ETSI TS 129 198-3 V4.0.0 (2001-03)

Technical Specification

**Universal Mobile Telecommunications System (UMTS);
Open Service Access (OSA);
Application Programming Interface (API);
Part 3: Framework
(3GPP TS 29.198-3 version 4.0.0 Release 4)**

Reference

RTS/TSGN-0529198-3Uv4

Keywords

UMTS

ETSI

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

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

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

Intellectual Property Rights

Foreword

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

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

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under www.etsi.org/key .

Contents

Foreword

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

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

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

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

Part 1:	Overview	
Part 2:	Common Data Definitions	
Part 3:	**Framework**	
Part 4:	Call Control SCF	
Part 5:	User Interaction SCF	
Part 6:	Mobility SCF	
Part 7:	Terminal Capabilities SCF	
Part 8:	Data Session Control SCF	
Part 9:	Generic Messaging SCF	(not part of 3GPP Release 4)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 4)
Part 11:	Account Management SCF	
Part 12:	Charging SCF	

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

OSA API specifications 29.198-family		OSA API Mapping - 29.998-family	
29.198-1	**Part 1: Overview**	**29.998-1**	**Part 1: Overview**
29.198-2	**Part 2: Common Data Definitions**	29.998-2	Not Applicable
29.198-3	**Part 3: Framework**	29.998-3	Not Applicable
29.198-4	**Part 4: Call Control SCF**	**29.998-4-1**	**Subpart 1: Generic Call Control – CAP mapping**
		29.998-4-2	
29.198-5	**Part 5: User Interaction SCF**	**29.998-5-1**	**Subpart 1: User Interaction – CAP mapping**
		29.998-5-2	
		29.998-5-3	
		29.998-5-4	**Subpart 4: User Interaction – SMS mapping**
29.198-6	**Part 6: Mobility SCF**	**29.998-6**	**User Status and User Location – MAP mapping**
29.198-7	**Part 7: Terminal Capabilities SCF**	29.998-7	Not Applicable
29.198-8	**Part 8: Data Session Control SCF**	**29.998-8**	**Data Session Control – CAP mapping**
29.198-9	Part 9: Generic Messaging SCF	29.998-9	Not Applicable
29.198-10	Part 10: Connectivity Manager SCF	29.998-10	Not Applicable
29.198-11	**Part 11: Account Management SCF**	29.998-11	Not Applicable
29.198-12	**Part 12: Charging SCF**	29.998-12	Not Applicable

1 Scope

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

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

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

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

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

This specification has been defined jointly between 3GPP TSG CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with the JAIN consortium.

2 References

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

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

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

[2] 3GPP TS 22.127: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 4)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 4)".

[4] IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August1996].

3 Definitions, symbols and abbreviations

3.1 Definitions

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

3.2 Abbreviations

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

4 Overview of the Framework

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

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

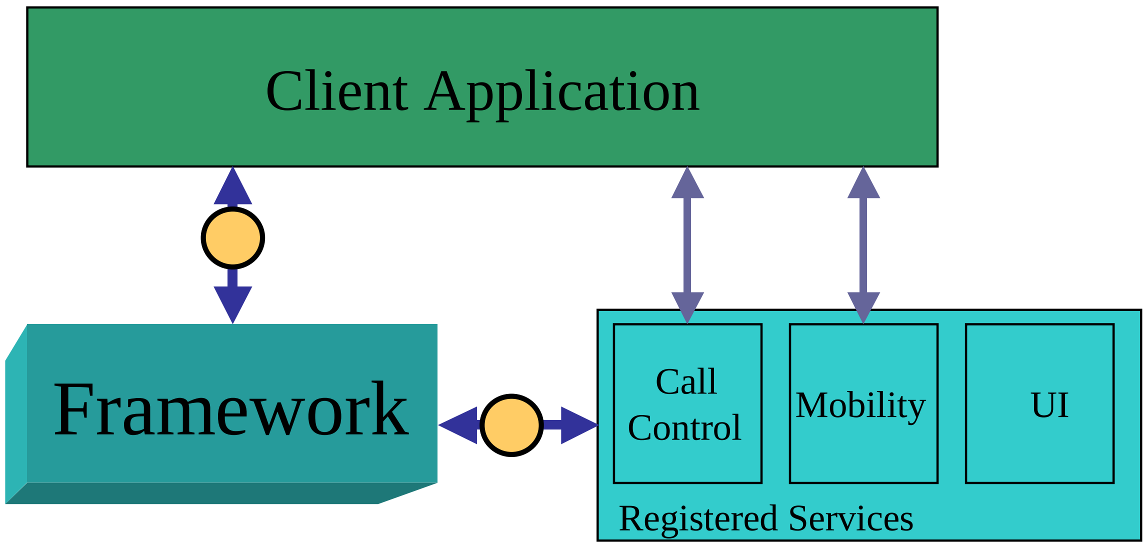


Figure:

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

Basic mechanisms between Application and Framework:

- **Authentication:** Once an off-line service agreement exists, the application can access the authentication interface. The authentication model of OSA is a peer-to-peer model. The application shall authenticate the Framework and vice versa. The application shall be authenticated before it is allowed to use any other OSA interface.
- **Authorisation:** Authorisation is distinguished from authentication in that authorisation is the action of determining what a previously authenticated application is allowed to do. Authentication shall precede authorisation. Once authenticated, an application is authorised to access certain SCFs.

- **Discovery of Framework and network SCFs:** After successful authentication, applications can obtain available Framework interfaces and use the discovery interface to obtain information on authorised network SCFs. The Discovery interface can be used at any time after successful authentication.
- **Establishment of service agreement:** Before any application can interact with a network SCF, a service agreement shall be established. A service agreement may consist of an off-line (e.g. by physically exchanging documents) and an on-line part. The application has to sign the on-line part of the service agreement before it is allowed to access any network SCF.
- **Access to network SCFs:** The Framework shall provide access control functions to authorise the access to SCFs or service data for any API method from an application, with the specified security level, context, domain, etc.

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

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

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

- The *sequence diagrams* give the reader a practical idea of how each of the Framework is implemented.
- The *class diagrams* clause shows how each of the interfaces applicable to the Framework relate to one another.
- The *interface specification* clause describes in detail each of the interfaces shown within the class diagram part.
- The *State Transition Diagrams (STD)* show the progression of internal processes, either in the application or in the gateway.
- The *data definitions* clause shows a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the common data types part of the present document (29.198-2).

5 The Base Interface Specification

5.1 Interface Specification Format

This clause 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 call-back 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 shall implement the relevant `IpApp<name> or IpSvc<name>` interfaces to provide the call-back mechanism.

5.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that shall 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.

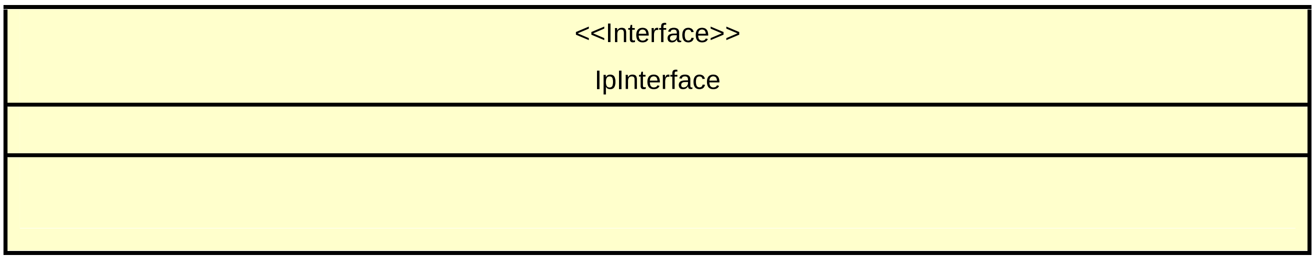
5.1.4 State Model

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.



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.

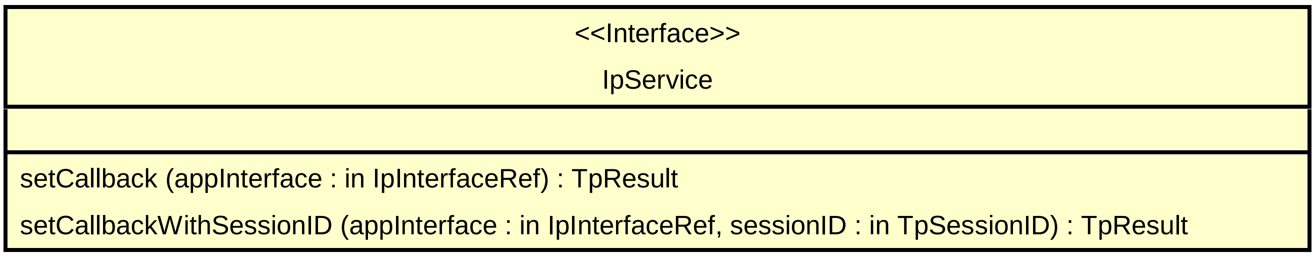
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that shall be implemented by the application (e.g. for API call-backs) are denoted as 'Application Interface'.

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) : TpResult
setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : TpResult

Method
`setCallback()`

This method specifies the reference address of the call-back interface that a service uses to invoke methods on the application.

Parameters

`appInterface : in IpInterfaceRef`

Specifies a reference to the application interface, which is used for call-backs

Raises

`TpGeneralException`

Method
`setCallbackWithSessionID()`

This method specifies the reference address of the application's call-back interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg.

Parameters

`appInterface : in IpInterfaceRef`

Specifies a reference to the application interface, which is used for call-backs.

`sessionID : in TpSessionID`

Specifies the session for which the service can invoke the application's call-back interface.

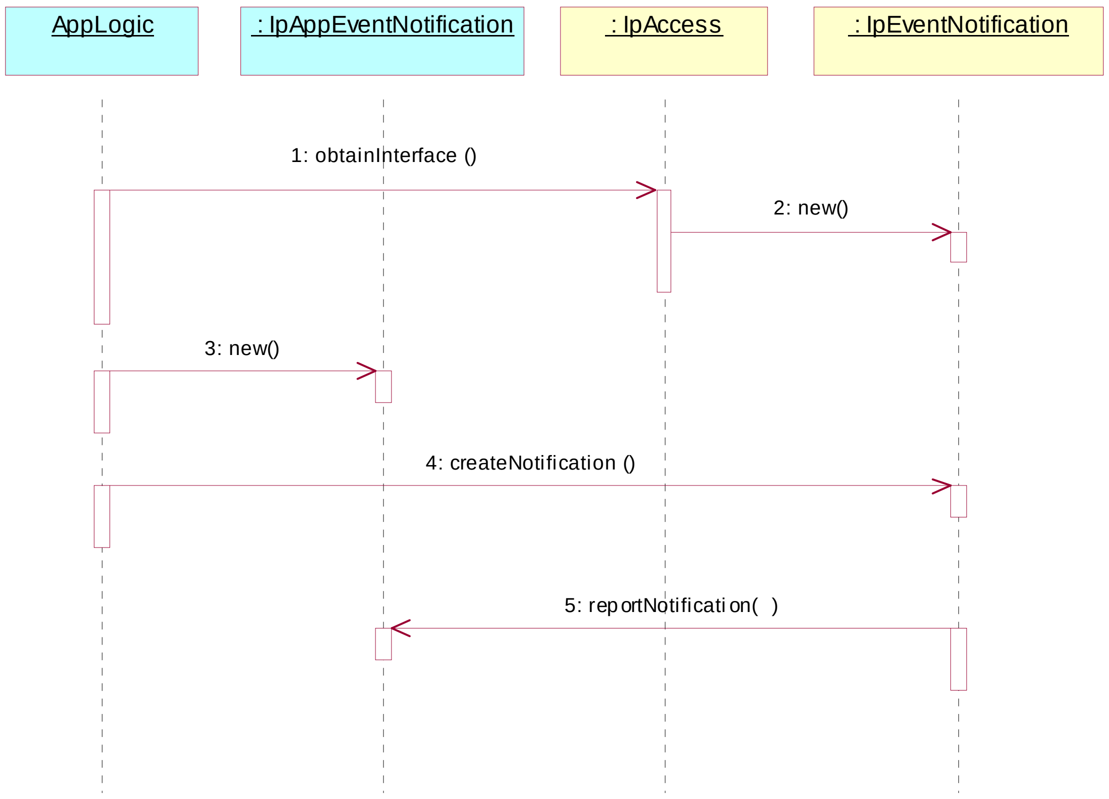
Raises

`TpGeneralException`

6 Framework-to-Application Sequence Diagrams

6.1 Event Notification Sequence Diagrams

6.1.1 Enable Event Notification



1: This message is used to receive a reference to the object implementing the IpEventNotification interface.

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

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

4: enableNotification(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;

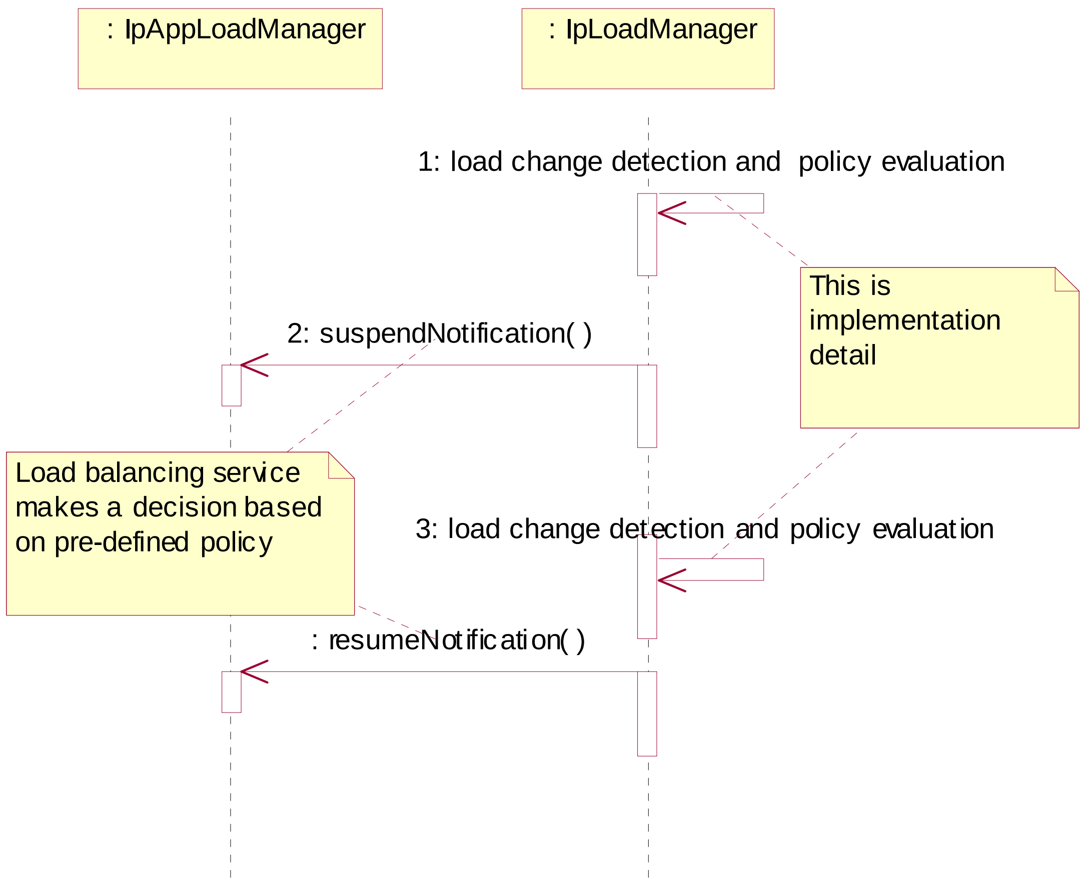
- in the case of using the event notification mechanism, the application needs not take the initiative to ask about the availability of SCFs, but it is only informed of the ones that are newly available.

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

6.2 Integrity Management Sequence Diagrams

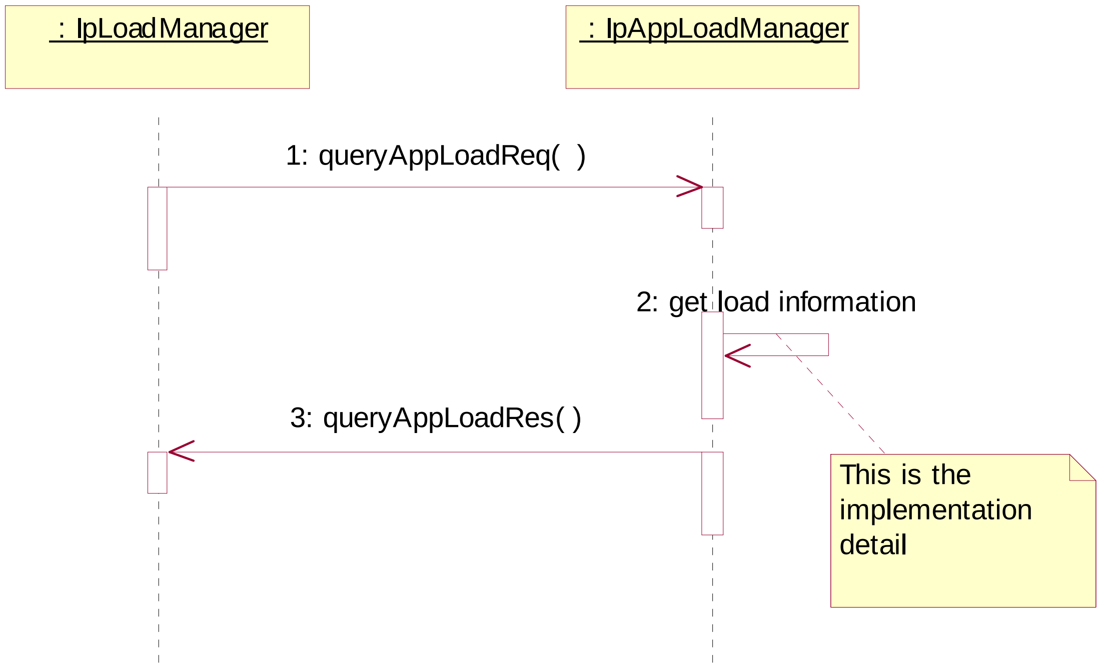
6.2.1 Load Management: Suspend/Resume notification from application

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


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

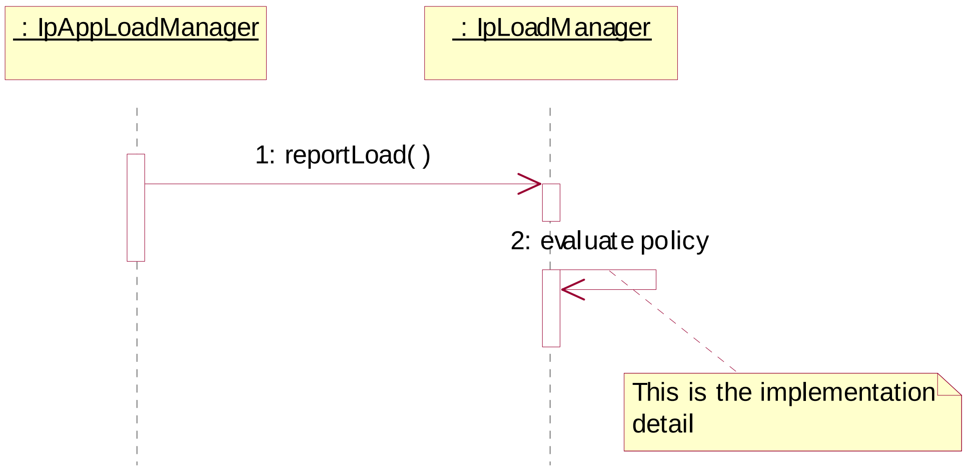
6.2.2 Load Management: Framework queries load status

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



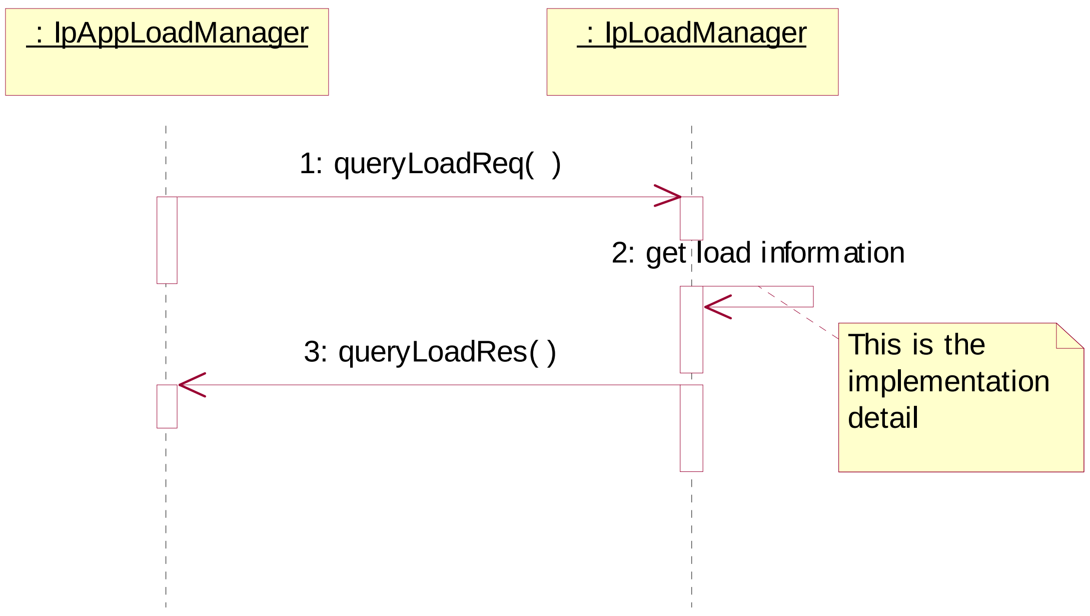
6.2.3 Load Management: Application reports current load condition

This sequence diagram shows how an application reports its load condition to the Framework load manager.



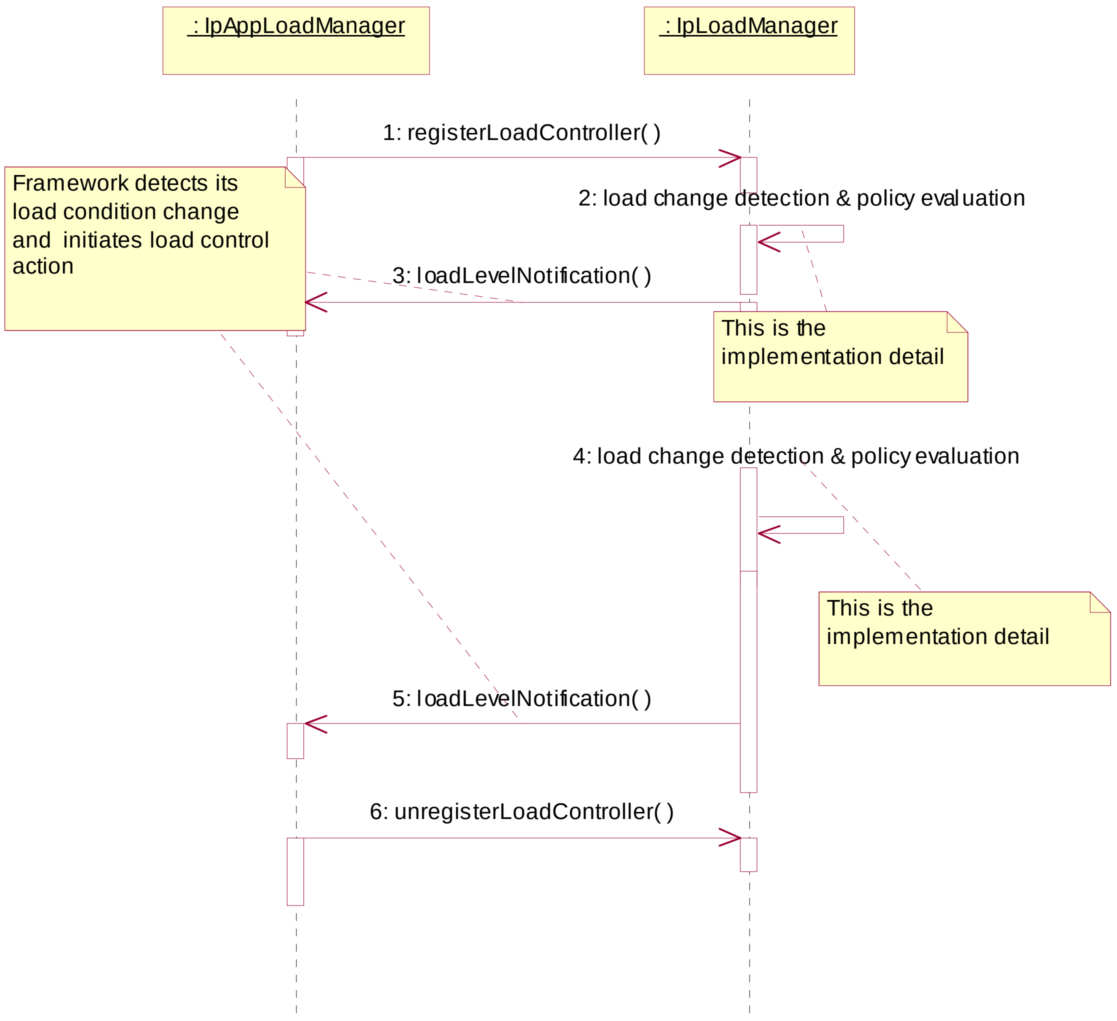
6.2.4 Load Management: Application queries load status

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

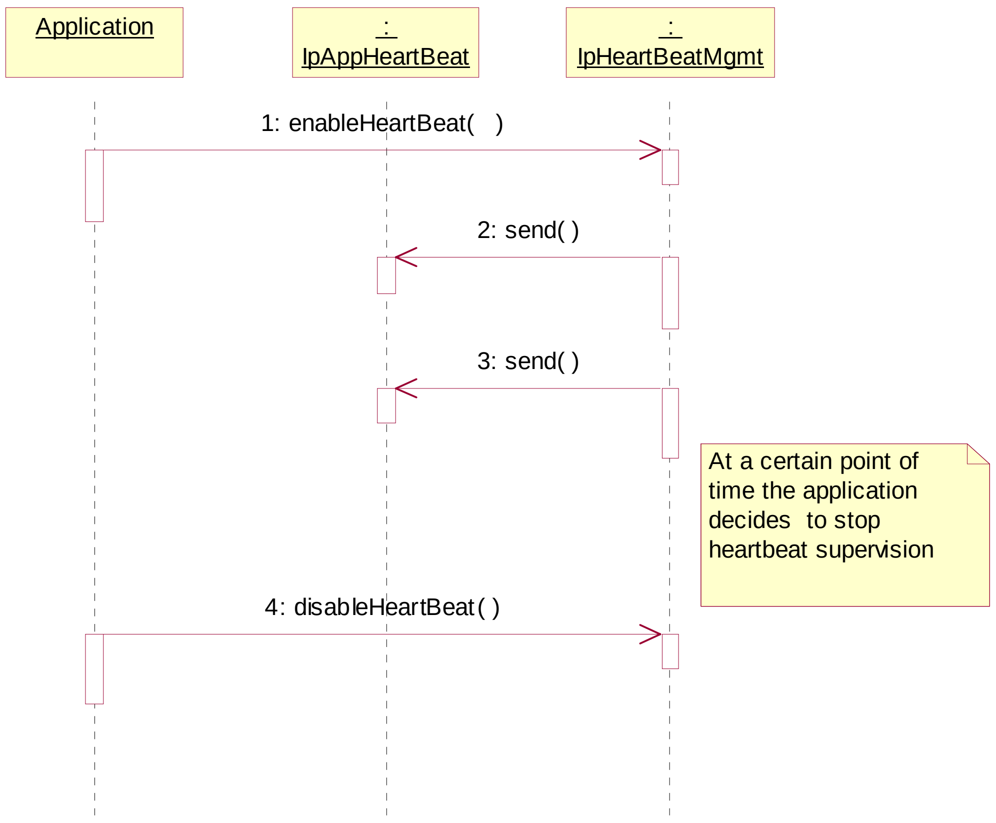


6.2.5 Load Management: Application call-back registration and load control

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

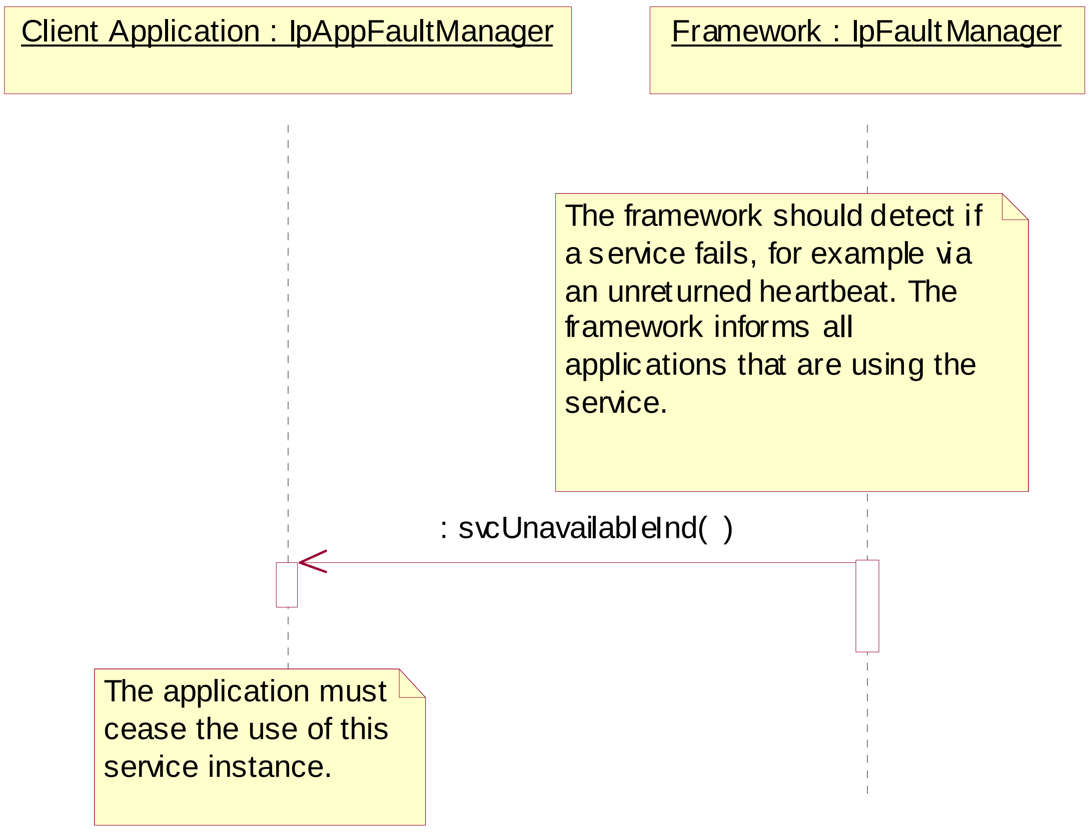

: IpAppLoadManager
: IpLoadManager
1: registerLoadController( )
2: load change detection & policy evaluation
3: loadLevelNotification( )
Framework detects its load condition change and initiates load control action
This is the implementation detail
4: load change detection & policy evaluation
This is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

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



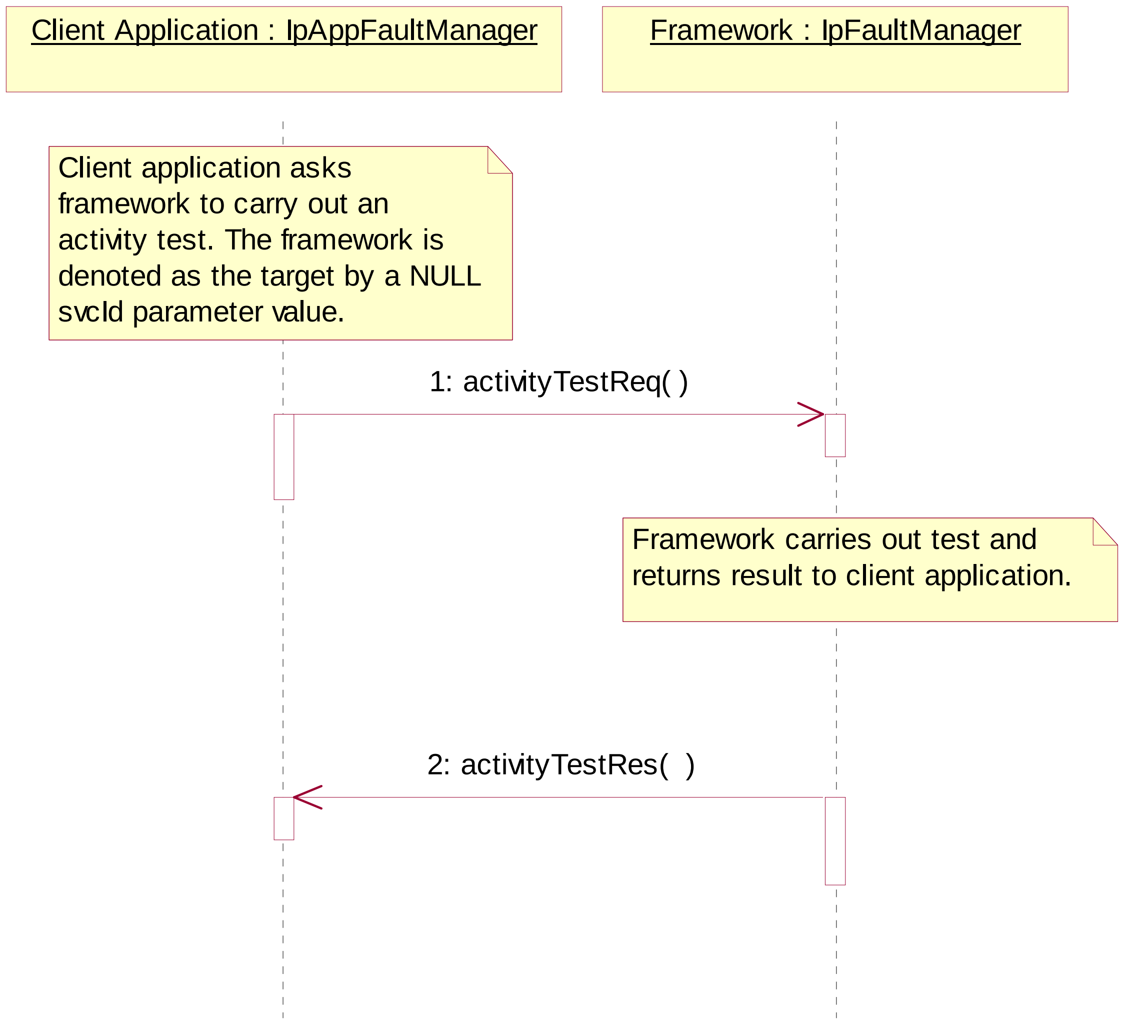
6.2.7 Fault Management: Framework detects a Service failure

The Framework has detected that the service has failed (probably by the use of the heartbeat mechanism). The Framework updates its own records and informs any client applications that are using the service to stop.


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

1: The Framework informs each 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.

6.2.8 Fault Management: Application requests a Framework activity test


Client Application : IpAppFaultManager
Framework : IpFaultManager
Client application asks framework to carry out an activity test. The framework is denoted as the target by a NULL svcId parameter value.
1: activityTestReq( )
Framework carries out test and returns result to client application.
2: activityTestRes( )

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.

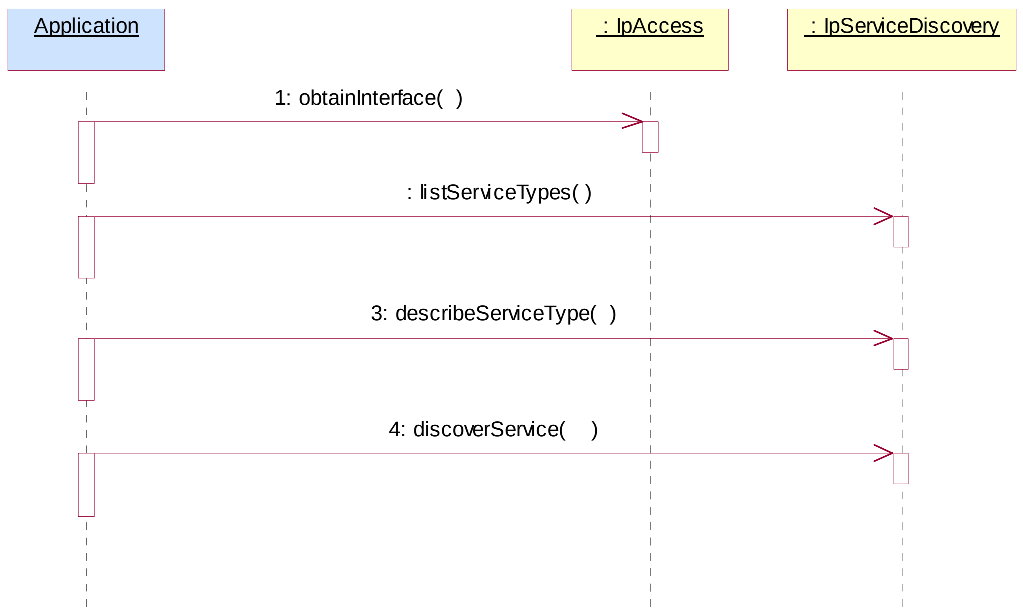
6.3 Service Discovery Sequence Diagrams

6.3.1 Service Discovery

The following figure shows how Applications discover a new SCF 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 is a three-step process:



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.

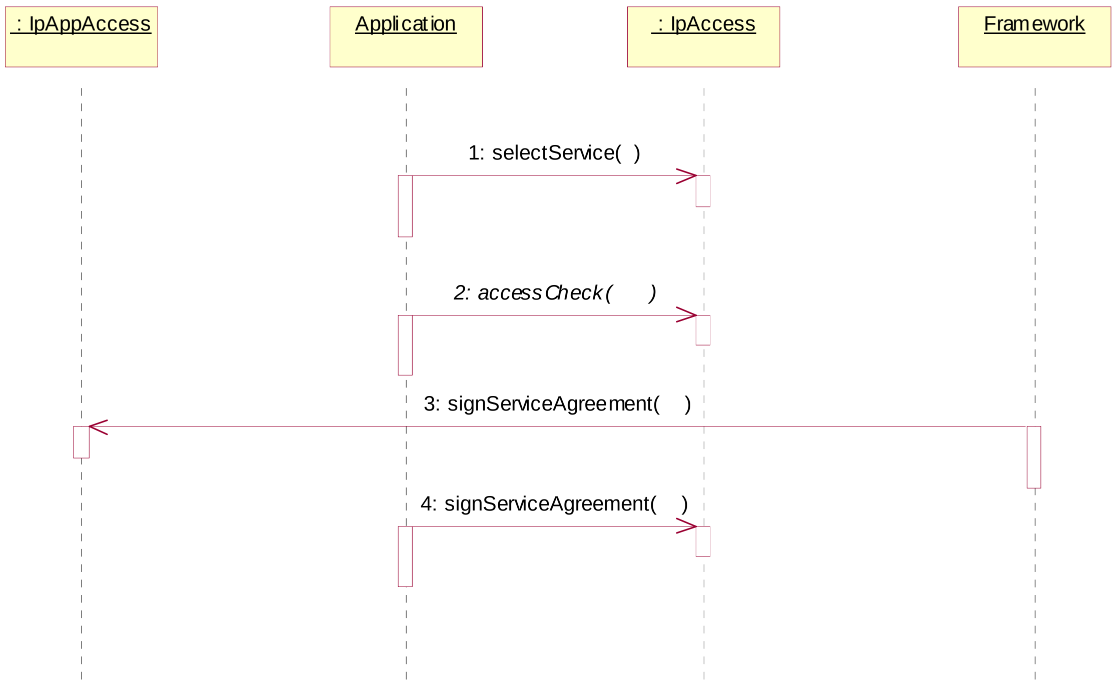
6.4 Trust and Security Management Sequence Diagrams

6.4.1 Service Selection

The following figure shows the process of selecting an SCF.

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

This is achieved by the following two steps:



1: Service Selection: first step - selectService

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

Input is:

· in serviceID

This identifies the SCF required.

And output:

· out serviceToken

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

3: 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 getServiceManager operation on the service factory the Framework retrieves this interface and returns it to the Application. The service properties suitable for this application are also fed to the SCF (via the service factory 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

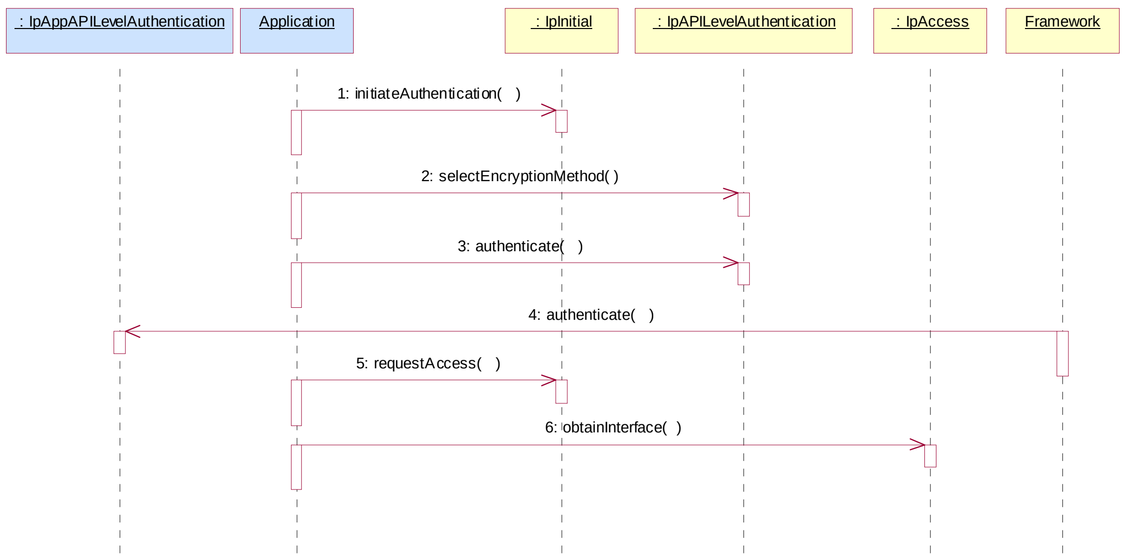
This is a reference to a structure containing the digital signature of the Framework for the service agreement, and a reference to the manager interface of the SCF.

6.4.2 Initial Access

The following figure shows an application accessing the OSA Framework for the first time.

Before being authorized to use the OSA SCFs, the Application shall first of all authenticate itself with the Framework. For this purpose the application 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 Application has no guarantee that this is a Framework interface reference, but it to initiate the authentication process with the Framework. The Initial Contact interface only supports the initiateAuthentication method to allow the authentication process to take place.

Once the Application 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 application requests a certain type of access SCF.



1: Initiate Authentication

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

2: Select Encryption Method

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

3: Authenticate

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

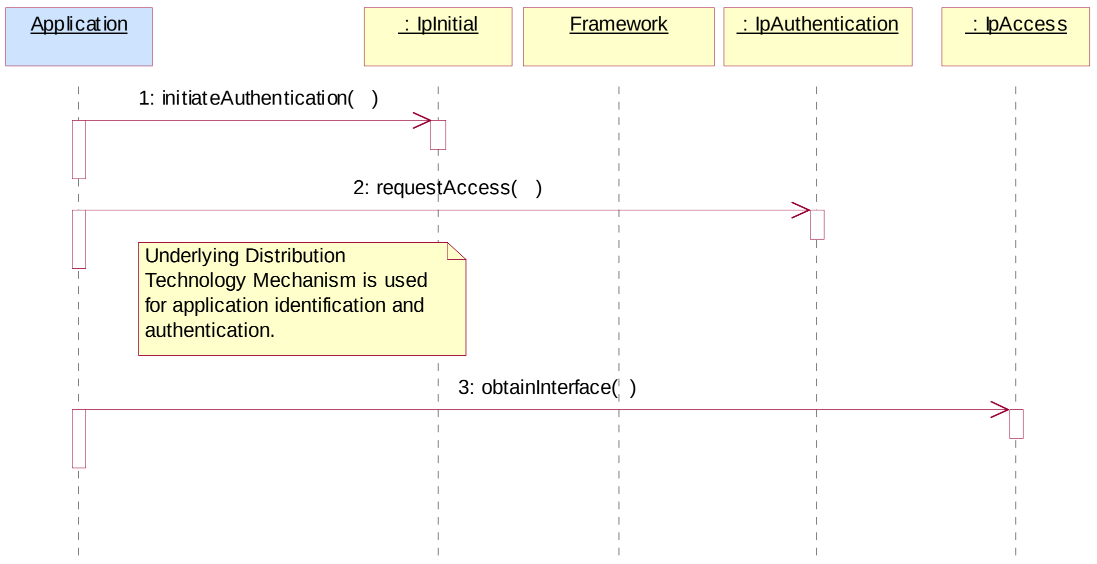
5: Request Access

Upon successful (mutual) authentication, the Application 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: The client invokes obtainInterface on the Framework's Access interface to obtain a reference to its service discovery interface.

6.4.3 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.


Application
: IpInitial
Framework
: IpAuthentication
: IpAccess
1: initiateAuthentication( )
2: requestAccess( )
Underlying Distribution Technology Mechanism is used for application identification and authentication.
3: obtainInterface( )

1: The application calls initiateAuthentication on the OSA Framework Initial interface. This allows the application to specify the type of authentication process. In this case, the application selects to use the underlying distribution technology mechanism for identification and authentication.

2: The application 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 application.

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

6.4.4 API Level Authentication

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

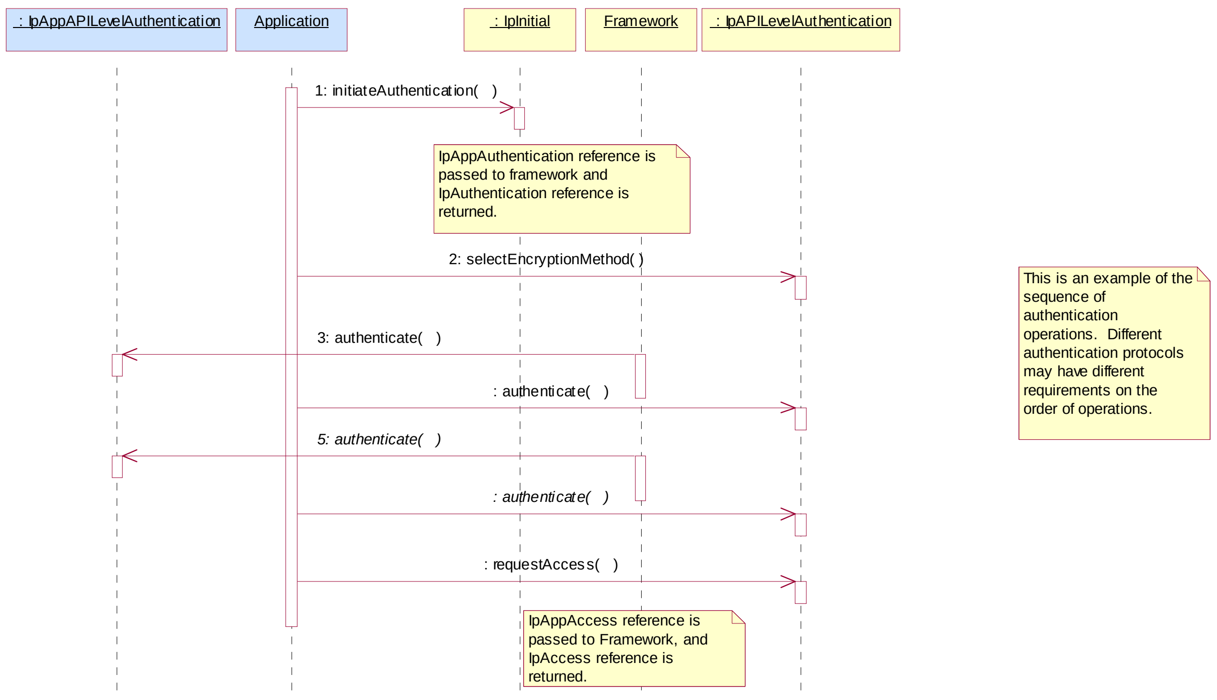
The OSA API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity.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

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

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

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

3) The application and Framework interact to authenticate each other. Depending on the method prescribed, this procedure may consist of a number of messages e.g. a challenge/ response protocol. This authentication protocol is performed using the authenticate method on the API Level Authentication interface. Depending on the authentication method selected, the protocol may require invocations on the API Level Authentication interface supported by the Framework; or on the application counterpart; or on both.


: IpAppAPILevelAuthentication
Application
: IpInitial
Framework
: IpAPILevelAuthentication
1: initiateAuthentication( )
IpAppAuthentication reference is passed to framework and IpAuthentication reference is returned.
2: selectEncryptionMethod( )
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
3: authenticate( )
: authenticate( )
5: authenticate( )
: authenticate( )
: requestAccess( )
IpAppAccess reference is passed to Framework, and IpAccess reference is returned.

Framework-to-Application Class Diagrams

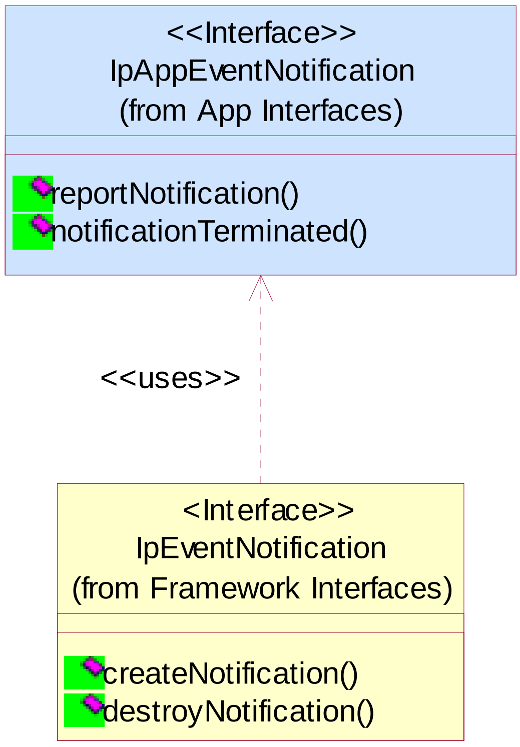


Figure: Event Notification Class Diagram

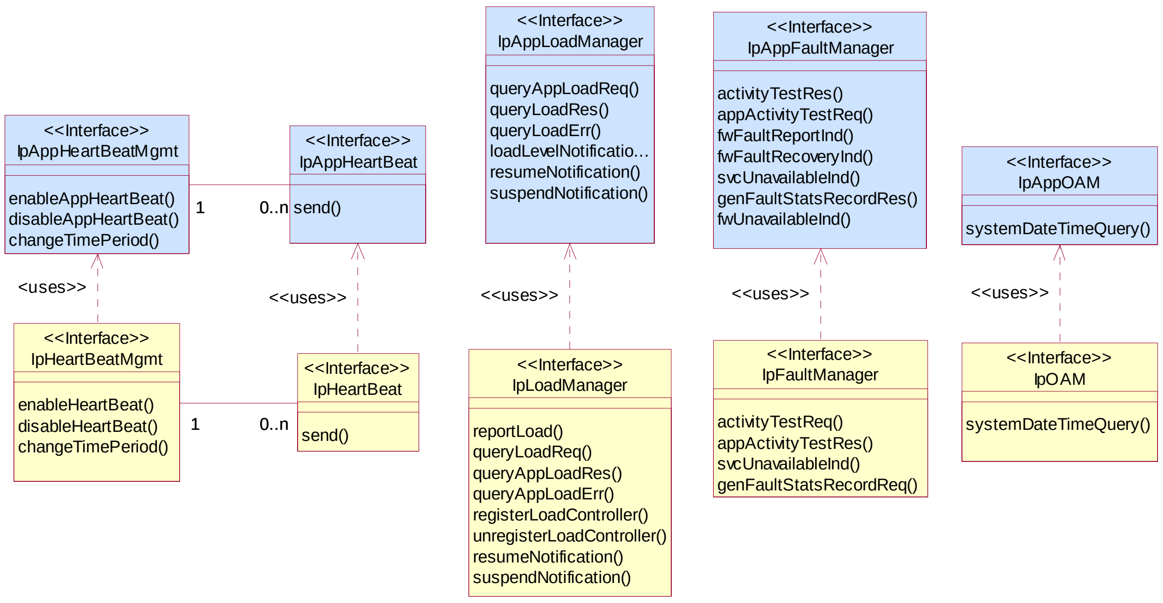


Figure: Integrity Management Package Overview

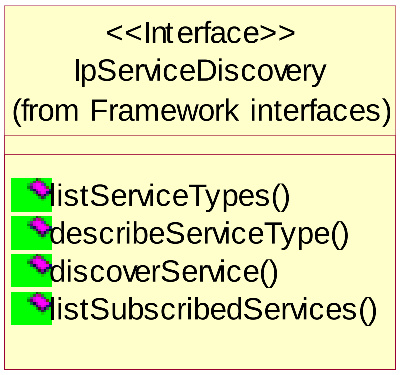


Figure: Service Discovery Package Overview

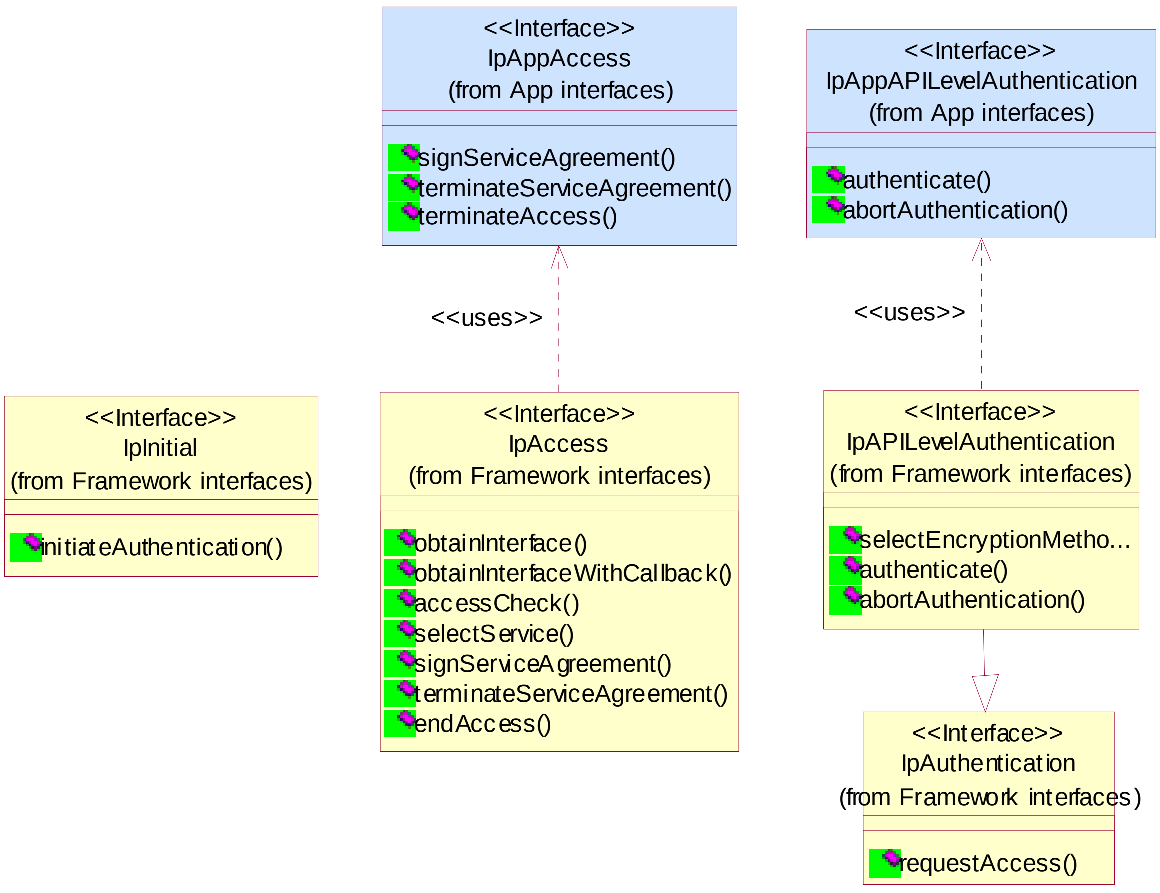


Figure: Trust and Security Management Package Overview

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:

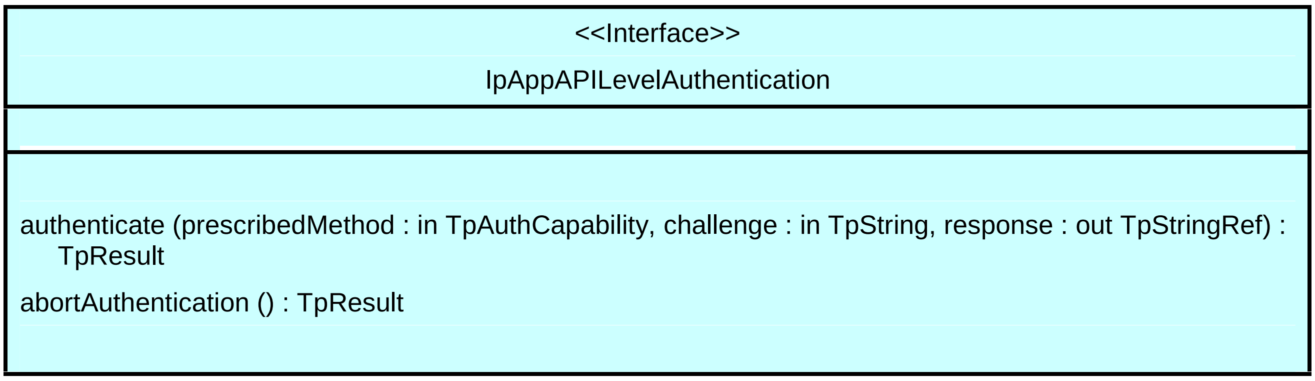
1) Initial Contact with the Framework;

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, response : out TpStringRef) : TpResult

abortAuthentication () : TpResult

Method
`authenticate()`

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

Parameters

`prescribedMethod : in TpAuthCapability`

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

`challenge : in 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 [4]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

`response : out TpStringRef`

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

Raises

`TpGeneralException,TpFWException`

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, (e.g. if the client application responds incorrectly to a challenge.) 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

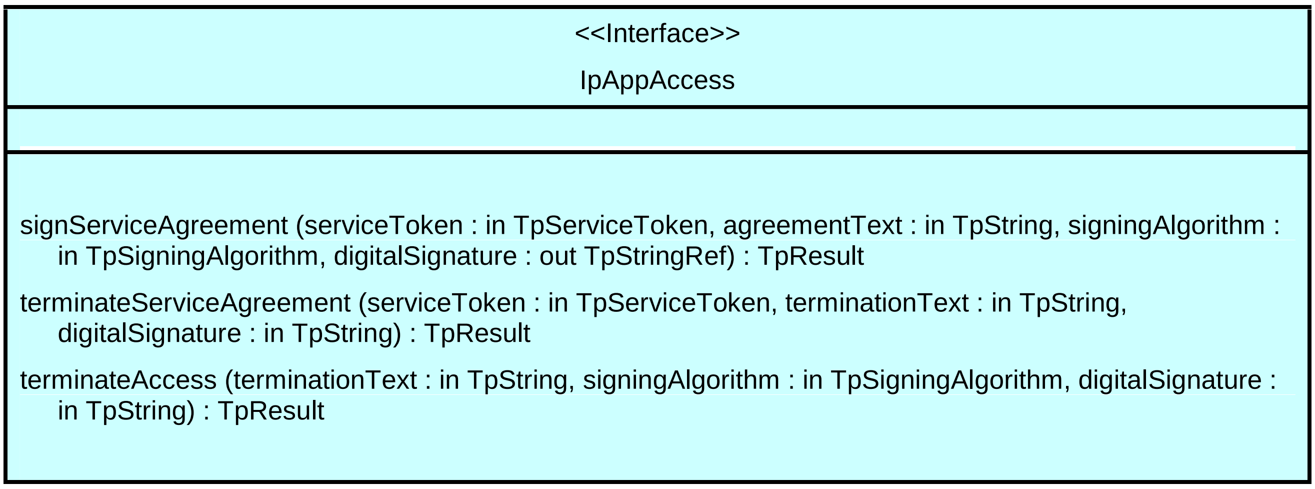
Raises

`TpGeneralException,TpFWException`

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, digitalSignature : out TpStringRef) : TpResult

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

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

Method

`signServiceAgreement()`

This method is used by the Framework to request that the client application sign an agreement on the service. It is called in response to the client application calling the selectService() method on the IpAccess interface of the Framework. The Framework provides the service agreement text for the client application to sign. If the client application agrees, it signs the service agreement, returning its digital signature to the Framework.

Parameters

`serviceToken : in TpServiceToken`

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

`agreementText : in TpString`

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

signingAlgorithm : in TpSigningAlgorithm

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

digitalSignature : out TpStringRef

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

Raises

TpGeneralException,TpFWException

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.

Raises

TpGeneralException,TpFWException

Method

terminateAccess()

The terminateAccess operation is used to end the client application's access session with the Framework. The Framework is terminating the client application's access session. (For example, this may be done if the Framework believes the client application is masquerading as someone else. Using this operation will force the client application to re-authenticate if it wishes to continue using the Framework's services.)

After terminateAccess() is invoked, the client application will not 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

`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.

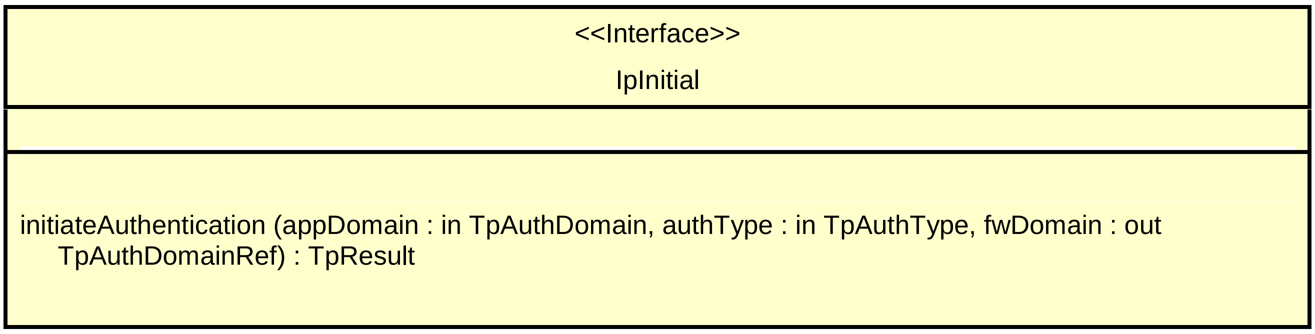
Raises

`TpGeneralException,TpFWException`

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, fwDomain : out TpAuthDomainRef) : TpResult

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.

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 references to interfaces of type Ip(App)Authentication which is used when an underlying distribution technology authentication mechanism is used.

`fwDomain : out TpAuthDomainRef`

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.

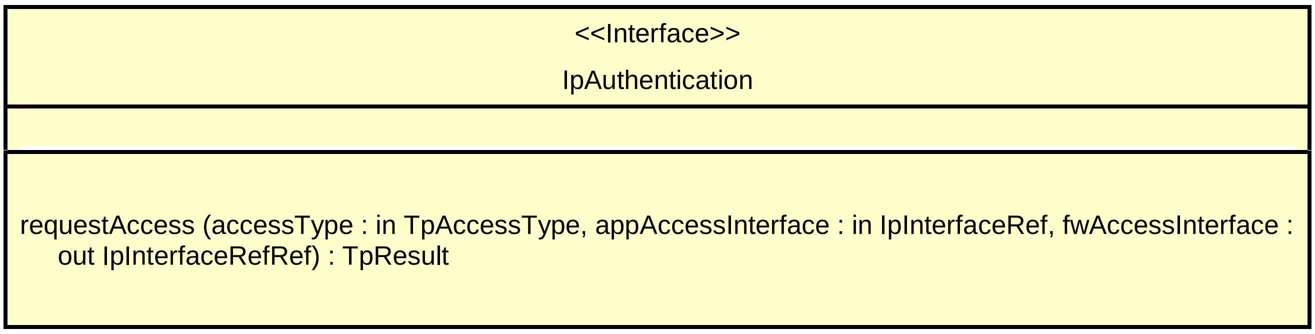
Raises

`TpGeneralException,TpFWException`

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, fwAccessInterface : out IpInterfaceRefRef) : TpResult

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.

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).

`fwAccessInterface : out IpInterfaceRefRef`

This provides the reference for the client application to call the access interface of the Framework.

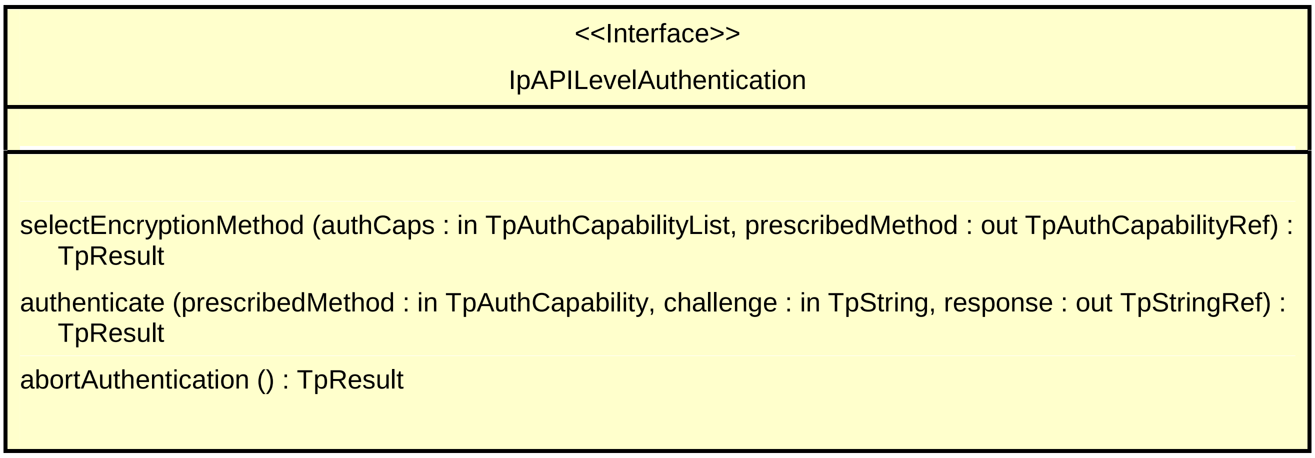
Raises

`TpGeneralException,TpFWException`

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, prescribedMethod : out TpAuthCapabilityRef) : TpResult

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

abortAuthentication () : TpResult

Method
`selectEncryptionMethod()`

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

Parameters

`authCaps : in TpAuthCapabilityList`

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

`prescribedMethod : out TpAuthCapabilityRef`

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

Raises

`TpGeneralException,TpFWException`

Method

`authenticate()`

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

Parameters

`prescribedMethod : in TpAuthCapability`

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

`challenge : in 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 [4]. The challenge will be encrypted with the mechanism prescribed by selectEncryptionMethod().

`response : out TpStringRef`

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

Raises

`TpGeneralException,TpFWException`

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 (e.g. if the Framework responds incorrectly to a challenge). 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

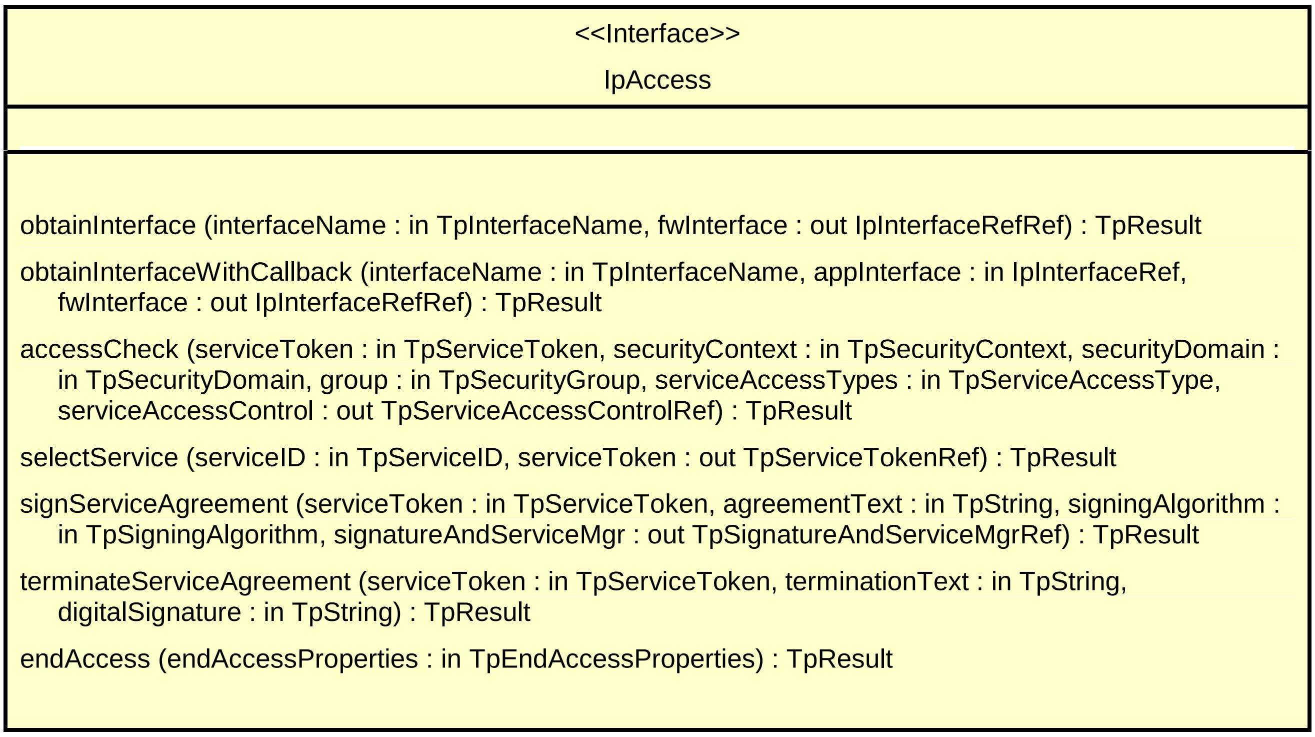
No Parameters were identified for this method.

Raises

`TpGeneralException,TpFWException`

8.1.6 Interface Class IpAccess

Inherits from: IpInterface.



<<Interface>>

IpAccess

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

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

accessCheck (serviceToken : in TpServiceToken, securityContext : in TpSecurityContext, securityDomain : in TpSecurityDomain, group : in TpSecurityGroup, serviceAccessTypes : in TpServiceAccessType, serviceAccessControl : out TpServiceAccessControlRef) : TpResult

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

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

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

endAccess (endAccessProperties : in TpEndAccessProperties) : TpResult

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 call-back interface to the Framework.)

Parameters

`interfaceName : in TpInterfaceName`

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

`fwInterface : out IpInterfaceRefRef`

This is the reference to the interface requested.

Raises

`TpGeneralException,TpFWException`

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 call-back interface to the Framework. (The obtainInterface method should be used when no call-back interface needs to be supplied.)

Parameters

interfaceName : in TpInterfaceName

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

appInterface : in IpInterfaceRef

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

fwInterface : out IpInterfaceRefRef

This is the reference to the interface requested.

Raises

TpGeneralException,TpFWException

Method
accessCheck()

This method may be used by the client application to check if it is authorised to access the specified service. The response is used to indicate whether the request for access has been granted or denied and if granted the level of trust that will be applied. The securityModelID and the relevant securityLevel are defined as part of the registration data for the service, and the service agreement. They are specific to the service.

securityModelID:

The identity of the specific Security Model that is to be used to define a set of appropriate policies for the service that can be used by the Framework to determine access rights. The model may include blanket permission, session permission or one shot permission. A number of security models will be stored by the Framework, and referenced by the access control module, according to the security model identifier of the service.

securityLevel:

The trust level required by the service for granting access. The Security Level is used by the Framework's access control module when it checks for access rights.

Parameters

serviceToken : in TpServiceToken

The serviceToken identifies the specific service that the client application wishes to access. The service Token identifies the service type and service properties selected by the client application when it invoked selectService().

securityContext : in TpSecurityContext

A context is a group of security relevant attributes that may have an influence on the result of the accessCheck request.

`securityDomain : in TpSecurityDomain`

The security domain in which the client application is operating may influence the access control decisions and the specific set of features that the requestor is entitled to use.

`group : in TpSecurityGroup`

A group can be used to define the access rights associated with all client applications that belong to that group. This simplifies the administration of access rights.

`serviceAccessTypes : in TpServiceAccessType`

These are defined by the specific Security Model in use but are expected to include: Create, Read, Update, Delete as well as those specific to services.

`serviceAccessControl : out TpServiceAccessControlRef`

This contains the access control policy information that controls access to the service feature, and the trustLevel that the service provider has assigned to the client application.

```
structure TpServiceAccessControl {
                policy:     TpString;
                trustLevel:     TpString;
                };
```

The policy parameter indicates whether access has been granted or denied. If granted then the parameter trustLevel shall also have a value.

The trustLevel parameter indicates the trust level that the service provider has assigned to the client application.

Raises

`TpGeneralException,TpFWException`

Method

`selectService()`

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

Parameters

`serviceID : in TpServiceID`

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

`serviceToken : out TpServiceTokenRef`

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

Raises

`TpGeneralException,TpFWException`

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. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned.

Parameters

`serviceToken : in TpServiceToken`

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

`agreementText : in TpString`

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

`signingAlgorithm : in TpSigningAlgorithm`

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

`signatureAndServiceMgr : out TpSignatureAndServiceMgrRef`

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

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

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

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

Raises

`TpGeneralException,TpFWException`

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

TpGeneralException,TpFWException

Method
endAccess()

The endAccess operation is used to end the client application's access session with the Framework. The client application requests that its access session 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

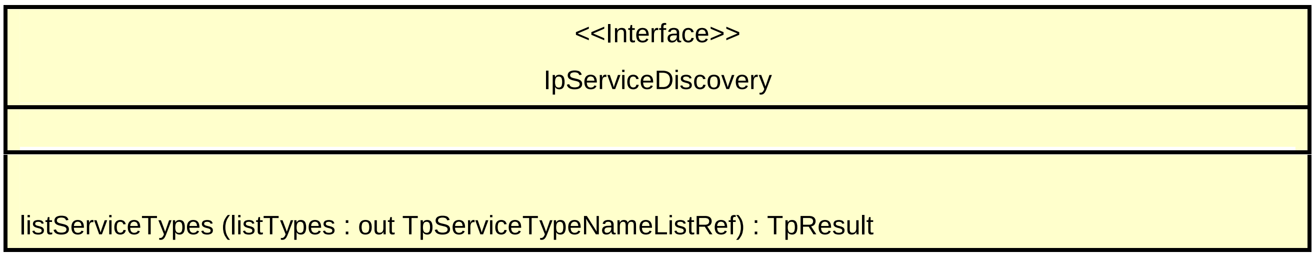
TpGeneralException,TpFWException

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) shall 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 (listTypes : out TpServiceTypeNameListRef) : TpResult describeServiceType (name : in TpServiceTypeName, serviceTypeDescription : out TpServiceTypeDescriptionRef) : TpResult

discoverService (serviceTypeName : in TpServiceTypeName, desiredPropertyList : in TpServicePropertyList, max : in TpInt32, serviceList : out TpServiceListRef) : TpResult

listSubscribedServices (serviceList : out TpServiceListRef) : TpResult

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.

Parameters

`listTypes : out TpServiceTypeNameListRef`

The names of the requested service types.

Raises

`TpGeneralException,TpFWException`

Method
`describeServiceType()`

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

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.

`serviceTypeDescription : out TpServiceTypeDescriptionRef`

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.

Raises

`TpGeneralException,TpFWException`

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 shall 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.

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 shall 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.

serviceList : out TpServiceListRef

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.

Raises

TpGeneralException,TpFWException

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.

Parameters

serviceList : out TpServiceListRef

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.

Raises

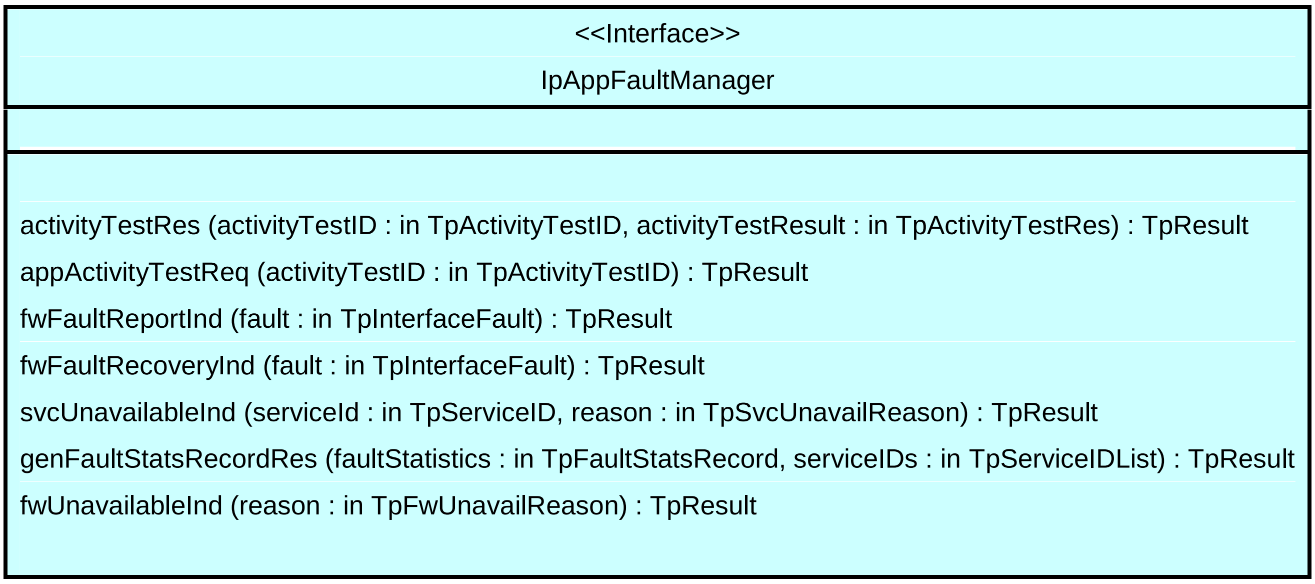
`TpGeneralException,TpFWException`

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) : TpResult
appActivityTestReq (activityTestID : in TpActivityTestID) : TpResult
fwFaultReportInd (fault : in TpInterfaceFault) : TpResult
fwFaultRecoveryInd (fault : in TpInterfaceFault) : TpResult
svcUnavailableInd (serviceId : in TpServiceID, reason : in TpSvcUnavailReason) : TpResult
genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord, serviceIDs : in TpServiceIDList) : TpResult
fwUnavailableInd (reason : in TpFwUnavailReason) : TpResult

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.

Raises

`TpGeneralException,TpFWException`

Method
`appActivityTestReq()`

The Framework invokes this method to test that the client application is operational. On receipt of this request, the application shall 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.

Raises

`TpGeneralException,TpFWException`

Method
`fwFaultReportInd()`

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

`TpGeneralException,TpFWException`

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.

Raises

`TpGeneralException,TpFWException`

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 shall 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

Raises

TpGeneralException,TpFWException

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.

Raises

TpGeneralException,TpFWException

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 call-back interfaces as it is assumed that the client application supplies its Fault Management call-back 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) : TpResult appActivityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : TpResult svcUnavailableInd (serviceID : in TpServiceID) : TpResult genFaultStatsRecordReq (timePeriod : in TpTimeInterval, serviceIDs : in TpServiceIDList) : TpResult

Method
activityTestReq()

The application invokes this method to test that the Framework or a service is operational. On receipt of this request, the Framework shall 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

TpGeneralException,TpFWException

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

TpGeneralException,TpFWException

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

TpGeneralException,TpFWException

Method

genFaultStatsRecordReq()

This method is used by the application to solicit fault statistics from the Framework. On receipt of this request the Framework shall 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

TpGeneralException,TpFWException

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) : TpResult disableAppHeartBeat (session : in TpSessionID) : TpResult changeTimePeriod (duration : in TpDuration, session : in TpSessionID) : TpResult

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 call-back interface the heartbeat is calling.

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpGeneralException,TpFWException

Method
disableAppHeartBeat()

Allows the stop of the heartbeat supervision of the application.

Parameters

session : in TpSessionID

Identifies the heartbeat session.

Raises

TpGeneralException,TpFWException

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.

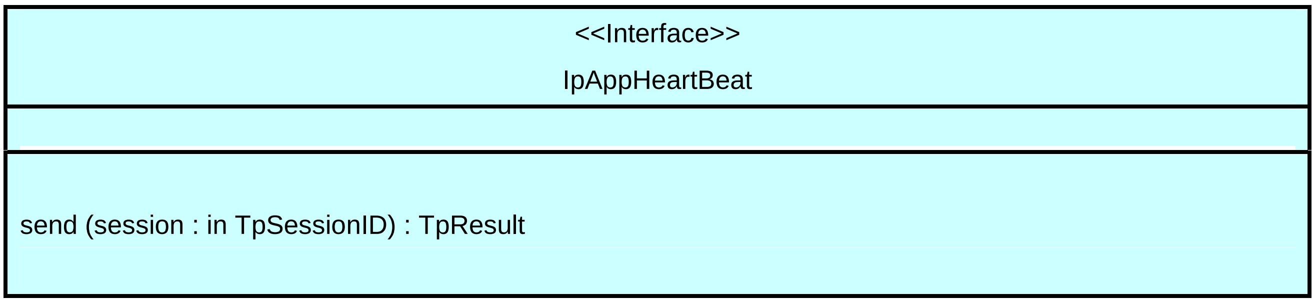
Raises

`TpGeneralException,TpFWException`

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.



Method
`send()`

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

Parameters

`session : in TpSessionID`

Identifies the heartbeat session.

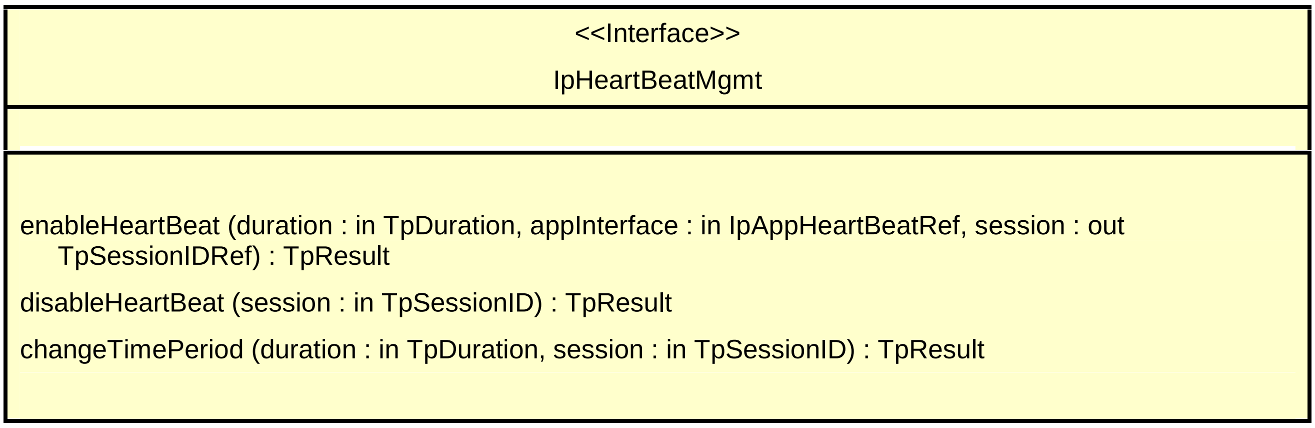
Raises

`TpGeneralException,TpFWException`

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, session : out TpSessionIDRef) : TpResult
disableHeartBeat (session : in TpSessionID) : TpResult
changeTimePeriod (duration : in TpDuration, session : in TpSessionID) : TpResult

Method
`enableHeartBeat()`

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

Parameters

`duration : in TpDuration`

The duration in milliseconds between the heartbeats.

`appInterface : in IpAppHeartBeatRef`

This parameter refers to the call-back interface the heartbeat is calling.

`session : out TpSessionIDRef`

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

Raises

`TpGeneralException,TpFWException`

Method
`disableHeartBeat()`

Allows the stop of the heartbeat supervision of the application.

Parameters

`session : in TpSessionID`

Identifies the heartbeat session.

Raises

`TpGeneralException,TpFWException`

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.

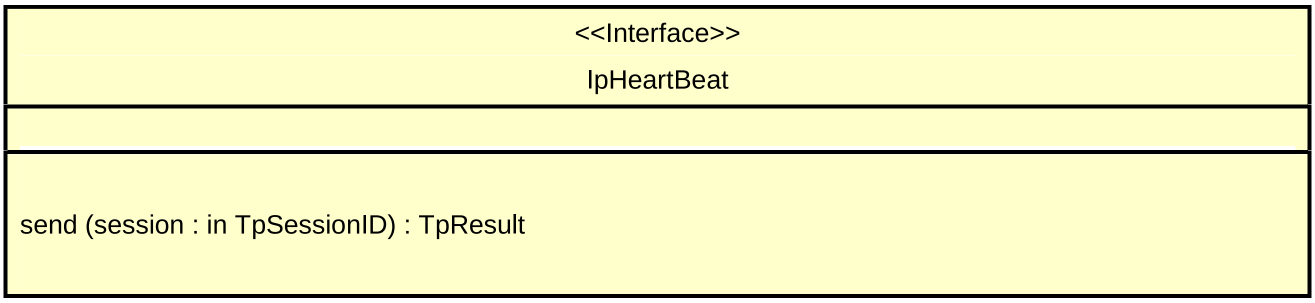
Raises

`TpGeneralException,TpFWException`

8.3.6 Interface Class IpHeartBeat

Inherits from: IpInterface.

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



Method
`send()`

This is the method the client application uses in case it supervises the Framework. The sender shall 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.

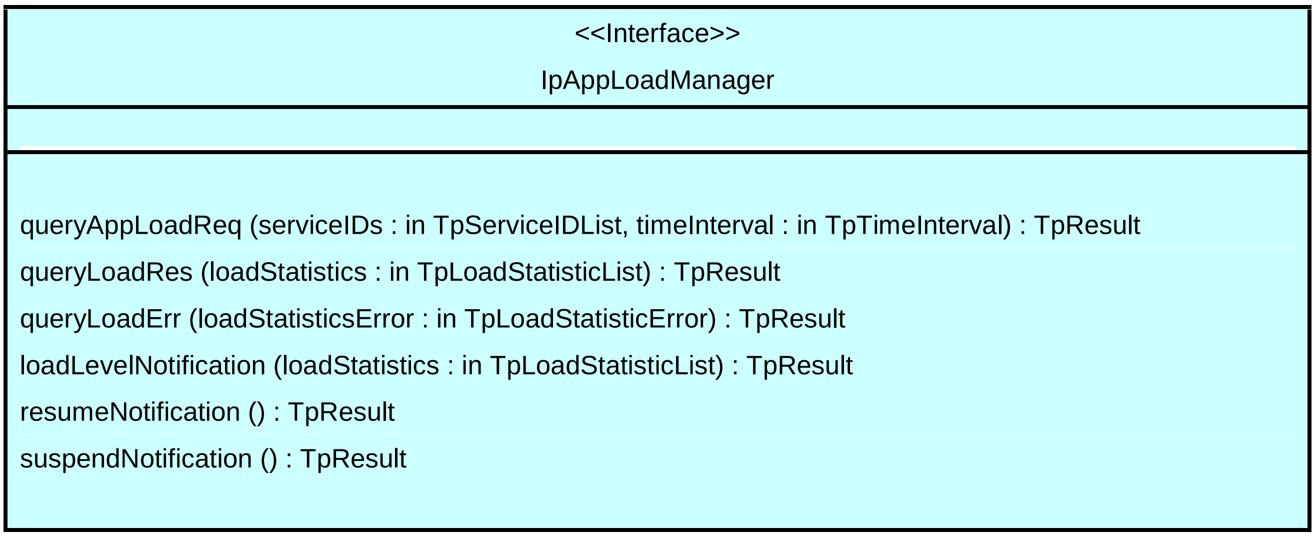
Raises

`TpGeneralException,TpFWException`

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 call-back 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) : TpResult

queryLoadRes (loadStatistics : in TpLoadStatisticList) : TpResult

queryLoadErr (loadStatisticsError : in TpLoadStatisticError) : TpResult

loadLevelNotification (loadStatistics : in TpLoadStatisticList) : TpResult

resumeNotification () : TpResult

suspendNotification () : TpResult

Method

`queryAppLoadReq()`

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

Parameters

`serviceIDs : in TpServiceIDList`

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

`timeInterval : in TpTimeInterval`

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

Raises

`TpGeneralException,TpFWException`

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

Raises

`TpGeneralException,TpFWException`

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.

Raises

`TpGeneralException,TpFWException`

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).

Raises

`TpGeneralException,TpFWException`

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.

Raises

`TpGeneralException,TpFWException`

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.

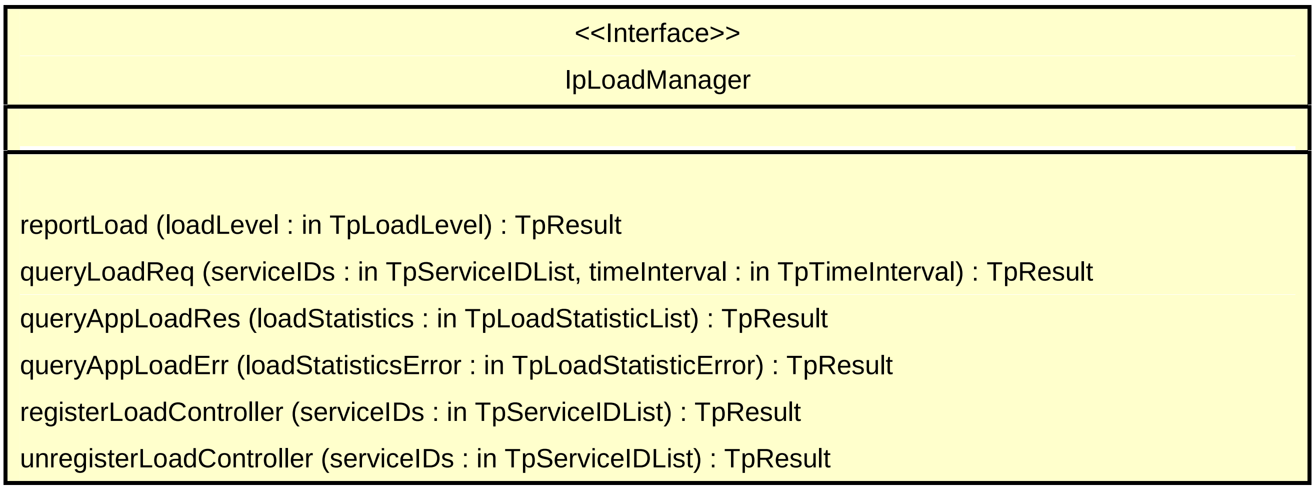
Raises

`TpGeneralException,TpFWException`

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 shall implement the IpAppLoadManager interface to provide the call-back mechanism. The application supplies the identity of this call-back interface at the time it obtains the Framework's load manager interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.


<<Interface>>

IpLoadManager

reportLoad (loadLevel : in TpLoadLevel) : TpResult

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

queryAppLoadRes (loadStatistics : in TpLoadStatisticList) : TpResult

queryAppLoadErr (loadStatisticsError : in TpLoadStatisticError) : TpResult

registerLoadController (serviceIDs : in TpServiceIDList) : TpResult

unregisterLoadController (serviceIDs : in TpServiceIDList) : TpResult resumeNotification (serviceIDs : in TpServiceIDList) : TpResult

suspendNotification (serviceIDs : in TpServiceIDList) : TpResult

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

`TpGeneralException,TpFWException`

Method
`queryLoadReq()`

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

Parameters

`serviceIDs : in TpServiceIDList`

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

`timeInterval : in TpTimeInterval`

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

Raises

`TpGeneralException,TpFWException`

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

`TpGeneralException,TpFWException`

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

`TpGeneralException,TpFWException`

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

`TpGeneralException,TpFWException`

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

TpGeneralException,TpFWException

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

TpGeneralException,TpFWException

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.

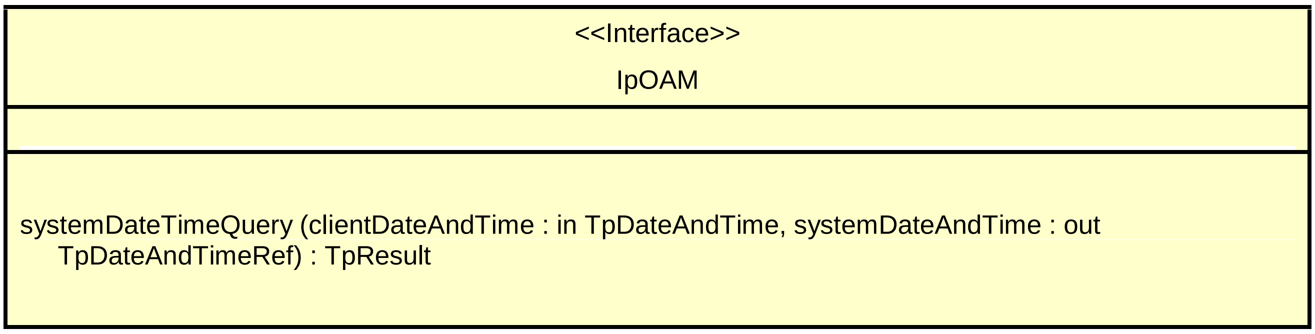
Raises

TpGeneralException,TpFWException

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 the OSA APIs.


<<Interface>>
IpOAM
systemDateTimeQuery (clientDateAndTime : in TpDateAndTime, systemDateAndTime : out TpDateAndTimeRef) : TpResult

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.

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.

`systemDateAndTime : out TpDateAndTimeRef`

This is the system date and time of the Framework.

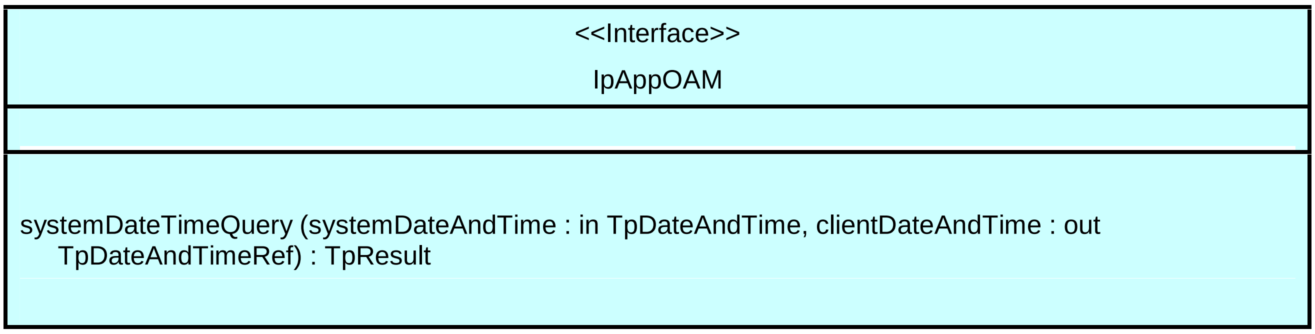
Raises

`TpGeneralException,TpFWException`

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 synchronization purposes. This method is invoked by the Framework to interchange the Framework and client application date and time.


<<Interface>>
IpAppOAM
systemDateTimeQuery (systemDateAndTime : in TpDateAndTime, clientDateAndTime : out TpDateAndTimeRef) : TpResult

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.

Parameters

`systemDateAndTime : in TpDateAndTime`

This is the system date and time of the Framework.

`clientDateAndTime : out TpDateAndTimeRef`

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.

Raises

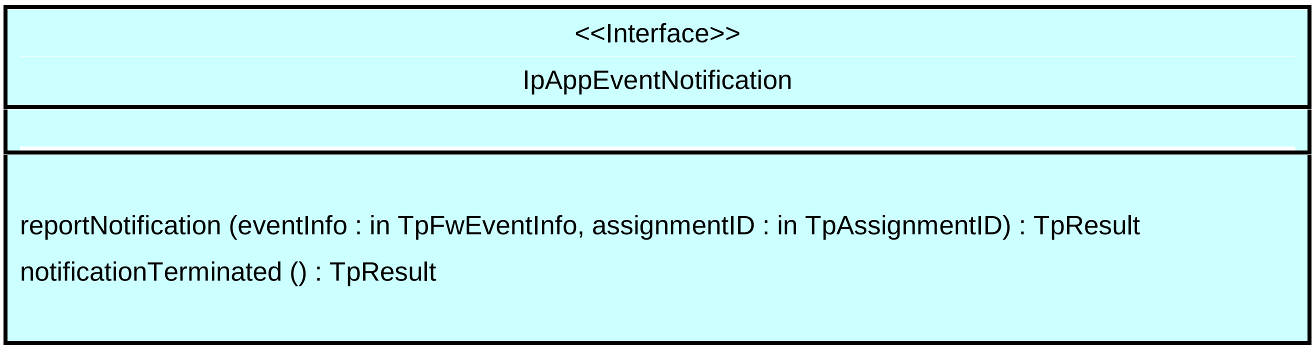
`TpGeneralException,TpFWException`

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) : TpResult
notificationTerminated () : TpResult

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.

Raises

`TpGeneralException,TpFWException`

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.

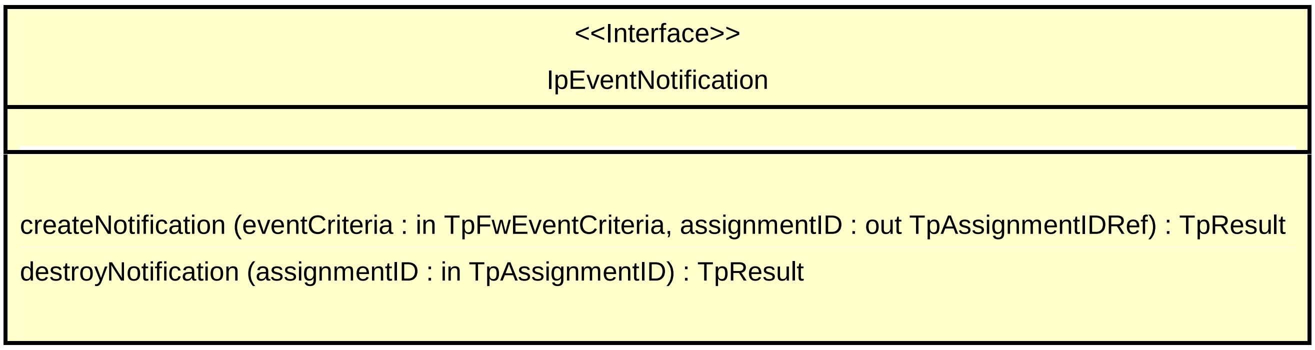
Raises

`TpGeneralException,TpFWException`

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, assignmentID : out TpAssignmentIDRef) : TpResult
destroyNotification (assignmentID : in TpAssignmentID) : TpResult

Method
`createNotification()`

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

Parameters

`eventCriteria : in TpFwEventCriteria`
Specifies the event specific criteria used by the application to define the event required.

`assignmentID : out TpAssignmentIDRef`
Specifies the ID assigned by the Framework for this newly installed notification.

Raises

`TpGeneralException,TpFWException`

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

`TpGeneralException,TpFWException`

9 Framework-to-Application State Transition Diagrams

This clause 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.

9.1 Trust and Security Management State Transition Diagrams

9.1.1 State Transition Diagrams for IpInitial

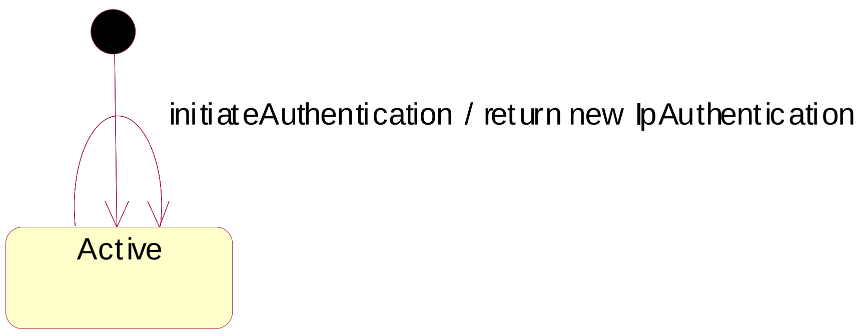


Figure : State Transition Diagram for IpInitial

9.1.1.1 Active State

9.1.2 State Transition Diagrams for IpAPILevelAuthentication

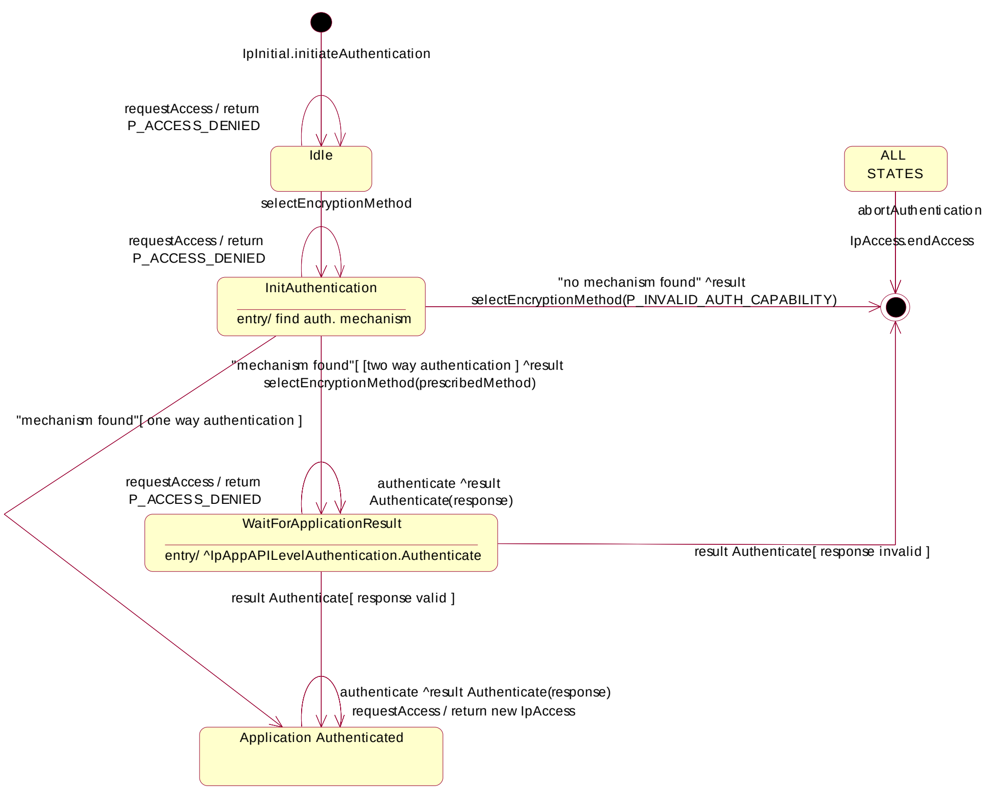

IpInitial.initiateAuthentication
requestAccess / return P_ACCESS_DENIED
Idle
selectEncryptionMethod
ALL STATES
abortAuthentication
IpAccess.endAccess
requestAccess / return P_ACCESS_DENIED
InitAuthentication
entry/ find auth. mechanism
"no mechanism found" ^result selectEncryptionMethod(P_INVALID_AUTH_CAPABILITY)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

Figure : State Transition Diagram for IpAPILevelAuthentication

9.1.2.1 Idle State

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

9.1.2.2 InitAuthentication State

In this state the Framework selects the preferred authentication mechanism within the capability of the application. When a proper mechanism is found, the Framework can decide that the application doesn't have to be authenticated (one way authentication) or that the application has to be authenticated. In case no mechanism can be found the error code P_INVALID_AUTH_CAPABILITY is returned and the Authentication object is destroyed. This implies that the application has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface.

9.1.2.3 WaitForApplicationResult State

When entering this state, the Framework requests the application to authenticate itself by invoking the Authenticate method on the application. In case the application requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework provides the correct response to the challenge of the application. When the Framework responds to the Authenticate request, the response is analysed and in case the response is valid a transition to the state Application Authenticated is made. In case the response is not valid, the Authentication object is destroyed. This implicates that the application has to re-initiate the authentication by calling once more the initiateAuthentication method on the IpInitial interface.

9.1.2.4 Application Authenticated State

In this state the application is considered authenticated and is now allowed to request access to the IpAccess interface. In case the application requests the Framework to authenticate itself by invoking Authenticate on the IpAPILevelAuthentication interface, the Framework provides the correct response to the challenge of the application.

9.1.3 State Transition Diagrams for IpAccess

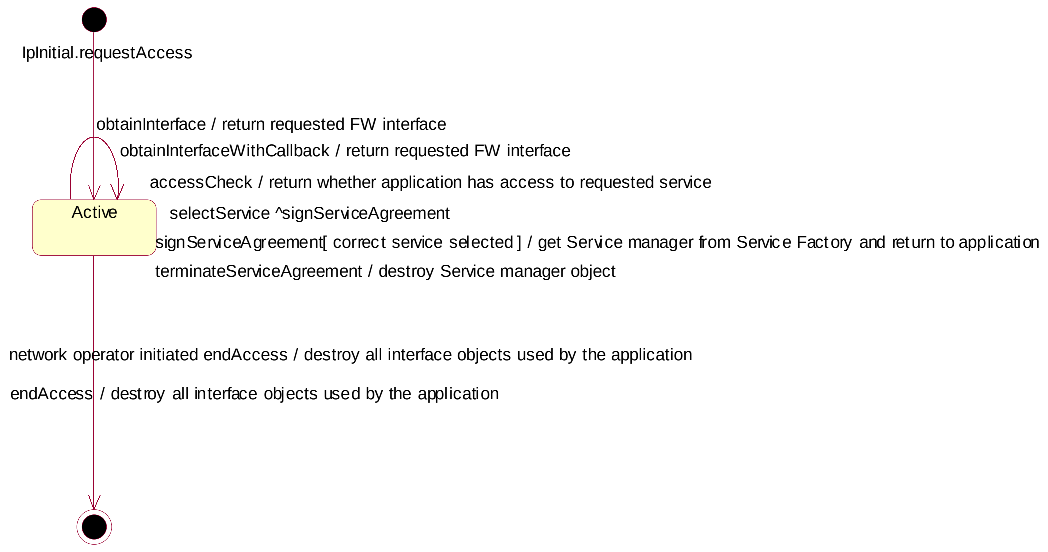

IpInitial.requestAccess
obtainInterface / return requested FW interface
obtainInterfaceWithCallback / return requested FW interface
accessCheck / return whether application has access to requested service
Active
selectService ^signServiceAgreement
signServiceAgreement[correct service selected] / get Service manager from Service Factory and return to application
terminateServiceAgreement / destroy Service manager object
network operator initiated endAccess / destroy all interface objects used by the application
endAccess / destroy all interface objects used by the application

Figure : State Transition Diagram for IpAccess

9.1.3.1 Active State

When the application requests access to the Framework on the IpInitial interface, an object implementing the IpAccess interface is created. The application can now request other Framework interfaces, including Service Discovery. When the application 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 application. In case the network operator decides that the application has no longer access to the interfaces the same will happen.

Service Discovery State Transition Diagrams

9.2.1 State Transition Diagrams for IpServiceDiscovery

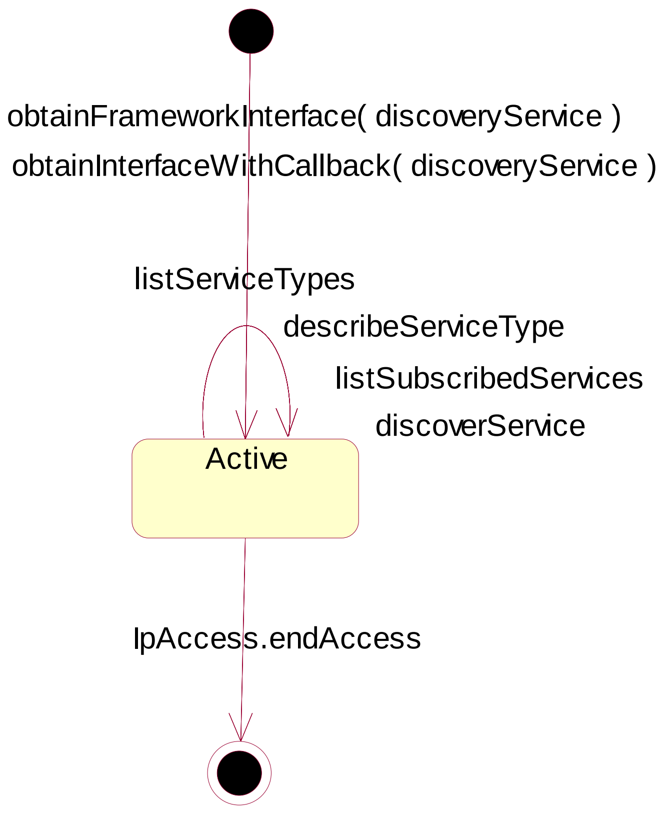


Figure : State Transition Diagram for IpServiceDiscovery

9.2.1.1 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.

9.3 Integrity Management State Transition Diagrams

9.3.1 State Transition Diagrams for IpHeartBeatMgmt

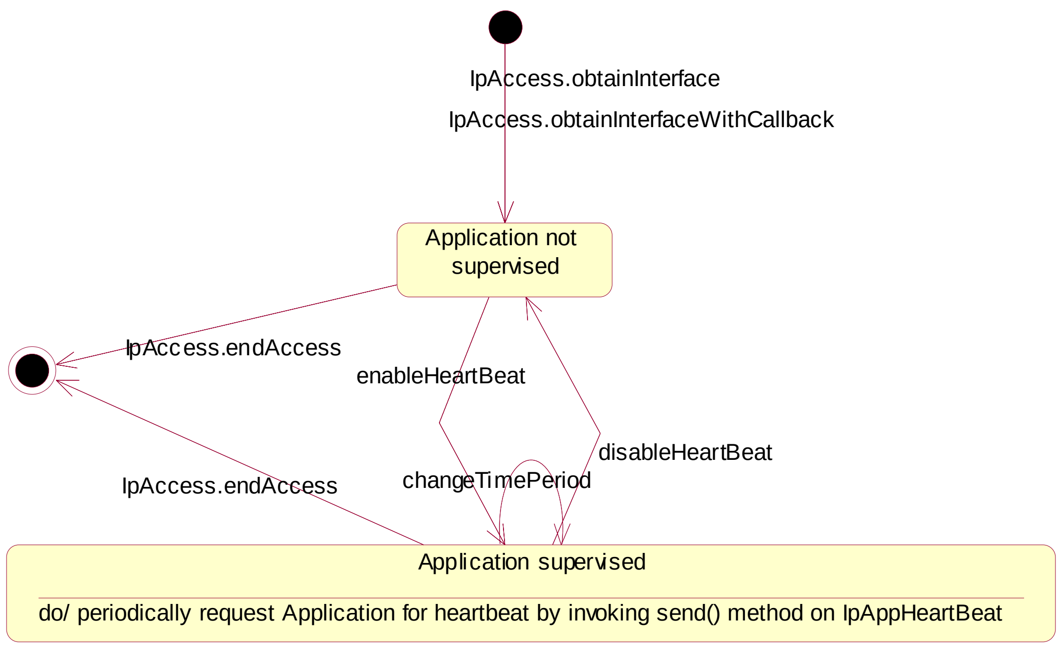

IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
Application not supervised
IpAccess.endAccess
enableHeartBeat
disableHeartBeat
IpAccess.endAccess
changeTimePeriod
Application supervised
do/ periodically request Application for heartbeat by invoking send() method on IpAppHeartBeat

Figure : State Transition Diagram for IpHeartBeatMgmg

9.3.1.1 Application not supervised State

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

9.3.1.2 Application supervised State

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

9.3.2 State Transition Diagrams for IpHeartBeat

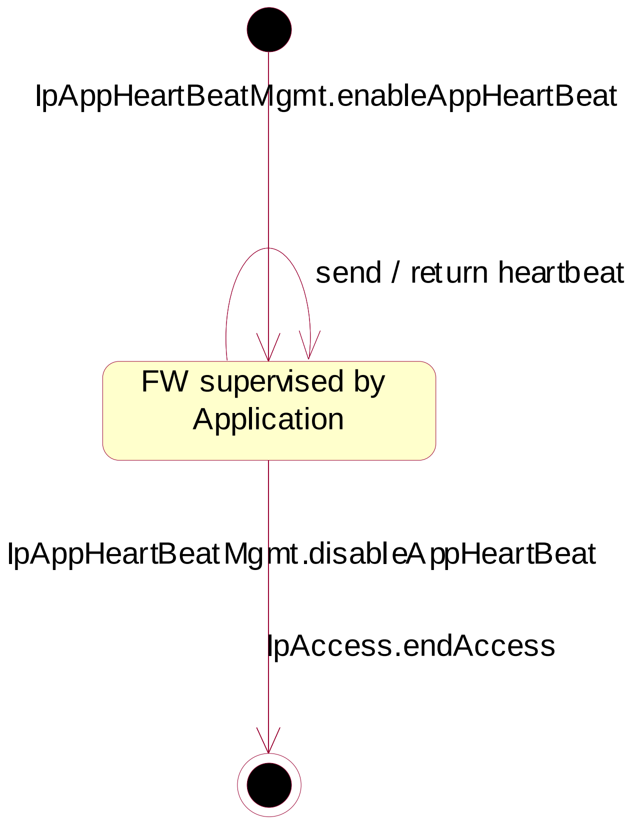

IpAppHeartBeatMgmt.enableAppHeartBeat
send / return heartbeat
FW supervised by Application
IpAppHeartBeatMgmt.disableAppHeartBeat
IpAccess.endAccess

Figure : State Transition Diagram for IpHeatBeat

9.3.2.1 FW supervised by Application State

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

9.3.3 State Transition Diagrams for IpLoadManager

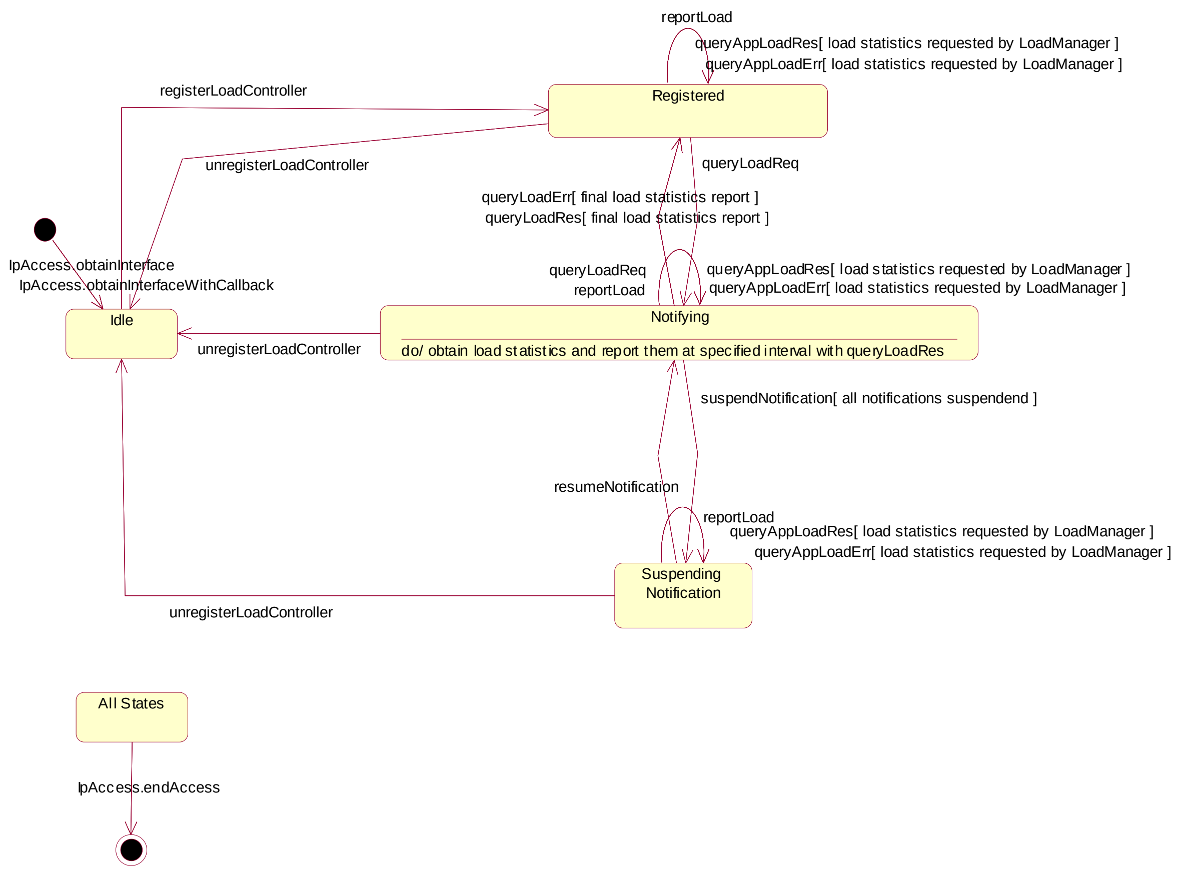

reportLoad
queryAppLoadRes[load statistics requested by LoadManager]
queryAppLoadErr[load statistics requested by LoadManager]
registerLoadController
Registered
unregisterLoadController
queryLoadReq
queryLoadErr[final load statistics report]
queryLoadRes[final load statistics report]
IpAccess.obtainInterface
IpAccess.obtainInterfaceWithCallback
queryLoadReq
reportLoad
queryAppLoadRes[load statistics requested by LoadManager]
queryAppLoadErr[load statistics requested by LoadManager]
Idle
Notifying
unregisterLoadController
do/ obtain load statistics and report them at specified interval with queryLoadRes
suspendNotification[all notifications suspendend]
resumeNotification
reportLoad
queryAppLoadRes[load statistics requested by LoadManager]
queryAppLoadErr[load statistics requested by LoadManager]
Suspending Notification
unregisterLoadController
All States
IpAccess.endAccess

Figure : State Transition Diagram for IpLoadManager

9.3.3.1 Idle State

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

9.3.3.2 Notifying State

In the Notifying state the application has requested for load statistics. The LoadManager gathers the requested information and (periodically) reports them to the application.

9.3.3.3 Suspending Notification State

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

9.3.3.4 Registered State

In this state the application has registered for load control with the method RegisterLoadController(). The LoadManager 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() 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().

When entering this state, an object called LoadManagerInternal is created that has an internal state machine encapsulating the internal behaviour of the LoadManager. The State Transition Diagram of LoadManagerInternal is shown in Figure .

9.3.4 State Transition Diagrams for IpLoadManagerInternal

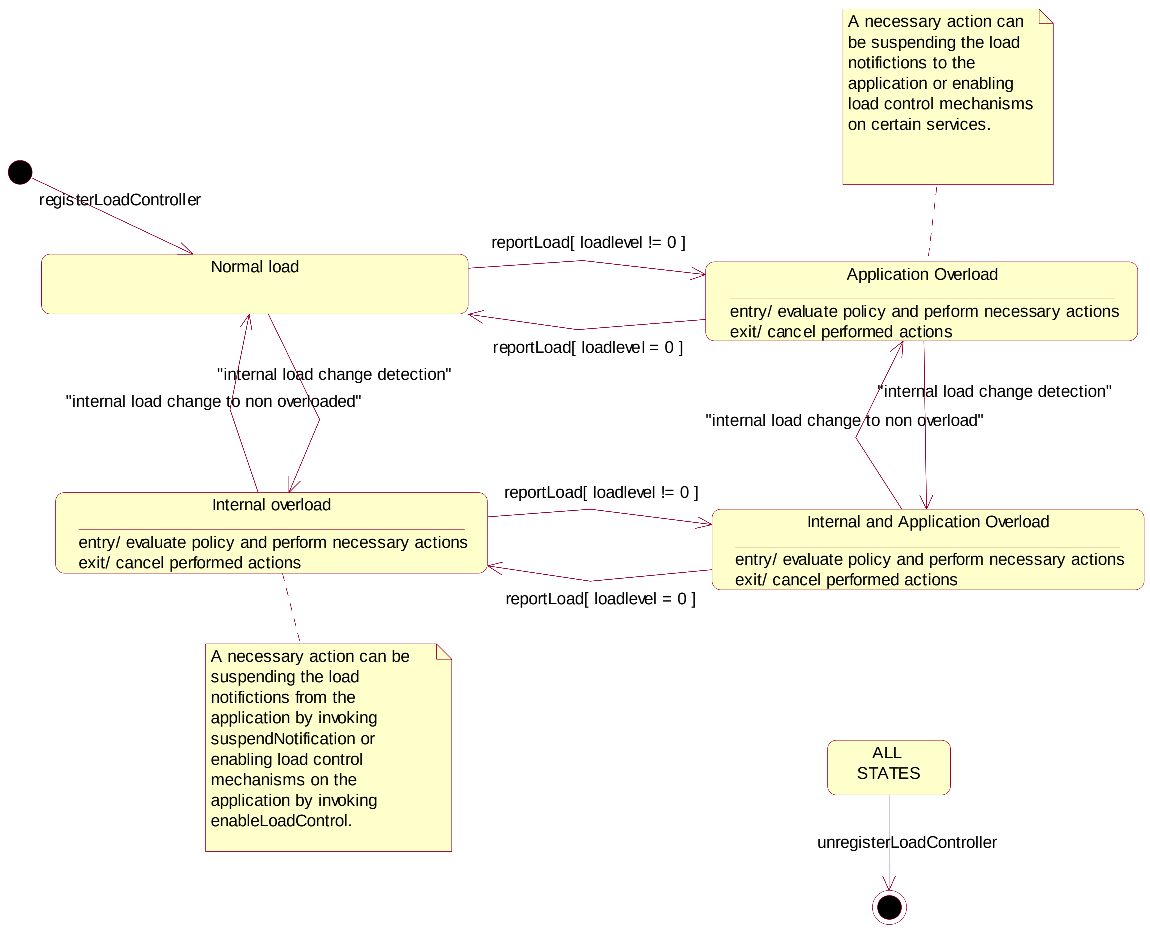

A necessary action can be suspending the load notifictions to the application or enabling load control mechanisms on certain services.
registerLoadController
Normal load
reportLoad[loadlevel != 0]
Application Overload
entry/ evaluate policy and perform necessary actions
exit/ cancel performed actions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

Figure : State Transition Diagram for IpLoadManagerInternal

9.3.4.1 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.

9.3.4.2 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.

9.3.4.3 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.

9.3.4.4 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.

9.3.5 State Transition Diagrams for IpOAM

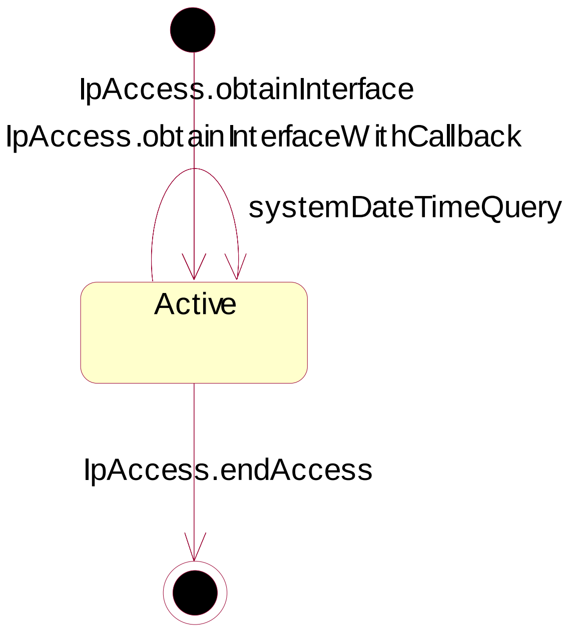


Figure : State Transition Diagram for IpOAM

9.3.5.1 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.

9.3.6 State Transition Diagrams for IpFaultManager

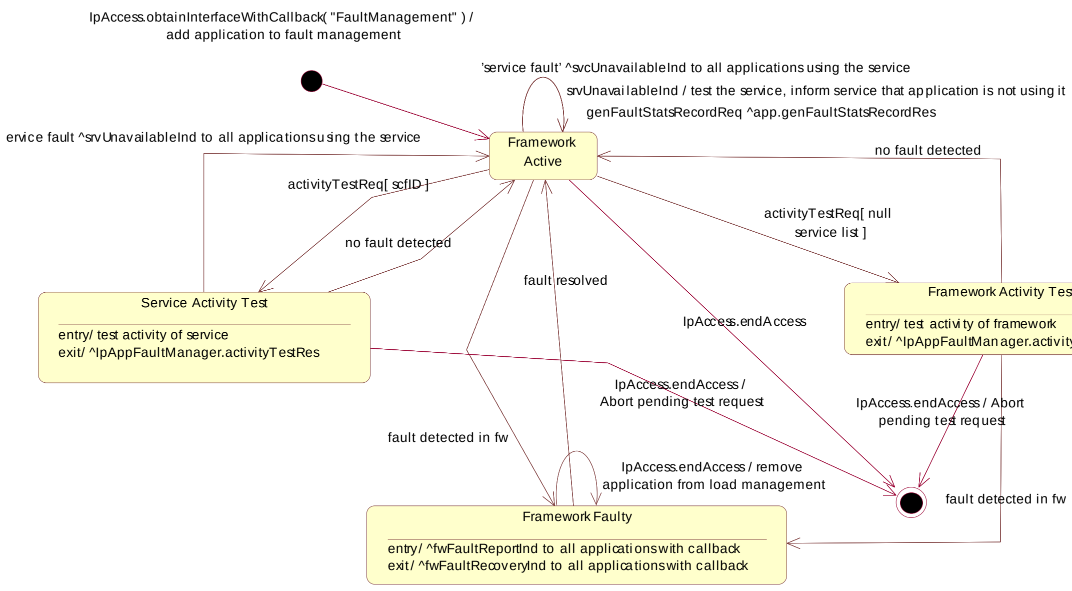

IpAccess.obtainInterfaceWithCallback("FaultManagement") / add application to fault management
'service fault' ^svcUnavailableInd to all applications using the service
srvUnavailableInd / test the service, inform service that application is not using it
genFaultStatsRecordReq ^app.genFaultStatsRecordRes
ervice fault ^srvUnavailableInd to all applications using the service
Framework Active
no fault detected
activityTestReq[scfID]
no fault detected
activityTestReq[null service list]
fault resolved
IpAccess.endAccess
Service Activity Test
entry/ test activity of service
exit/ ^IpAppFaultManager.activityTestRes
Framework Activity Test
entry/ test activity of framework
exit/ ^IpAppFaultManager.activityTestRes
IpAccess.endAccess / Abort pending test request
IpAccess.endAccess / Abort pending test request
fault detected in fw
IpAccess.endAccess / remove application from load management
fault detected in fw
Framework Faulty
entry/ ^fwFaultReportInd to all applications with callback
exit/ ^fwFaultRecoveryInd to all applications with callback

Figure : State Transition Diagram for IpFaultManager

9.3.6.1 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.

9.3.6.2 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 call-backs will be notified via a fwFaultRecoveryInd message.

9.3.6.3 Framework Activity Test State

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

9.3.6.4 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 call-backs are notified accordingly through a svcUnavailableInd message.

9.4 Event Notification State Transition Diagrams

9.4.1 State Transition Diagrams for IpEventNotification

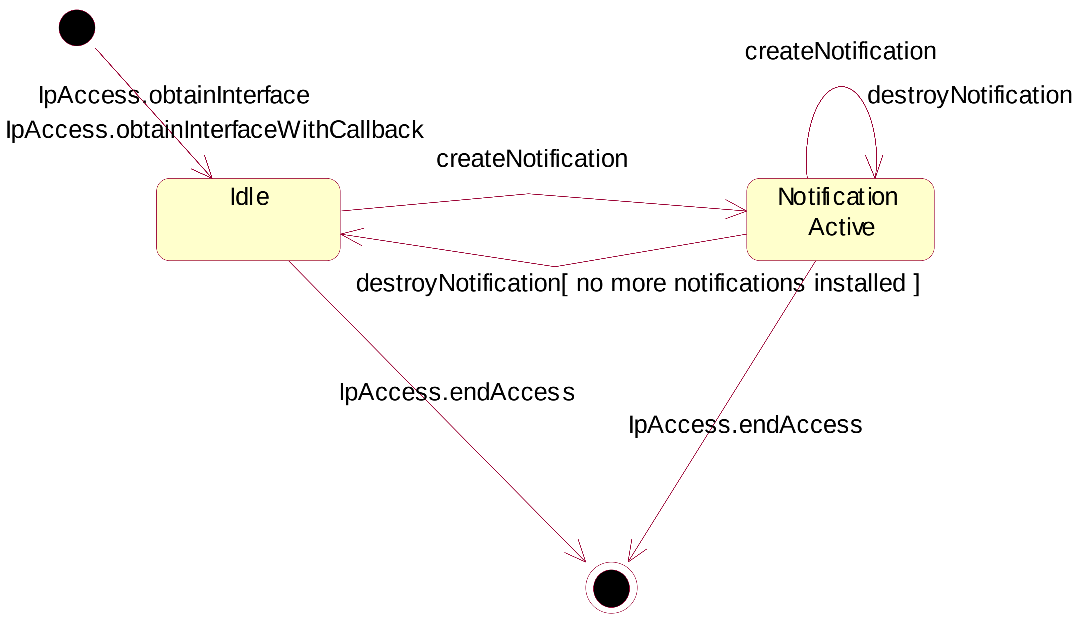


Figure : State Transition Diagram for IpEventNotification

9.4.1.1 Idle State

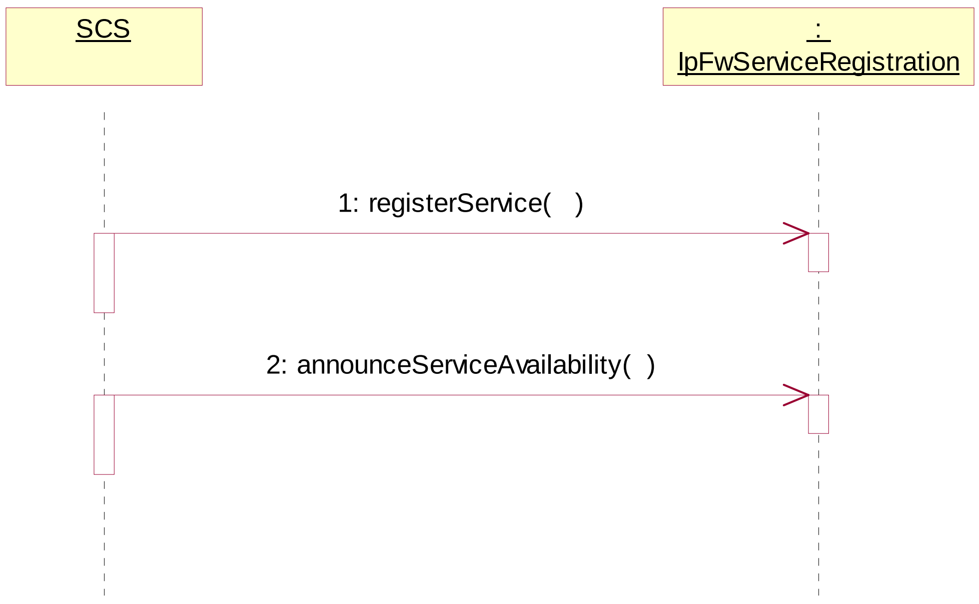
9.4.1.2 Notification Active State

10 Framework-to-Service Sequence Diagrams

10.1 Service Registration Sequence Diagrams

10.1.1 New SCF Registration

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



1: Registration: first step - register service

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

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

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

· in serviceTypeName

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

· in servicePropertyList

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

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

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

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

The following output parameter results from service registration:

· out serviceID

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

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

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

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

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

2: Registration: second step - announce service availability

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

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

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

· in serviceID

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

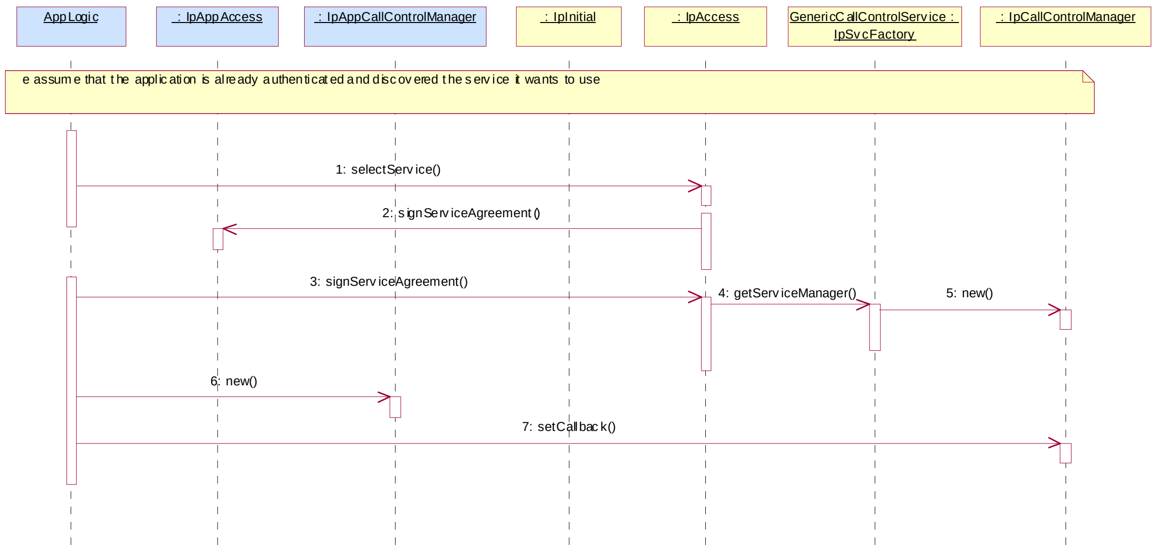
· in serviceFactoryRef

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

Service Factory Sequence Diagrams

10.2.1 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 clauses.


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

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

2: The Framework signs the service agreement.

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

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

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

6: The application creates a new IpAppCallControlManager interface to be used for call-backs.

7: The Application sets the call-back interface to the interface created with the previous message.

11 Framework-to-Service Class Diagrams

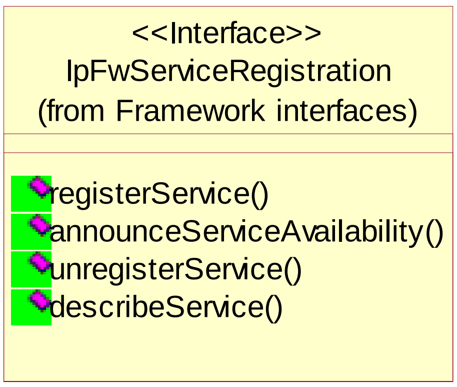


Figure: Service Registration Package Overview

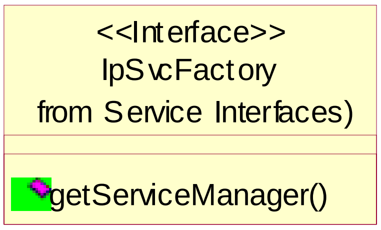


Figure: Service Factory Package Overview

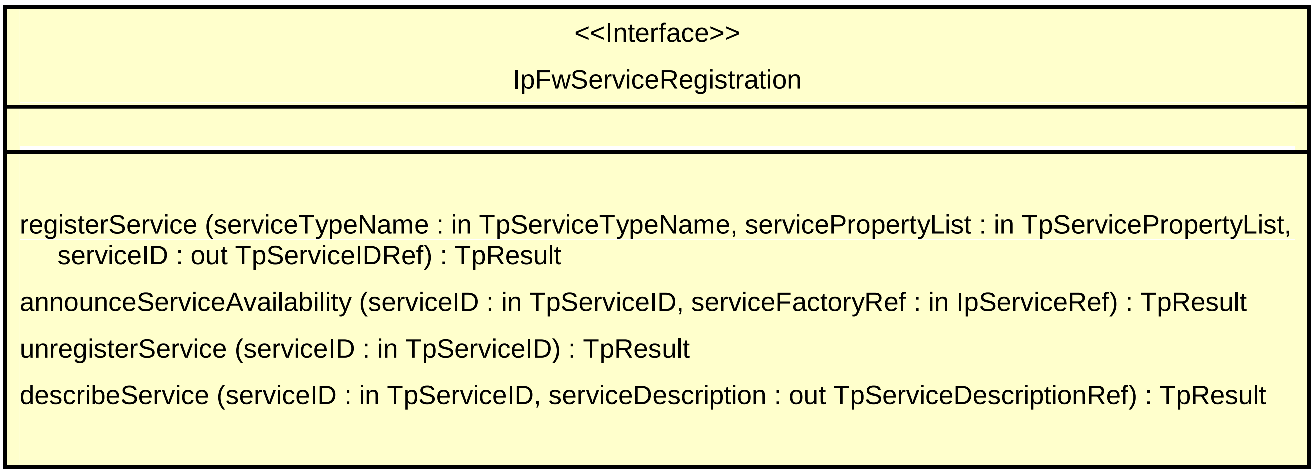
Framework-to-Service Interface Classes

12.1 Service Registration Interface Classes

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, serviceID : out TpServiceIDRef) : TpResult

announceServiceAvailability (serviceID : in TpServiceID, serviceFactoryRef : in IpServiceRef) : TpResult

unregisterService (serviceID : in TpServiceID) : TpResult

describeService (serviceID : in TpServiceID, serviceDescription : out TpServiceDescriptionRef) : TpResult

Method

registerService()

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

Parameters

serviceTypeName : in TpServiceTypeName

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

servicePropertyList : in TpServicePropertyList

The "servicePropertyList" parameter is a list of property name and property value pairs. They describe the service being registered. This description typically covers behavioural, non-functional and non-computational aspects of the service. Service properties are marked "mandatory" or "read-only". These property mode attributes have the following semantics: a. mandatory - a service associated with this service type shall provide an appropriate value for this property when registering.

b. read-only - 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 shall 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 read-only property, then the P_READONLY_DYNAMIC_PROPERTY exception is raised. If the "servicePropertyList" parameter omits any property declared in the service type with a mode of mandatory, then a P_MISSING_MANDATORY_PROPERTY exception is raised. If two or more properties with the same property name are included in this parameter, the P_DUPLICATE_PROPERTY_NAME exception is raised.

serviceID : out TpServiceIDRef

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

Raises

TpGeneralException,TpFWException

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 getServiceManager(application: in TpClientAppID, serviceManager: out IpServiceRefRef). When the service agreement is signed for some serviceID (using signServiceAgreement()), the Framework calls the getServiceManager() 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 IpServiceRef

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

Raises

TpGeneralException,TpFWException

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. After the unregisterService(), the service can no longer be discovered by the enterprise client application.

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

`TpGeneralException,TpFWException`

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.

This operation is intended to be used between a certain Framework and the SCS that registered the SCF, since it is only between them that the serviceID is valid. 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. Getting the description of these SCFs from the Framework where they have been registered helps the SCS internal maintenance.

Parameters

`serviceID : in TpServiceID`

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

`serviceDescription : out TpServiceDescriptionRef`

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

Raises

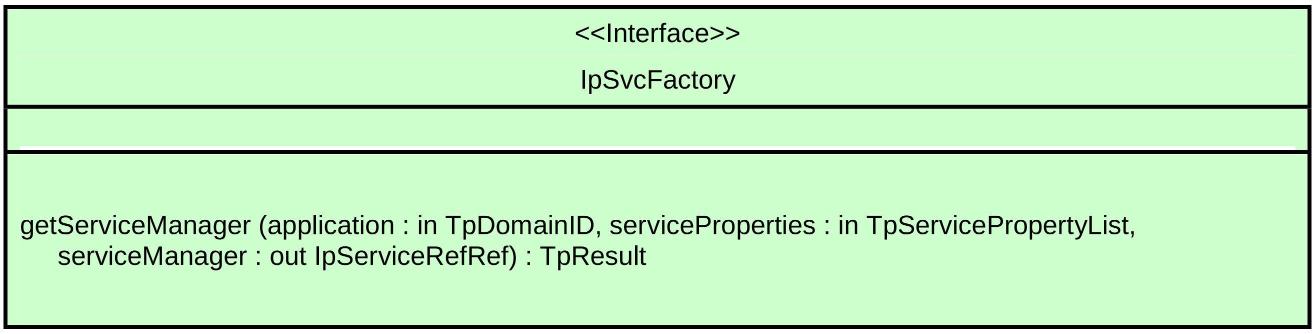
`TpGeneralException,TpFWException`

12.2 Service Factory Interface Classes

12.2.1 Interface Class IpSvcFactory

Inherits from: IpInterface.

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.


<<Interface>>
IpSvcFactory
getServiceManager (application : in TpDomainID, serviceProperties : in TpServicePropertyList, serviceManager : out IpServiceRefRef) : TpResult

Method
`getServiceManager()`

This method returns a service manager interface reference for the specified application. Usually, but not necessarily, this involves the instantiation of a new service manager interface.

Parameters

`application : in TpDomainID`

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

`serviceProperties : in TpServicePropertyList`

`serviceManager : out IpServiceRefRef`

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

Raises

`TpGeneralException,TpFWException`

13 Framework-to-Service State Transition Diagrams

13.1 Service Registration State Transition Diagrams

13.1.1 State Transition Diagrams for IpFwServiceRegistration

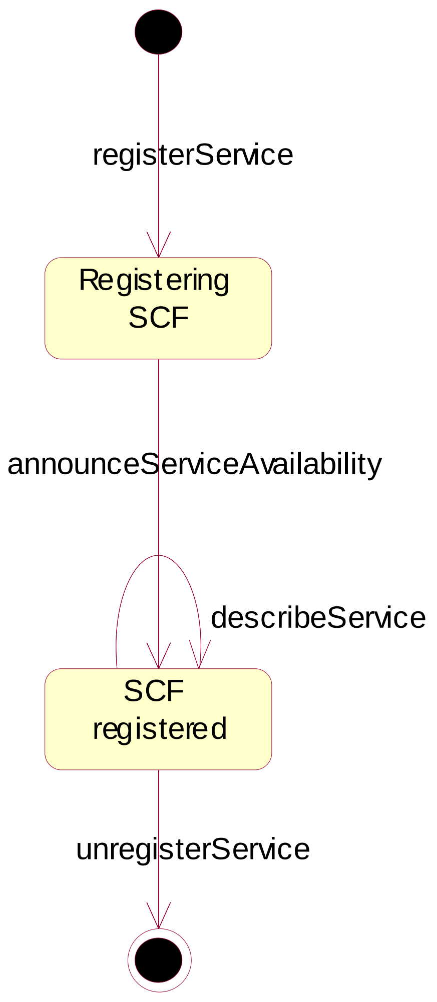


Figure : State Transition Diagram for IpFwServiceRegistration

13.1.1.1 Registering SCF State

This is the state entered when a Service Capability Server (SCS) starts the registration of 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. When receiving this ID, the SCS instantiates a manager interface for this SCF, which will be the entry point for applications that want to use it.

13.1.1.2 SCF registered State

This is the state entered when, the service manager interface having been instantiated, the SCS informs the Framework of the availability of the SCF, and makes it actually available by providing the Framework with the manager interfaces to be used by applications. Anytime the SCF availability may be withdrawn by un-registering it.

13.2 Service Factory State Transition Diagrams

There are no State Transition Diagrams defined for Service Factory.

14 Service Properties

14.1 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 getServiceManager() operation in the Service Factory 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.

14.2 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

14.2.1 Service Name

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

14.2.2 Service Version

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

14.2.3 Service Instance ID

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

14.2.4 Service Instance Description

This property contains a textual description of the service.

14.2.5 Product Name

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

14.2.6 Product Version

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

14.2.7 Supported Interfaces

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

14.2.8 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”}.

15 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.

15.1 Common Framework Data Definitions

15.1.1 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.

15.1.2 TpClientAppIDList

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

15.1.3 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_REGISTERED_SERVICE	TpServiceID	`ServiceID`
P_SERVICE_SUPPLIER	TpServiceSupplierID	`ServiceSupplierID`

15.1.4 TpDomainIDType

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

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

15.1.5 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).

15.1.6 TpPropertyName

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

15.1.7 TpPropertyValue

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

15.1.8 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

15.1.9 TpPropertyList

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

15.1.10 TpEntOpIDList

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

15.1.11 TpFwID

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

15.1.12 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	
`ServicePropertyList`	TpServicePropertyList	

15.1.13 TpServiceList

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

15.1.14 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	

15.1.15 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, and comprises a TpUniqueServiceNumber, TpServiceTypeName, and a number of relevant TpServiceSpecString, which are concatenated using a forward separator (/) as the separation character.

15.1.16 TpServiceIDList

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

15.1.17 TpServiceIDRef

Defines a Reference to type TpServiceId.

15.1.18 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

15.1.19 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).

15.1.20 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	
ServicePropertyMode	TpServicePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

15.1.21 TpServiceTypePropertyList

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

15.1.22 TpServicePropertyMode

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 may not be modified
MANDATORY_READONLY	3	The value of the corresponding SCF property type shall be provided and subsequently it may not be modified.

15.1.23 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.

15.1.24 TpServicePropertyName

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

15.1.25 TpServicePropertyNameList

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

15.1.26 TpServicePropertyValue

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

15.1.27 TpServicePropertyValueList

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

15.1.28 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	

15.1.29 TpServicePropertyList

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

15.1.30 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`.

15.1.31 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
EnabledOrDisabled	TpBoolean	an indication whether the SCF type is enabled (true) or disabled (false)

15.1.32 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_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

15.1.33 TpServiceTypeNameList

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

15.2 Event Notification Data Definitions

15.2.1 TpFwEventName

Defines the name of event being notified.

Name	Value	Description
P_EVENT_FW_NAME_UNDEFINED	0	Undefined
P_EVENT_FW_NEW_SERVICE_AVAILABLE	1	Notification of a new SCS available

15.2.2 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_NEW_SERVICE_AVAILABLE	TpServiceTypeNameList	ServiceTypeNameList

15.2.3 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`

15.3 Trust and Security Management Data Definitions

15.3.1 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 IpAppAccess

15.3.2 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 IpAppAPILevelAuthentication
`P_AUTHENTICATION`	Authenticate using the implementation specific authentication mechanism, e.g. CORBA Security.

15.3.3 TpAuthCapability

This data type is identical to a TpString, and is defined as a string of characters that identify the authentication capabilities that could be supported by the OSA. 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.

15.3.4 TpAuthCapabilityList

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

15.3.5 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).

15.3.6 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 to access another.

15.3.7 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_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.

15.3.8 TpServiceAccessControl

This is Sequence of Data Elements containing the access control policy information controlling access to the service capability feature, and the trustLevel that the Network operator has assigned to the client application.

Sequence Element Name	Sequence Element Type
`Policy`	`TpString`
`TrustLevel`	`TpString`

The policy parameter indicates whether access has been granted or denied. If granted then the parameter trustLevel shall also have a value.

The trustLevel parameter indicates the trust level that the Network operator has assigned to the client application.

15.3.9 TpSecurityContext

This data type is identical to a TpString and contains a group of security relevant attributes.

15.3.10 TpSecurityDomain

This data type is identical to a TpString and contains the security domain in which the client application is operating.

15.3.11 TpSecurityGroup

This data type is identical to a TpString and contains a definition of the access rights associated with all clients that belong to that group.

15.3.12 TpServiceAccessType

This data type is identical to a TpString and contains a definition of the specific security model in use.

15.3.13 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.

15.3.14 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`
`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.

15.3.15 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

15.4 Integrity Management Data Definitions

15.4.1 TpActivityTestRes

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

15.4.2 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

15.4.3 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.

15.4.4 TpFaultStatsSet

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

15.4.5 TpActivityTestID

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

15.4.6 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

15.4.7 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)

15.4.8 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

15.4.9 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

15.4.10 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

15.4.11 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

15.4.12 TpTimeInterval

Defines the Sequence of Data Elements that specify a time interval.

Sequence Element Name	Sequence Element Type
StartTime	TpDateAndTime
StopTime	TpDateAndTime

15.4.13 TpLoadPolicy

Defines the load balancing policy.

Sequence Element Name	Sequence Element Type
LoadPolicy	TpString

15.4.14 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

15.4.15 TpLoadStatisticList

Defines a `Numbered List of Data Elements` of type TpLoadStatistic.

15.4.16 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.	

15.4.17 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

15.4.18 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

15.4.19 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`

15.4.20 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

15.4.21 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`

15.5 Service Subscription Data Definitions

15.5.1 TpPropertyName

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

15.5.2 TpPropertyValue

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

15.5.3 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

15.5.4 TpPropertyList

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

15.5.5 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).

15.5.6 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

15.5.7 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.

15.5.8 TpPersonName

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

15.5.9 TpPostalAddress

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

15.5.10 TpTelephoneNumber

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

15.5.11 TpEmail

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

15.5.12 TpHomePage

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

15.5.13 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".

15.5.14 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

15.5.15 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).

15.5.16 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).

15.5.17 TpServiceRequestor

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

15.5.18 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.

15.5.19 TpServiceSubscriptionProperties

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

15.5.20 TpServiceContract

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
`ServiceContractID`	TpServiceContractID
`ServiceRequestor`	TpServiceRequestor
`BillingContact`	TpBillingContact
`ServiceStartDate`	TpServiceStartDate
`ServiceEndDate`	TpServiceEndDate
`ServiceTypeName`	TpServiceTypeName
`ServiceID`	TpServiceID
`ServiceSubscriptionProperties`	TpServiceSubscriptionProperties

15.5.21 TpPassword

This data type is identical to `TpString`. It is a password assigned to a client application for authentication purposes.

15.5.22 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.

15.5.23 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
`Password`	TpPassword
`ClientAppProperties`	TpClientAppProperties

15.5.24 TpSagID

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

15.5.25 TpSagIDList

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

15.5.26 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".

15.5.27 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

15.5.28 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.

15.5.29 TpServiceProfileIDList

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

15.5.30 TpServiceProfile

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
`ServiceProfileID`	TpServiceProfileID
`ServiceContractID`	TpServiceContractID
`ServiceStartDate`	TpServiceStartDate
`ServiceEndDate`	TpServiceEndDate
`ServiceTypeName`	TpServiceTypeName
`ServiceSubscriptionProperties`	TpServiceSubscriptionProperties

Annex A (normative): OMG IDL Description of Framework

The OMG IDL representation of this interface specification is contained in a text file (fw.idl contained in archive2919803IDL.ZIP) which accompanies 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 IpService Registration

Interface Class IpServiceRegistration in R99 renamed IpFwServiceRegistration

B.2 IDL Namespace

IDL namespace has been extended. Instead of all interfaces being under org::open-service-access::fw, now all interfaces except IpFwServiceRegistratin and IpSvcFactory are under fw::fw_client, and IpFwServiceRegistration and IpSvcFactory are under fw::fw_service

B.3 IpAccess

accessCheck(serviceToken: in TpServiceToken,securityContext: in ~~TpString~~TpSecurityContext, securityDomain: in ~~TpString~~TpSecurityDomain, group : in ~~TpString~~TpSecurityGroup, serviceAccessTypes: in ~~TpString~~TpServiceAccessType, serviceAccessControl: out TpServiceAccessControlRef): TpResult

B.4 IpAPILevelAuthentication, IpAppAPILevelAuthentication

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

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

B.5 New IpAuthentication

requestAccess (accessType : in TpAccessType, appAccessInterface : in IpInterfaceRef, fwAccessInterface : out IpInterfaceRefRef) : TpResult added.

B.6 IpInitial

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

B.7 IpAppLoadManager

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

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

loadLevelNotification(loadStatistics : in TpLoadStatisticList) : TpResult

B.8 Data Type Changes

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, 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.

~~**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	
ServicePropertyMode	TpServicePropertyMode	
ServicePropertyTypeName	TpServicePropertyTypeName	

TpServiceTypePropertyList

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

TpServicePropertyMode

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 may not be modified
MANDATORY_READONLY	3	The value of the corresponding SCF property type shall be provided and subsequently it may not be modified.

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	Sequence Element	Documentation
Name	Type	
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	Sequence Element	Documentation
Name	Type	
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
EnabledOrDisabled	TpBoolean	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.~~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 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

TpServiceTypeNameList

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

TpSecurityContext

This data type is identical to a TpString and contains a group of security relevant attributes.

TpSecurityDomain

This data type is identical to a TpString and contains the security domain in which the client application is operating.

TpSecurityGroup

This data type is identical to a TpString and contains a definition of the access rights associated with all clients that belong to that group.

TpServiceAccessType

This data type is identical to a TpString and contains a definition of the specific security model in use.

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

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
16 Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	4.0.0

History

Document history		
V4.0.0	March 2001	Publication