAVAILABLE_OVERLOADED	3	The SCF is fully overloaded
SERVICE_UNAVAILABLE_CLOSED	4	The SCF has closed itself (e.g. to protect from fraud or malicious attack)

10.4.9 TpFWUnavailReason

Defines the reason why the Framework is unavailable.

Name	Value	Description
FW_UNAVAILABLE_UNDEFINED	0	Undefined
FW_UNAVAILABLE_LOCAL_FAILURE	1	The Local API software or hardware has failed
FW_UNAVAILABLE_GATEWAY_FAILURE	2	The gateway API software or hardware has failed
FW_UNAVAILABLE_OVERLOADED	3	The Framework is fully overloaded
FW_UNAVAILABLE_CLOSED	4	The Framework has closed itself (e.g. to protect from fraud or malicious attack)
FW_UNAVAILABLE_PROTOCOL_FAILURE	5	The protocol used on the client-gateway link has failed

10.4.10 TpLoadLevel

Defines the Sequence of Data Elements that specify load level values.

Name	Value	Description
LOAD_LEVEL_NORMAL	0	Normal load
LOAD_LEVEL_OVERLOAD	1	Overload
LOAD_LEVEL_SEVERE_OVERLOAD	2	Severe Overload

10.4.11 TpLoadThreshold

Defines the Sequence of Data Elements that specify the load threshold value. The actual load threshold value is application and SCF dependent, so is their relationship with load level.

Sequence Element Name	Sequence Element Type
LoadThreshold	TpFloat

10.4.12 TpLoadInitVal

Defines the Sequence of Data Elements that specify the pair of load level and associated load threshold value.

Sequence Element Name	Sequence Element Type
LoadLevel	TpLoadLevel
LoadThreshold	TpLoadThreshold

10.4.13 TpLoadPolicy

Defines the load balancing policy.

Sequence Element Name	Sequence Element Type
LoadPolicy	TpString

10.4.14 TpLoadStatistic

Defines the Sequence of Data Elements that represents a load statistic record for a specific entity (i.e. Framework, service or application) at a specific date and time.

Sequence Element Name	Sequence Element Type
LoadStatisticEntityID	TpLoadStatisticEntityID
TimeStamp	TpDateAndTime
LoadStatisticInfo	TpLoadStatisticInfo

10.4.15 TpLoadStatisticList

Defines a Numbered List of Data Elements of type TpLoadStatistic.

10.4.16 TpLoadStatisticData

Defines the Sequence of Data Elements that represents load statistic information

Sequence Element Name	Sequence Element Type
LoadValue (see Note)	TpFloat
LoadLevel	TpLoadLevel
NOTE: LoadValue is expressed as a percentage.	

10.4.17 TpLoadStatisticEntityID

Defines the `Tagged Choice of Data Elements` that specify the type of entity (i.e. service, application or Framework) providing load statistics.

	Tag Element Type	
	TpLoadStatisticEntityType	

Tag Element Value	Choice Element Type	Choice Element Name
P_LOAD_STATISTICS_FW_TYPE	TpFwID	FrameworkID
P_LOAD_STATISTICS_SVC_TYPE	TpServiceID	ServiceID
P_LOAD_STATISTICS_APP_TYPE	TpClientAppID	ClientAppID

10.4.18 TpLoadStatisticEntityType

Defines the type of entity (i.e. service, application or Framework) supplying load statistics.

Name	Value	Description
P_LOAD_STATISTICS_FW_TYPE	0	Framework-type load statistics
P_LOAD_STATISTICS_SVC_TYPE	1	Service-type load statistics
P_LOAD_STATISTICS_APP_TYPE	2	Application-type load statistics

10.4.19 TpLoadStatisticInfo

Defines the `Tagged Choice of Data Elements` that specify the type of load statistic information (i.e. valid or invalid).

	Tag Element Type	
	TpLoadStatisticInfoType	

Tag Element Value	Choice Element Type	Choice Element Name
P_LOAD_STATISTICS_VALID	TpLoadStatisticData	LoadStatisticData
P_LOAD_STATISTICS_INVALID	TpLoadStatisticError	LoadStatisticError

10.4.20 TpLoadStatisticInfoType

Defines the type of load statistic information (i.e. valid or invalid).

Name	Value	Description
P_LOAD_STATISTICS_VALID	0	Valid load statistics
P_LOAD_STATISTICS_INVALID	1	Invalid load statistics

10.4.21 TpLoadStatisticError

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

Name	Value	Description
P_LOAD_INFO_ERROR_UNDEFINED	0	Undefined error
P_LOAD_INFO_UNAVAILABLE	1	Load statistics unavailable

10.5 Service Subscription Data Definitions

10.5.1 TpPropertyName

This data type is identical to `TpString`. It is the name of a generic "property".

10.5.2 TpPropertyValue

This data type is identical to `TpString`. It is the value (or the list of values) associated with a generic "property".

10.5.3 TpProperty

This data type is a `Sequence of Data Elements` which describes a generic "property". It is a structured data type consisting of the following {name,value} pair:

Sequence Element Name	Sequence Element Type
PropertyName	TpPropertyName
PropertyValue	TpPropertyValue

10.5.4 TpPropertyList

This data type defines a `Numbered List of Data Elements` of type TpProperty.

10.5.5 TpEntOpProperties

This data type is of type TpPropertyList. It identifies the list of properties associated with an enterprise operator: e.g. name, organisation, address, phone, e-mail, fax, payment method (credit card, bank account).

10.5.6 TpEntOp

This data type is a `Sequence of Data Elements` which describes an enterprise operator. It is a structured data type, consisting of a unique "enterprise operator ID" and a list of "enterprise operator properties", as follows:

Sequence Element Name	Sequence Element Type
EntOpID	TpEntOpID
EntOpProperties	TpEntOpProperties

10.5.7 TpServiceContractID

This data type is identical to `TpString`. It uniquely identifies the contract, between an enterprise operator and the Framework, for the use of a Parlay service by the enterprise.

10.5.8 TpServiceContractIDList

This data type defines a `Numbered List of Data Elements` of type TpServiceContractID.

10.5.9 TpPersonName

This data type is identical to `TpString`. It is the name of a generic "person".

10.5.10 TpPostalAddress

This data type is identical to `TpString`. It is the mailing address of a generic "person".

10.5.11 TpTelephoneNumber

This data type is identical to `TpString`. It is the telephone number of a generic "person".

10.5.12 TpEmail

This data type is identical to `TpString`. It is the email address of a generic "person".

10.5.13 TpHomePage

This data type is identical to `TpString`. It is the web address of a generic "person".

10.5.14 TpPersonProperties

This data type is of type TpPropertyList. It identifies the list of additional properties, other than those listed above, that can be associated with a generic "person".

10.5.15 TpPerson

This data type is a `Sequence of Data Elements` which describes a generic "person": e.g. a billing contact, a service requestor. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
PersonName	TpPersonName
PostalAddress	TpPostalAddress
TelephoneNumber	TpTelephoneNumber
Email	TpEmail
HomePage	TpHomePage
PersonProperties	TpPersonProperties

10.5.16 TpServiceStartDate

This is of type `TpDateAndTime`. It identifies the contractual start date and time for the use of a Parlay service by an enterprise or an enterprise Subscription Assignment Group (SAG).

10.5.17 TpServiceEndDate

This is of type `TpDateAndTime`. It identifies the contractual end date and time for the use of a Parlay service by an enterprise or an enterprise Subscription Assignment Group (SAG).

10.5.18 TpServiceRequestor

This is of type TpPerson. It identifies the enterprise person requesting use of a Parlay service: e.g. the enterprise operator.

10.5.19 TpBillingContact

This is of type TpPerson. It identifies the enterprise person responsible for billing issues associated with an enterprise's use of a Parlay service.

10.5.20 TpServiceSubscriptionProperties

This is of type TpServicePropertyList. It specifies a subset of all available service properties and service property values that apply to an enterprise's use of a Parlay service.

10.5.21 TpServiceContract

This data type is a `Sequence of Data Elements` which represents a service contract. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceContractID	TpServiceContractID
ServiceContractDescription	TpServiceContractDescription

10.5.22 TpServiceContractDescription

This data type is a `Sequence of Data Elements` which describes a service contract. This contract should conform to a previously negotiated high-level agreement (regarding Parlay services, their usage and the price, etc.), if any, between the enterprise operator and the framework operator. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceRequestor	TpServiceRequestor
BillingContact	TpBillingContact
ServiceStartDate	TpServiceStartDate
ServiceEndDate	TpServiceEndDate
ServiceTypeName	TpServiceTypeName
ServiceID	TpServiceID
ServiceSubscriptionProperties	TpServiceSubscriptionProperties

10.5.23 TpClientAppProperties

This is of type TpPropertyList. The client application properties is a list of {name,value} pairs, for bilateral agreement between the enterprise operator and the Framework.

10.5.24 TpClientAppDescription

This data type is a `Sequence of Data Elements` which describes an enterprise client application. It is a structured data type, consisting of a unique "client application ID", password and a list of "client application properties:

Sequence Element Name	Sequence Element Type
ClientAppID	TpClientAppID
ClientAppProperties	TpClientAppProperties

10.5.25 TpSagID

This data type is identical to `TpString`. It uniquely identifies a Subscription Assignment Group (SAG) of client applications within an enterprise.

10.5.26 TpSagIDList

This data type defines a `Numbered List of Data Elements` of type TpSagID.

10.5.27 TpSagDescription

This data type is identical to `TpString`. It describes a SAG: e.g. a list of identifiers of the constituent client applications, the purpose of the "grouping".

10.5.28 TpSag

This data type is a `Sequence of Data Elements` which describes a Subscription Assignment Group (SAG) of client applications within an enterprise. It is a structured data type consisting of a unique SAG ID and a description:

Sequence Element Name	Sequence Element Type
SagID	TpSagID
SagDescription	TpSagDescription

10.5.29 TpServiceProfileID

This data type is identical to `TpString`. It uniquely identifies the service profile, which further constrains how an enterprise SAG uses a Parlay service.

10.5.30 TpServiceProfileIDList

This data type defines a `Numbered List of Data Elements` of type TpServiceProfileID.

10.5.31 TpServiceProfile

This data type is a `Sequence of Data Elements` which represents a Service Profile. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceProfileID	TpServiceProfileID
ServiceProfileDescription	TpServiceProfileDescription

10.5.32 TpServiceProfileDescription

This data type is a `Sequence of Data Elements` which describes a Service Profile. A service contract contains one or more Service Profiles, one for each SAG in the enterprise operator domain. A service profile is a restriction of the service contract in order to provide restricted service features to a SAG. It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type
ServiceContractID	TpServiceContractID
ServiceStartDate	TpServiceStartDate
ServiceEndDate	TpServiceEndDate
ServiceTypeName	TpServiceTypeName
ServiceSubscriptionProperties	TpServiceSubscriptionProperties

11 Exception Classes

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

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

Each exception class contains the following structure:

Structure Element Name	Structure Element Type	Structure Element Description
ExtraInformation	TpString	Carries extra information to help identify the source of the exception, e.g. a parameter name

Annex A (normative):
OMG IDL Description of Framework

The OMG IDL representation of this interface specification is contained in text files (fw_data.idl, fw_if_access.idl, fw_if_app.idl, fw_if_service.idl contained in archive ??????.ZIP) which accompany the present document.

Annex B (informative): Differences between this draft and 3GPP TS 29.198 R99

The following is a list of the differences between the present document and 3GPP TS 29.198 R99, for those items which are common to both documents. Any new interfaces/methods with respect to Release 99 are not listed.

B.1 IpServiceRegistration

Interface Class IpServiceRegistration in R99 renamed IpFwServiceRegistration

announceServiceAvailability (serviceID : in TpServiceID, serviceFactoryRef : in Ip~~Svc~~Factory~~Service~~Ref) : TpResult

unannounceService(serviceID : in TpServiceID) : TpResult

B.2 IDL Namespace

IDL namespace has been completely revised. All data types are under org::csapi::fw. 4 further namespaces have been created: org::csapi::fw::fw_access, org::csapi::fw::fw_app, org::csapi::fw::fw_entop and org::csapi::fw::fw_service

B.3 IpAccess, IpAppAccess

Method accessCheck() has been removed

~~accessCheck(serviceToken: in TpServiceToken,securityContext: in TpStringTpSecurityContext, securityDomain: in TpStringTpSecurityDomain, group : in TpStringTpSecurityGroup, serviceAccessTypes: in TpStringTpServiceAccessType, serviceAccessControl: out TpServiceAccessControlRef): TpResult~~

Methods listInterfaces(), releaseInterface() added:

listInterfaces () : TpInterfaceNameList

releaseInterface (interfaceName : in TpInterfaceName) : void

obtainInterfaceWithCallback (interfaceName : in TpInterfaceName, ~~app~~clientInterface : in IpInterfaceRef) : IpInterfaceRef

Methods associated with signing service agreements moved to new interface, IpServiceAgreementManagement, in which new method initiateSignServiceAgreement() is added:

initiateSignServiceAgreement (serviceToken : in TpServiceToken) : void

IpAppAccess renamed to IpClientAccess

terminateAccess (terminationText : in TpString, signingAlgorithm : in TpSigningAlgorithm, digitalSignature : in ~~TpString~~TpOctetSet) : void

B.4 IpAPILevelAuthentication, IpAppAPILevelAuthentication

Interfaces IpAuthentication and IpAppAuthentication renamed as IpAPILevelAuthentication and IpAppAPILevelAuthentication. New interface IpAuthentication added. IpAPILevelAuthentication inherits from IpAuthentication.

selectEncryptionMethod~~selectAuthMethod~~ (~~auth~~encryptionCaps : in TpEncryptionCapabilityList~~TpAuthCapabilityList~~, prescribedMethod : out Tp~~Auth~~EncryptionCapabilityRef) : TpResult

The following method is added to both interfaces:

authenticationSucceeded () : TpResult

IpAppAPILevelAuthentication renamed IpClientAPIAuthentication

authenticate (~~prescribedMethod : in TpAuthCapability,~~ challenge : in ~~TpString~~TpOctetSet) : ~~TpString~~TpOctetSet

B.5 New IpAuthentication

requestAccess (accessType : in TpAccessType, ~~app~~clientAccessInterface : in IpInterfaceRef, fwAccessInterface : out IpInterfaceRefRef) : TpResult   added.

B.6 IpInitial

~~requestAccess (accessType : in TpAccessType, appAccessInterface : in IpInterfaceRef, fwAccessInterface : out IpInterfaceRefRef) : TpResult~~ deleted from interface.

initiateAuthentication (~~app~~clientDomain : in TpAuthDomain, authType : in TpAuthType) : TpAuthDomain

B.7 IpAppLoadManager, IpLoadManager

~~disableLoadControl (serviceIDs : in TpServiceIDList) : TpResult~~

~~enableLoadControl (loadStatistics : in TpLoadStatisticList) : TpResult~~

loadLevelNotification(loadStatistics : in TpLoadStatisticList) : TpResult

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

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

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

B.8 IpSvcFactory

~~getServiceManager~~createServiceManager (application : in Tp~~Client~~App~~Domain~~ID, serviceProperties : in TpServicePropertyList, serviceInstanceID : in TpServiceInstanceID, serviceManager : out IpServiceRefRef) : TpResult

IpSvcFactory renamed IpServiceInstanceLifecycleManager

New method destroyServiceManager added:

destroyServiceManager (serviceInstance : in TpServiceInstanceID) : void

B.9 IpAppFaultManager

The following methods were added:

activityTestErr (activityTestID : in TpActivityTestID) : void

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

appUnavailableInd () : void

B.10 IpFaultManager

The following methods were added:

appActivityTestErr (activityTestID : in TpActivityTestID) : void

appUnavailableInd (serviceID : in TpServiceID) : void

B.11   IpHeartbeat, IpAppHeartbeat

These interfaces are completely revised. Behaviour has changed significantly with respect to R99 so no comparison of methods is useful (though the methods look similar)

B.12   All Interfaces

All methods on IpApp interfaces no longer throw exceptions.

All methods on the other interfaces throw TpCommonExceptions and individual, identified exceptions

All methods now return void or the former out parameter.

B.13   Data Type Changes

TpDomainID

Defines the `Tagged Choice of Data Elements` that specify either the Framework or the type of entity attempting to access the Framework.

	Tag Element Type	
	TpDomainIDType	

Tag Element Value	Choice Element Type	Choice Element Name
P_FW	TpFwID	FwID
P_CLIENT_APPLICATION	TpClientAppID	ClientAppID
P_ENT_OP	TpEntOpID	EntOpID
P_SERVICE_INSTANCE	TpServiceInstanceID	ServiceID
P_SERVICE_SUPPLIER	TpServiceSupplierID	ServiceSupplierID

TpDomainIDType

Defines either the Framework or the type of entity attempting to access the Framework.

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

TpService

This data type is a Sequence of Data Elements which describes a registered SCFs. It is a structured type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceID	TpServiceID	
~~ServicePropertyList~~ServiceDescription	~~TpServicePropertyList~~TpServiceDescription	This field contains the description of the service

TpServiceID

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies an instance of a SCF interface. The string is automatically generated by the Framework.~~This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies an instance of a SCF interface. The string is automatically generated by the Framework, and comprises a TpUniqueServiceNumber, TpServiceNameString TpServiceTypeName, and a number of relevant TpServiceSpecString, which are concatenated using a forward separator (/) as the separation character.~~

TpServiceIDList

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

~~TpServiceIDRef~~

~~Defines a Reference to type TpServiceId.~~

TpServiceInstanceID

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

~~TpServiceNameString~~

~~This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the name of an SCF interface. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined for OSA release 99.~~

~~Character String Value~~	~~Description~~
~~*NULL*~~	~~An empty (NULL) string indicates no SCF name~~
~~P_CALL_CONTROL~~	~~The name of the Call Control SCF~~
~~P_USER_INTERACTION~~	~~The name of the User Interaction SCFs~~
~~P_TERMINAL_CAPABILITIES~~	~~The name of the Terminal Capabilities SCF~~
~~P_USER_LOCATION_CAMEL~~	~~The name of the Network User Location SCF~~
~~P_USER_STATUS~~	~~The name of the User Status SCF~~
~~P_DATA_SESSION_CONTROL~~	~~The name of the Data Session Control SCF~~

TpServiceSpecString

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the name of an SCF specialization interface. Other network operator specific capabilities may also be used, but should be preceded by the string "SP_".The following values are defined for OSA release 99.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF specialization
P_CALL	The Call specialization of the of the User Interaction SCF

~~TpUniqueServiceNumber~~

~~This data type is identical to a TpString, and is defined as a string of characters that represents a unique number that is used to build the service ID (refer to TpServiceID).~~

TpServiceTypeProperty

This data type is a Sequence of Data Elements which describes a service property associated with a service type. It defines the name and mode of the service property, and also the service property type: e.g. Boolean, integer. It is similar to, but distinct from, TpServiceProperty. The latter is associated with an actual service: it defines the service property's name and mode, but also defines the list of values assigned to it.

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServiceTypePropertyMode	TpServiceTypePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

TpServiceTypePropertyList

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

TpServiceTypePropertyMode

This type is left as a placeholder but is not used in release 99.This defines SCF property modes.

Name	Value	Documentation
NORMAL	0	The value of the corresponding SCF property type may optionally be provided
MANDATORY	1	The value of the corresponding SCF property type shall be provided at service registration time
READONLY	2	The value of the corresponding SCF property type is optional, but once given a value it can ~~may~~ not be modified/restricted by a service level agreement
MANDATORY_READONLY	3	The value of the corresponding SCF property type shall be provided but can not ~~and~~ subsequently ~~it may not~~ be modified/restricted by a service level agreement.

TpServicePropertyTypeName

This data type is identical to TpString and describes a valid SCF property name. The valid SCF property names are listed in the SCF data definition.

TpServicePropertyName

This data type is identical to TpString. It defines a valid S~~F~~CF property name. ~~Valid SCF property names are listed in the SCF data definition.~~

TpServicePropertyNameList

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

TpServicePropertyValue

This data type is identical to TpString and describes a valid value of a SCF property. ~~The valid SCF property values are given in the SCF data definition.~~

TpServicePropertyValueList

This data type defines a Numbered Set of Data Elements of type TpServicePropertyValue

TpServiceProperty

This data type is a Sequence of Data Elements which describes an "SCF property". It is a structured data type which consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServicePropertyValueList	TpServicePropertyValueList	

~~ServicePropertyMode~~	~~TpServicePropertyMode~~	

TpServicePropertyList

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

TpServiceSupplierID

This is an identifier for a service supplier. It is used to identify the supplier to the Framework. This data type is identical to `TpString`.

TpServiceTypeDescription

This type is left as a placeholder but is not used in release 99.

This data type is a Sequence_of_Data_Elements which describes an SCF type. It is a structured data type. It consists of:

Sequence Element Name	Sequence Element Type	Documentation
ServiceTypeProperty List	TpServiceTypePropertyList	a sequence of property name and property mode tuples associated with the SCF type
ServiceTypeNameList	TpServiceTypeNameList	the names of the super types of the associated SCF type
<u>AvailableOrUnavaila ble</u>~~EnabledOrDisable d~~	TpBoolean	<u>an indication whether the SCF type is available (true) or unavailable (false)</u>~~an indication whether the SCF type is enabled or disabled~~

TpServiceTypeName

~~This data type is identical to TpString and describes a valid SCF type name.~~<u>This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the type of an SCF interface. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_".</u>The following values are defined for OSA release 99.

Character String Value	Description
NULL	An empty (NULL) string indicates no SCF name
P_GENERIC_CALL_CONTROL	The name of the Generic Call Control SCF
P_MULTI_PARTY_CALL_CONTROL	The name of the MultiParty Call Control SCF
P_MULTI_MEDIA_CALL_CONTROL	The name of the MultiMedia Call Control SCF
P_CONFERENCE_CALL_CONTROL	The name of the Conference Call Control SCF
P_USER_INTERACTION	The name of the User Interaction SCFs
P_TERMINAL_CAPABILITIES	The name of the Terminal Capabilities SCF
P_USER_LOCATION	The name of the User Location SCF
P_USER_LOCATION_CAMEL	The name of the Network User Location SCF
P_USER_LOCATION_EMERGENCY	The name of the User Location Emergency SCF
P_USER_STATUS	The name of the User Status SCF
P_DATA_SESSION_CONTROL	The name of the Data Session Control SCF
P_GENERIC_MESSAGING	The name of the Generic Messaging SCF
P_CONNECTIVITY_MANAGER	The name of the Connectivity Manager SCF
P_CHARGING	The name of the Charging SCF
P_ACCOUNT_MANAGEMENT	The name of the Account Management SCF

TpServiceTypeNameList

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

TpAccessType

This data type is identical to a TpString. This identifies the type of access interface requested by the client application. If they request P_OSA_ACCESS, then a reference to the IpAccess interface is returned. (Network operators can define their own access interfaces to satisfy client requirements for different types of access. These can be selected using the TpAccessType, but should be preceded by the string "SP_". The following value is defined :

String Value	Description
P_OSA_ACCESS	Access using the OSA Access Interfaces: IpAccess and IpAppAccess

TpAuthType

This data type is identical to a TpString. It identifies the type of authentication mechanism requested by the client. It provides Network operators and client's with the opportunity to use an alternative to the OSA API Level Authentication interface. This can for example be an implementation specific authentication mechanism, e.g. CORBA Security, or a proprietary Authentication interface supported by the Network Operator. OSA API Level Authentication is the default authentication method. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined :

String Value	Description
P_OSA_AUTHENTICATION	Authenticate using the OSA API Level Authentication Interfaces: IpAPILevelAuthentication and IpAppAPILevelAuthentication
P_AUTHENTICATION	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.

TpFaultStatsRecord

This defines the set of records to be returned giving fault information for the requested time period.

Sequence Element Name	Sequence Element Type
Period	TpTimeInterval
FaultStatsSet~~FaultRecords~~	TpFaultStatsSet

TpFaultStatisticsError

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

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

TpSubjectType

Defines the subject of a query/notification request/result.

Name	Value	Description
P_SUBJECT_UNDEFINED	0	The subject is neither the framework nor the client application
P_SUBJECT_CLIENT_APP	1	The subject is the client application
P_SUBJECT_FW	2	The subject is the framework

TpEncryptionCapability~~TpAuthCapability~~

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

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

TpEncryptionCapabilityList ~~TpAuthCapabilityList~~

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

TpInterfaceName

This data type is identical to a TpString, and is defined as a string of characters that identify the names of the Framework SCFs that are to be supported by the OSA API. Other Network operator specific SCFs may also be used, but should be preceded by the string "SP_". The following values are defined.

Character String Value	Description
P_DISCOVERY	The name for the Discovery interface.
P_EVENT_NOTIFICATION	The name for the Event Notification interface.
P_OAM	The name for the OA&M interface.
P_LOAD_MANAGER	The name for the Load Manager interface.
P_FAULT_MANAGER	The name for the Fault Manager interface.
P_HEARTBEAT_MANAGEMENT	The name for the Heartbeat Management interface.
P_SERVICE_AGREEMENT_MANAGEMENT	The name of the Service Agreement Management interface.
P_REGISTRATION	The name for the Service Registration interface.
P_ENT_OP_ACCOUNT_MANAGEMENT	The name for the Service Subscription: Enterprise Operator Account Management interface.
P_ENT_OP_ACCOUNT_INFO_QUERY	The name for the Service Subscription: Enterprise Operator Account Information Query interface.
P_SVC_CONTRACT_MANAGEMENT	The name for the Service Subscription: Service Contract Management interface.
P_SVC_CONTRACT_INFO_QUERY	The name for the Service Subscription: Service Contract Information Query interface.
P_CLIENT_APP_MANAGEMENT	The name for the Service Subscription: Client Application Management interface.
P_CLIENT_APP_INFO_QUERY	The name for the Service Subscription: Client Application Information Query interface.
P_SVC_PROFILE_MANAGEMENT	The name for the Service Subscription: Service Profile Management interface.
P_SVC_PROFILE_INFO_QUERY	The name for the Service Subscription: Service Profile Information Query interface.

TpSignatureAndServiceMgr

This is a Sequence of Data Elements containing the digital signature of the Framework for the service agreement, and a reference to the SCF manager interface of the SCF.

Sequence Element Name	Sequence Element Type
DigitalSignature	~~TpString~~TpOctetSet
ServiceMgrInterface	IpServiceRef

TpInterfaceNameList

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

TpServiceContractIDList

This data type defines a `Numbered List of Data Elements` of type TpServiceContractID.