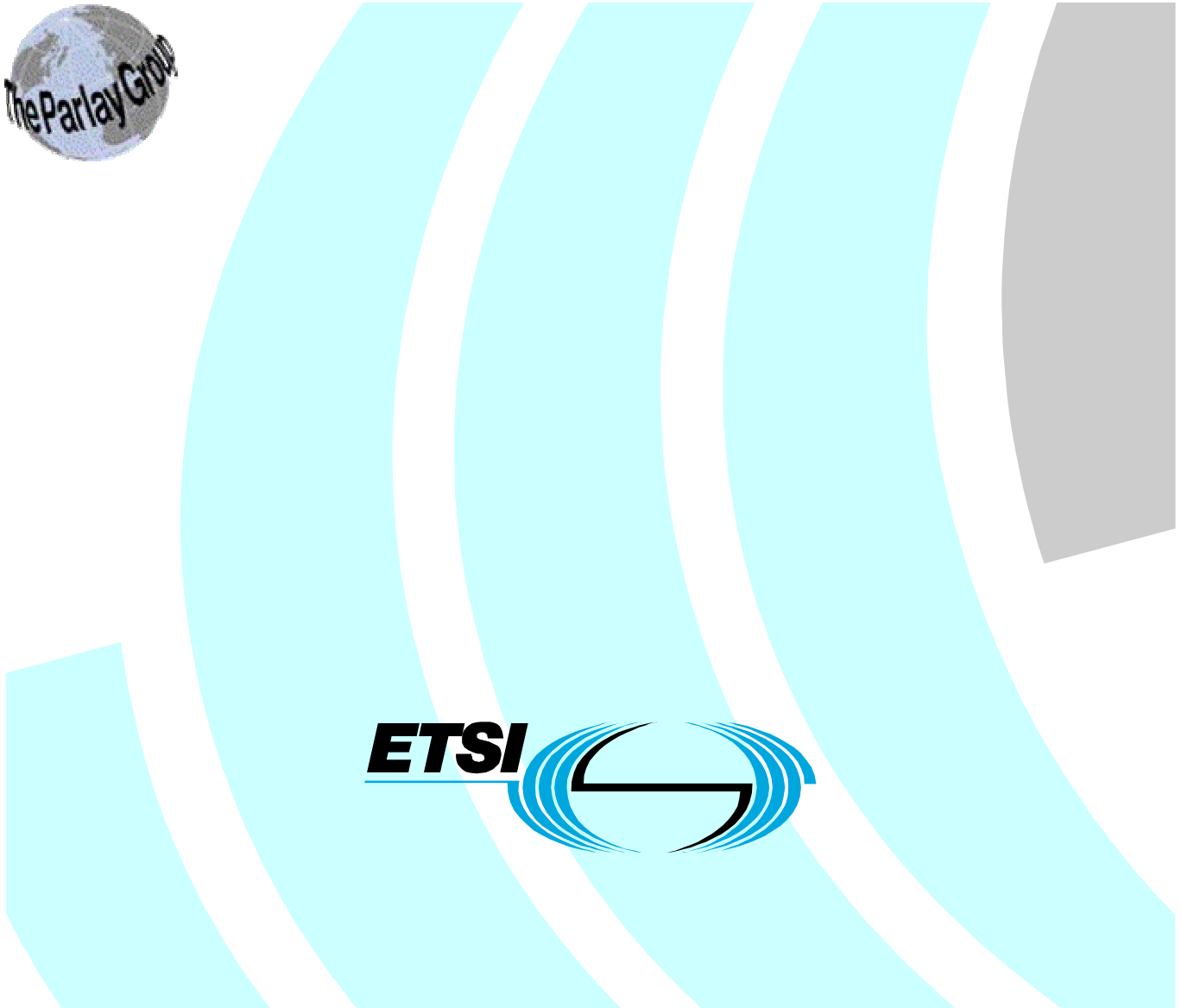
Final draft ETSI ES 201 915-3 V1.5.1 (2004-11)

ETSI Standard

Open Service Access (OSA); Application Programming Interface (API); Part 3: Framework (Parlay 3)



Reference
RES/TISPAN-01008-03-OSA

Keywords
API, OSA, IDL, UML

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

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Contents

Intellectual Property Rights 9
Foreword 9
1 Scope 10
2 References 10
3 Definitions and abbreviations 10
3.1 Definitions 10
3.2 Abbreviations 10
4 Overview of the Framework 10
4.1 General requirements on support of methods 12
5 The Base Interface Specification 12
5.1 Interface Specification Format 12
5.1.1 Interface Class 12
5.1.2 Method descriptions 13
5.1.3 Parameter descriptions 13
5.1.4 State Model 13
5.2 Base Interface 13
5.2.1 Interface Class IpInterface 13
5.3 Service Interfaces 13
5.3.1 Overview 13
5.4 Generic Service Interface 13
5.4.1 Interface Class IpService 13
6 Framework Access Session API 15
6.1 Sequence Diagrams 15
6.1.1 Trust and Security Management Sequence Diagrams 15
6.1.1.1 Initial Access for trusted parties 15
6.1.1.2 Initial Access 15
6.1.1.3 Authentication 17
6.1.1.4 API Level Authentication 17
6.2 Class Diagrams 19
6.3 Interface Classes 19
6.3.1 Trust and Security Management Interface Classes 19
6.3.1.1 Interface Class IpClientAPILevelAuthentication 20
6.3.1.2 Interface Class IpClientAccess 21
6.3.1.3 Interface Class IpInitial 22
6.3.1.4 Interface Class IpAuthentication 23
6.3.1.5 Interface Class IpAPILevelAuthentication 24
6.3.1.6 Interface Class IpAccess 26
6.4 State Transition Diagrams 28
6.4.1 Trust and Security Management State Transition Diagrams 28
6.4.1.1 State Transition Diagrams for IpInitial 28
6.4.1.2 State Transition Diagrams for IpAPILevelAuthentication 29
6.4.1.2.1 Idle State 29
6.4.1.2.2 Selecting Method State 29
6.4.1.2.3 Authenticating Client State 30
6.4.1.2.4 Client Authenticated State 30
6.4.1.3 State Transition Diagrams for IpAccess 30
6.4.1.3.1 Active State 31
7 Framework-to-Application API 31
7.1 Sequence Diagrams 31
7.1.1 Event Notification Sequence Diagrams 31
7.1.1.1 Enable Event Notification 31
7.1.2 Integrity Management Sequence Diagrams 33

7.1.2.1 Load Management: Suspend/resume notification from application 33
7.1.2.2 Load Management: Framework queries load statistics 34
7.1.2.3 Load Management: Framework callback registration and Application load control 35
7.1.2.4 Load Management: Application reports current load condition 36
7.1.2.5 Load Management: Application queries load statistics 36
7.1.2.6 Load Management: Application callback registration and load control 37
7.1.2.7 Heartbeat Management: Start/perform/end heartbeat supervision of the application 38
7.1.2.8 Fault Management: Framework detects a Service failure 39
7.1.2.9 Fault Management: Application requests a Framework activity test 40
7.1.3 Service Discovery Sequence Diagrams 40
7.1.3.1 Service Discovery 40
7.1.4 Service Agreement Management Sequence Diagrams 42
7.1.4.1 Service Selection 42
7.2 Class Diagrams 44
7.3 Interface Classes 47
7.3.1 Service Discovery Interface Classes 47
7.3.1.1 Interface Class IpServiceDiscovery 47
7.3.2 Service Agreement Management Interface Classes 50
7.3.2.1 Interface Class IpAppServiceAgreementManagement 50
7.3.2.2 Interface Class IpServiceAgreementManagement 52
7.3.3 Integrity Management Interface Classes 54
7.3.3.1 Interface Class IpAppFaultManager 54
7.3.3.2 Interface Class IpFaultManager 58
7.3.3.3 Interface Class IpAppHeartBeatMgmt 61
7.3.3.4 Interface Class IpAppHeartBeat 62
7.3.3.5 Interface Class IpHeartBeatMgmt 62
7.3.3.6 Interface Class IpHeartBeat 63
7.3.3.7 Interface Class IpAppLoadManager 64
7.3.3.8 Interface Class IpLoadManager 66
7.3.3.9 Interface Class IpOAM 69
7.3.3.10 Interface Class IpAppOAM 70
7.3.4 Event Notification Interface Classes 71
7.3.4.1 Interface Class IpAppEventNotification 71
7.3.4.2 Interface Class IpEventNotification 71
7.4 State Transition Diagrams 72
7.4.1 Service Discovery State Transition Diagrams 73
7.4.1.1 State Transition Diagrams for IpServiceDiscovery 73
7.4.1.1.1 Active State 73
7.4.2 Service Agreement Management State Transition Diagrams 73
7.4.3 Integrity Management State Transition Diagrams 74
7.4.3.1 State Transition Diagrams for IpLoadManager 74
7.4.3.1.1 Idle State 74
7.4.3.1.2 Notification Suspended State 74
7.4.3.1.3 Active State 74
7.4.3.2 State Transition Diagrams for LoadManagerInternal 75
7.4.3.2.1 Normal load State 75
7.4.3.2.2 Application Overload State 75
7.4.3.2.3 Internal overload State 75
7.4.3.2.4 Internal and Application Overload State 75
7.4.3.3 State Transition Diagrams for IpOAM 76
7.4.3.3.1 Active State 76
7.4.3.4 State Transition Diagrams for IpFaultManager 77
7.4.3.4.1 Framework Active State 77
7.4.3.4.2 Framework Faulty State 77
7.4.3.4.3 Framework Activity Test State 77
7.4.3.4.4 Service Activity Test State 77
7.4.4 Event Notification State Transition Diagrams 78
7.4.4.1 State Transition Diagrams for IpEventNotification 78

8 Framework-to-Enterprise Operator API 78
8.1 Sequence Diagrams 82
8.1.1 Service Subscription Sequence Diagrams 82

8.1.1.1 Service Discovery and Subscription Scenario 82
8.1.1.2 Enterprise Operator and Client Application Subscription Management Sequence Diagram 84
8.2 Class Diagrams 87
8.3 Interface Classes 88
8.3.1 Service Subscription Interface Classes 88
8.3.1.1 Interface Class IpClientAppManagement 88
8.3.1.2 Interface Class IpClientAppInfoQuery 92
8.3.1.3 Interface Class IpServiceProfileManagement 94
8.3.1.4 Interface Class IpServiceProfileInfoQuery 96
8.3.1.5 Interface Class IpServiceContractManagement 97
8.3.1.6 Interface Class IpServiceContractInfoQuery 99
8.3.1.7 Interface Class IpEntOpAccountManagement 101
8.3.1.8 Interface Class IpEntOpAccountInfoQuery 102
8.4 State Transition Diagrams 102
8.4.1 Service Subscription State Transition Diagrams 102

9 Framework-to-Service API 103
9.1 Sequence Diagrams 103
9.1.1 Service Discovery Sequence Diagrams 103
9.1.2 Service Registration Sequence Diagrams 103
9.1.2.1 New SCF Registration 103
9.1.3 Service Instance Lifecycle Manager Sequence Diagrams 104
9.1.3.1 Sign Service Agreement 104
9.1.4 Integrity Management Sequence Diagrams 106
9.1.4.1 Load Management: Service callback registration and load control 106
9.1.4.2 Load Management: Framework callback registration and service load control 107
9.1.4.3 Load Management: Client and Service Load Balancing 108
9.1.4.4 Heartbeat Management: Start/perform/end heartbeat supervision of the service 108
9.1.4.5 Fault Management: Service requests Framework activity test 109
9.1.4.6 Fault Management: Service requests Application activity test 110
9.1.4.7 Fault Management: Application requests Service activity test 111
9.1.4.8 Fault Management: Application detects service is unavailable 112
9.1.5 Event Notification Sequence Diagrams 112
9.2 Class Diagrams 113
9.3 Interface Classes 115
9.3.1 Service Registration Interface Classes 115
9.3.1.1 Interface Class IpFwServiceRegistration 116
9.3.2 Service Instance Lifecycle Manager Interface Classes 119
9.3.2.1 Interface Class IpServiceInstanceLifecycleManager 119
9.3.3 Service Discovery Interface Classes 120
9.3.3.1 Interface Class IpFwServiceDiscovery 120
9.3.4 Integrity Management Interface Classes 123
9.3.4.1 Interface Class IpFwFaultManager 123
9.3.4.2 Interface Class IpSvcFaultManager 126
9.3.4.3 Interface Class IpFwHeartBeatMgmt 129
9.3.4.4 Interface Class IpFwHeartBeat 131
9.3.4.5 Interface Class IpSvcHeartBeatMgmt 131
9.3.4.6 Interface Class IpSvcHeartBeat 132
9.3.4.7 Interface Class IpFwLoadManager 133
9.3.4.8 Interface Class IpSvcLoadManager 136
9.3.4.9 Interface Class IpFwOAM 139
9.3.4.10 Interface Class IpSvcOAM 139
9.3.5 Event Notification Interface Classes 140
9.3.5.1 Interface Class IpFwEventNotification 140
9.3.5.2 Interface Class IpSvcEventNotification 141
9.4 State Transition Diagrams 142
9.4.1 Service Registration State Transition Diagrams 143
9.4.1.1 State Transition Diagrams for IpFwServiceRegistration 143
9.4.1.1.1 SCF Registered State 143
9.4.1.1.2 SCF Announced State 143
9.4.2 Service Instance Lifecycle Manager State Transition Diagrams 143
9.4.3 Service Discovery State Transition Diagrams 144

9.4.4 Integrity Management State Transition Diagrams 144
9.4.4.1 State Transition Diagrams for IpFwLoadManager 144
9.4.4.1.1 Idle State 144
9.4.4.1.2 Notification Suspended State 144
9.4.4.1.3 Active State 145
9.4.5 Event Notification State Transition Diagrams 145

10 Service Properties 145
10.1 Service Property Types 145
10.2 General Service Properties 147
10.2.1 Service Name 147
10.2.2 Service Version 147
10.2.3 Service Instance ID 147
10.2.4 Service Instance Description 147
10.2.5 Product Name 147
10.2.6 Product Version 148
10.2.7 Supported Interfaces 148
10.2.8 Operation Set 148

11 Data Definitions 148
11.1 Common Framework Data Definitions 148
11.1.1 TpClientAppID 148
11.1.2 TpClientAppIDList 148
11.1.3 TpDomainID 148
11.1.4 TpDomainIDType 149
11.1.5 TpEntOpID 149
11.1.6 TpPropertyName 149
11.1.7 TpPropertyValue 149
11.1.8 TpProperty 149
11.1.9 TpPropertyList 149
11.1.10 TpEntOpIDList 149
11.1.11 TpFwID 150
11.1.12 TpService 150
11.1.13 TpServiceList 150
11.1.14 TpServiceDescription 150
11.1.15 TpServiceID 150
11.1.16 TpServiceIDList 150
11.1.17 TpServiceInstanceID 150
11.1.18 TpServiceSpecString 150
11.1.19 TpServiceTypeProperty 151
11.1.20 TpServiceTypePropertyList 151
11.1.21 TpServiceTypePropertyMode 151
11.1.22 TpServicePropertyTypeName 151
11.1.23 TpServicePropertyName 151
11.1.24 TpServicePropertyNameList 151
11.1.25 TpServicePropertyValue 151
11.1.26 TpServicePropertyValueList 151
11.1.27 TpServiceProperty 152
11.1.28 TpServicePropertyList 152
11.1.29 TpServiceSupplierID 152
11.1.30 TpServiceTypeDescription 152
11.1.31 TpServiceTypeName 152
11.1.32 TpServiceTypeNameList 153
11.1.33 TpSubjectType 153
11.2 Event Notification Data Definitions 153
11.2.1 TpFwEventName 153
11.2.2 TpFwEventCriteria 153
11.2.3 TpFwEventInfo 153
11.3 Trust and Security Management Data Definitions 154
11.3.1 TpAccessType 154
11.3.2 TpAuthType 154
11.3.3 TpEncryptionCapability 154

11.3.4 TpEncryptionCapabilityList 155
11.3.5 TpEndAccessProperties 155
11.3.6 TpAuthDomain 155
11.3.7 TpInterfaceName 155
11.3.8 TpInterfaceNameList 155
11.3.9 TpServiceToken 156
11.3.10 TpSignatureAndServiceMgr 156
11.3.11 TpSigningAlgorithm 156
11.4 Integrity Management Data Definitions 156
11.4.1 TpActivityTestRes 156
11.4.2 TpFaultStatsRecord 156
11.4.3 TpFaultStats 157
11.4.4 TpFaultStatisticsError 157
11.4.5 TpFaultStatsSet 157
11.4.6 TpActivityTestID 157
11.4.7 TpInterfaceFault 157
11.4.8 TpSvcUnavailReason 157
11.4.9 TpFwUnavailReason 158
11.4.10 TpLoadLevel 158
11.4.11 TpLoadThreshold 158
11.4.12 TpLoadInitVal 158
11.4.13 TpLoadPolicy 158
11.4.14 TpLoadStatistic 159
11.4.15 TpLoadStatisticList 159
11.4.16 TpLoadStatisticData 159
11.4.17 TpLoadStatisticEntityID 159
11.4.18 TpLoadStatisticEntityType 159
11.4.19 TpLoadStatisticInfo 160
11.4.20 TpLoadStatisticInfoType 160
11.4.21 TpLoadStatisticError 160
11.5 Service Subscription Data Definitions 160
11.5.1 TpPropertyName 160
11.5.2 TpPropertyValue 160
11.5.3 TpProperty 160
11.5.4 TpPropertyList 161
11.5.5 TpEntOpProperties 161
11.5.6 TpEntOp 161
11.5.7 TpServiceContractID 161
11.5.8 TpServiceContractIDList 161
11.5.9 TpPersonName 161
11.5.10 TpPostalAddress 161
11.5.11 TpTelephoneNumber 161
11.5.12 TpEmail 161
11.5.13 TpHomePage 161
11.5.14 TpPersonProperties 161
11.5.15 TpPerson 162
11.5.16 TpServiceStartDate 162
11.5.17 TpServiceEndDate 162
11.5.18 TpServiceRequestor 162
11.5.19 TpBillingContact 162
11.5.20 TpServiceSubscriptionProperties 162
11.5.21 TpServiceContract 162
11.5.22 TpServiceContractDescription 163
11.5.23 TpClientAppProperties 163
11.5.24 TpClientAppDescription 163
11.5.25 TpSagID 163
11.5.26 TpSagIDList 163
11.5.27 TpSagDescription 163
11.5.28 TpSag 164
11.5.29 TpServiceProfileID 164
11.5.30 TpServiceProfileIDList 164
11.5.31 TpServiceProfile 164

11.5.32 TpServiceProfileDescription....164
12 Exception Classes....165
Annex A (normative): OMG IDL description of Framework....166
Annex B (informative): Contents of 3GPP OSA R4 Framework....167
Annex C (informative): Record of changes....168
C.1 Interfaces....168
C.1.1 New....168
C.1.2 Deprecated....168
C.1.3 Removed....168
C.2 Methods....169
C.2.1 New....169
C.2.2 Deprecated....169
C.2.3 Modified....169
C.2.4 Removed....169
C.3 Data Definitions....170
C.3.1 New....170
C.3.2 Modified....170
C.3.3 Removed....170
C.4 Service Properties....170
C.4.1 New....170
C.4.2 Deprecated....171
C.4.3 Modified....171
C.4.4 Removed....171
C.5 Exceptions....171
C.5.1 New....171
C.5.2 Modified....172
C.5.3 Removed....172
C.6 Others....172
History....173

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 3 of a multi-part deliverable covering Open Service Access (OSA); Application Programming Interface (API), as identified below. The API specification (ES 201 915) 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";

Part 10: "Connectivity Manager SCF";

Part 11: "Account Management SCF";

Part 12: "Charging SCF".

The present document has been defined jointly between ETSI, The Parlay Group (http://www.parlay.org) and the 3GPP, in co-operation with a number of JAIN™ Community (http://www.java.sun.com/products/jain) member companies.

The present document forms part of the Parlay 3.4 set of specifications.

A subset of the present document is in 3GPP TS 29.198-3 4.9.0 (Release 4).

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 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).

2 References

The references listed in clause 2 of ES 201 915-1 contain provisions which, through reference in this text, constitute provisions of the present document.

ETSI ES 201 915-1: "Open Service Access (OSA); Application Programming Interface (API);
Part 1: Overview (Parlay 3)".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ES 201 915-1 apply.

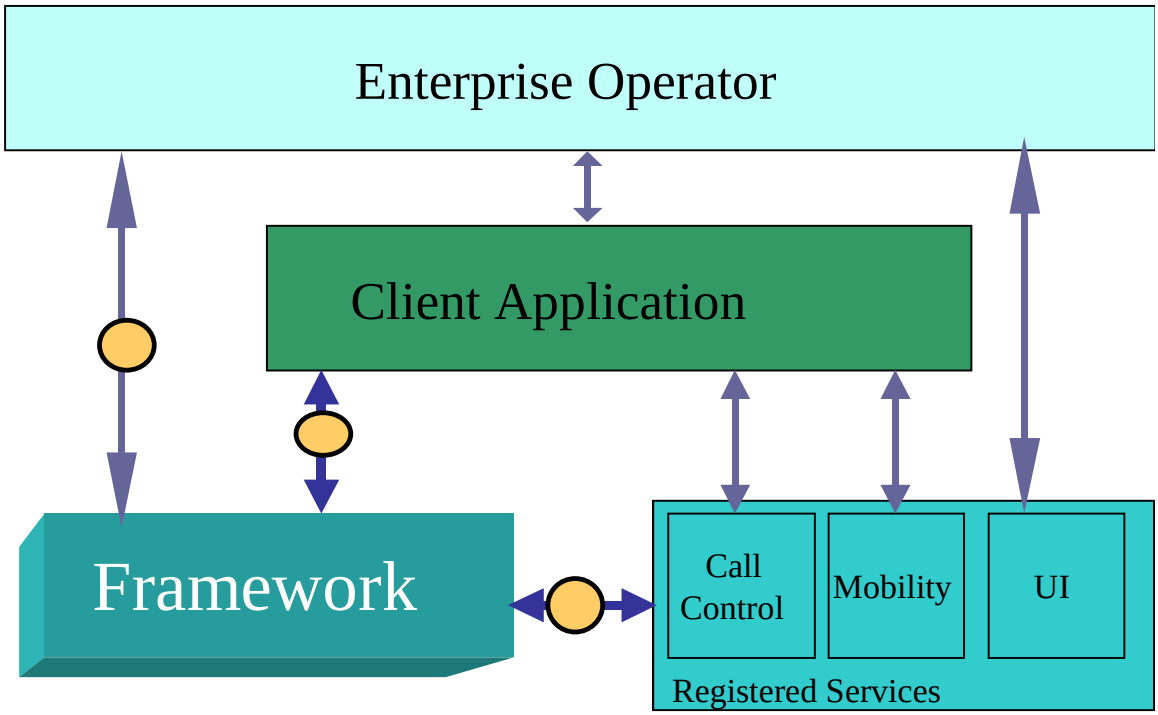
3.2 Abbreviations

For the purposes of the present document, the abbreviations defined in ES 201 915-1 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, between the Network Service Capability Server (SCS) and the Framework, and between the Enterprise Operator and the Framework (these interfaces are represented by the yellow circles in the diagram below). The description of the Framework in the present document separates the interfaces into these three distinct sets: Framework to Application interfaces, Framework to Enterprise Operator interfaces and Framework to Service interfaces.



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

Basic mechanisms between Application and Framework:

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

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

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

- **Establishment of service agreement:** Before any application can interact with a network service capability feature, a service agreement must 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 service capability feature.

- **Access to network service capability features:** The framework must provide access control functions to authorise the access to service capability features 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:

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

Basic mechanism between Framework and Enterprise Operator:

- **Service Subscription function:** This function represents a contractual agreement between the Enterprise Operator and the Framework. In this subscription business model, the enterprise operators act in the role of *subscriber/customer* of services and the client applications act in the role of *users or consumers* of *services*. The framework itself acts in the role of *retailer* of services.

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

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

4.1 General requirements on support of methods

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

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

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

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 callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>.

5.1.2 Method descriptions

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

5.1.3 Parameter descriptions

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

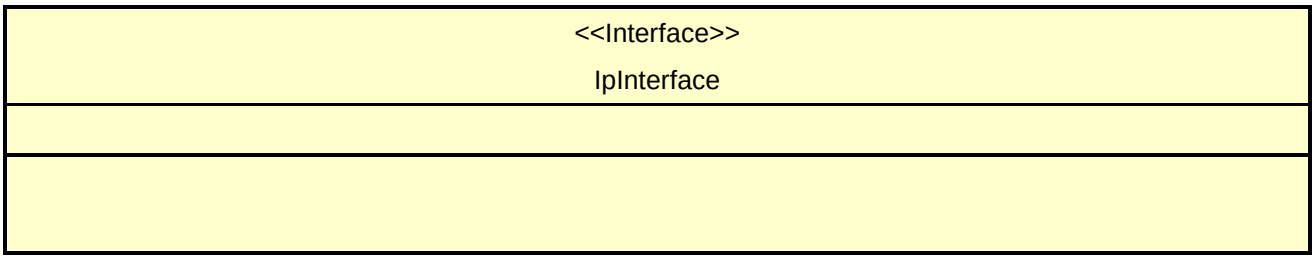
5.1.4 State Model

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

5.2 Base Interface

5.2.1 Interface Class IpInterface

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



5.3 Service Interfaces

5.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

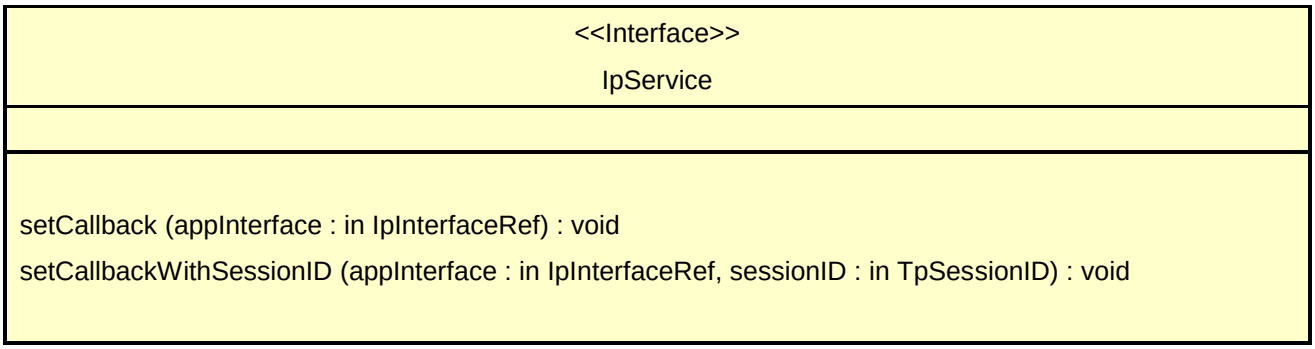
The interfaces that are implemented by the services are denoted as "Service Interface". The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as "Application Interface".

5.4 Generic Service Interface

5.4.1 Interface Class IpService

Inherits from: IpInterface.

All service interfaces inherit from the following interface.



Method

setCallback()

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

Parameters

appInterface : in IpInterfaceRef

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

Raises

TpCommonExceptions, P_INVALID_INTERFACE_TYPE

Method

setCallbackWithSessionID()

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

Parameters

appInterface : in IpInterfaceRef

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

sessionID : in TpSessionID

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

Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE

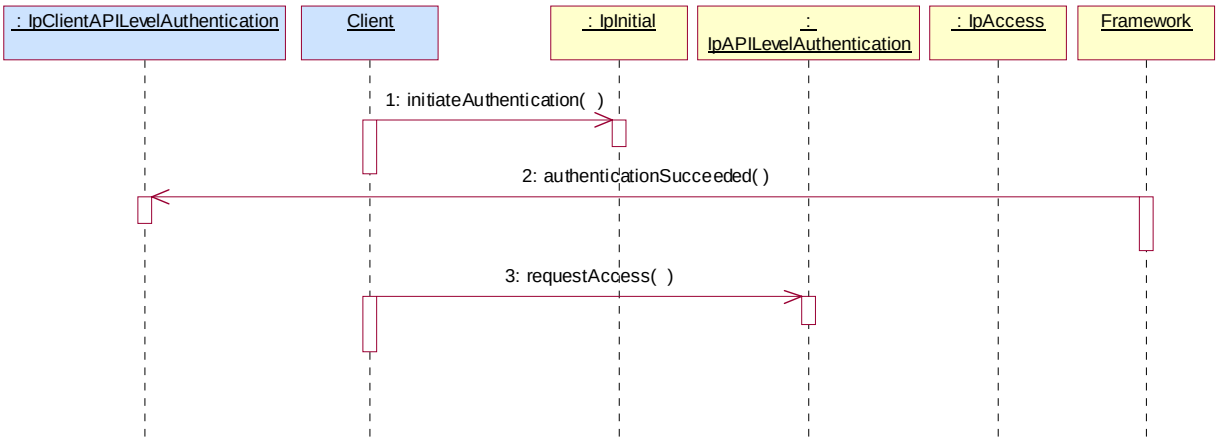
6 Framework Access Session API

6.1 Sequence Diagrams

6.1.1 Trust and Security Management Sequence Diagrams

6.1.1.1 Initial Access for trusted parties

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



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

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

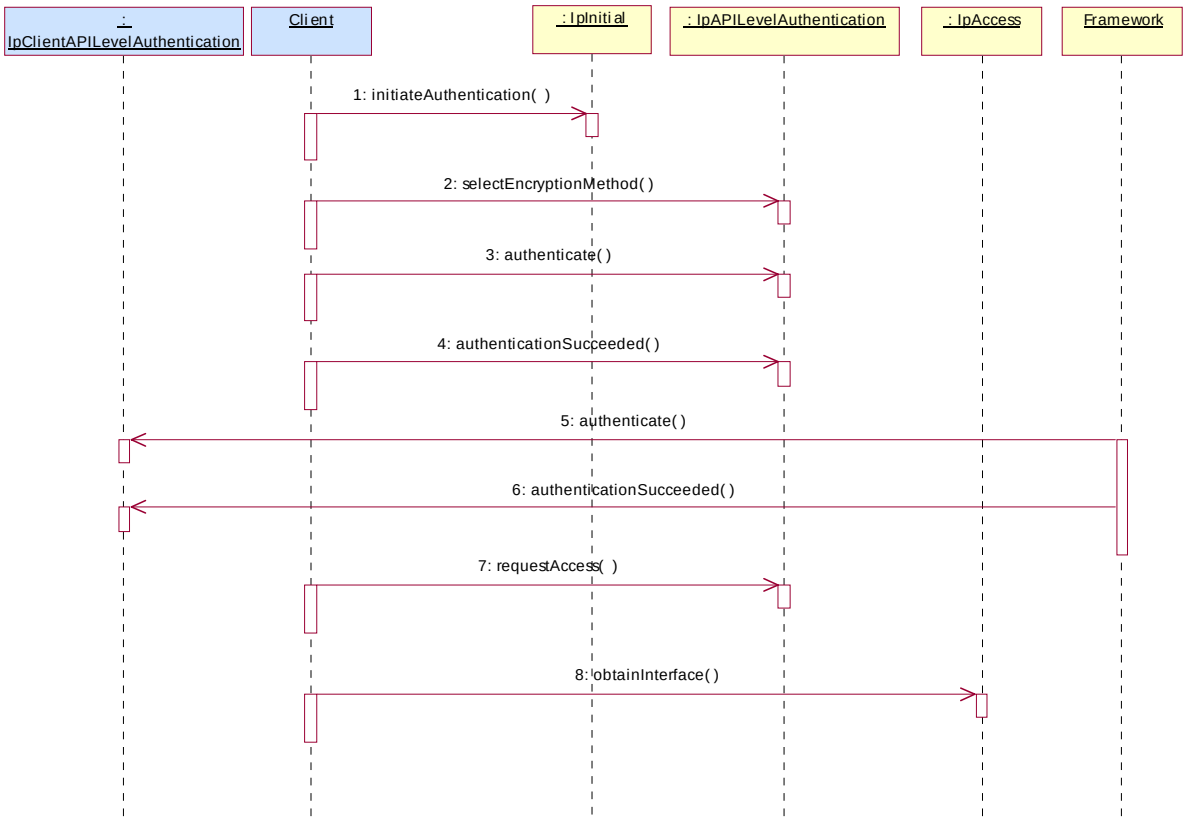
3: The Client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

6.1.1.2 Initial Access

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

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

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



1: Initiate Authentication

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

2: Select Encryption Method

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

3: Authenticate

4: The client provides an indication if authentication succeeded.

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

6: The Framework provides an indication if authentication succeeded.

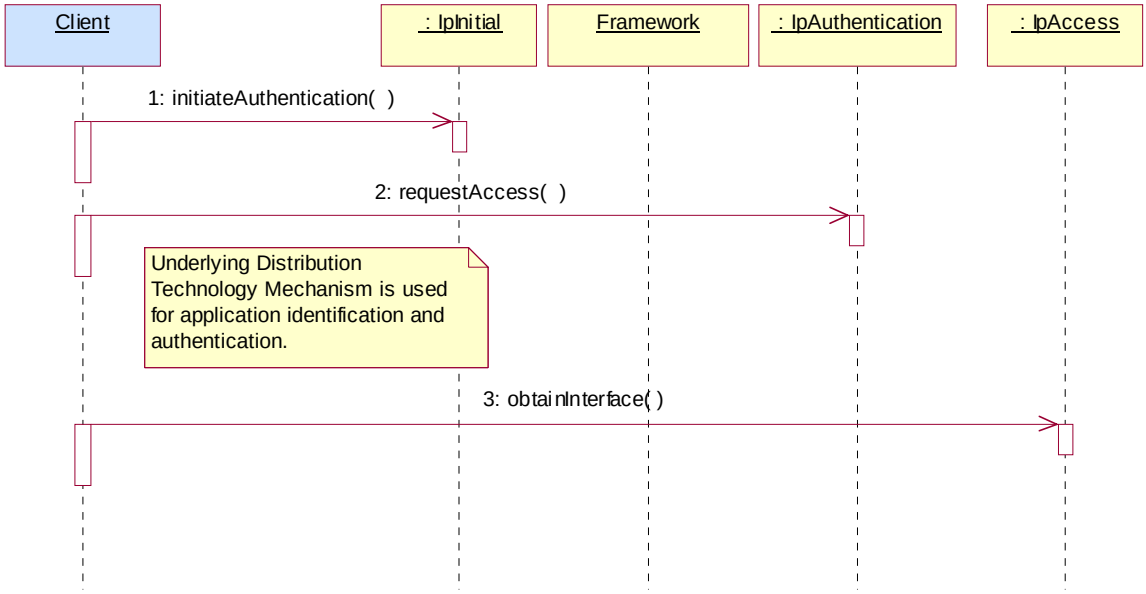
7: Request Access

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

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

6.1.1.3 Authentication

This sequence diagram illustrates the two-way mechanism by which the client and the framework mutually authenticate one another using an underlying distribution technology mechanism.



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

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

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

6.1.1.4 API Level Authentication

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

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

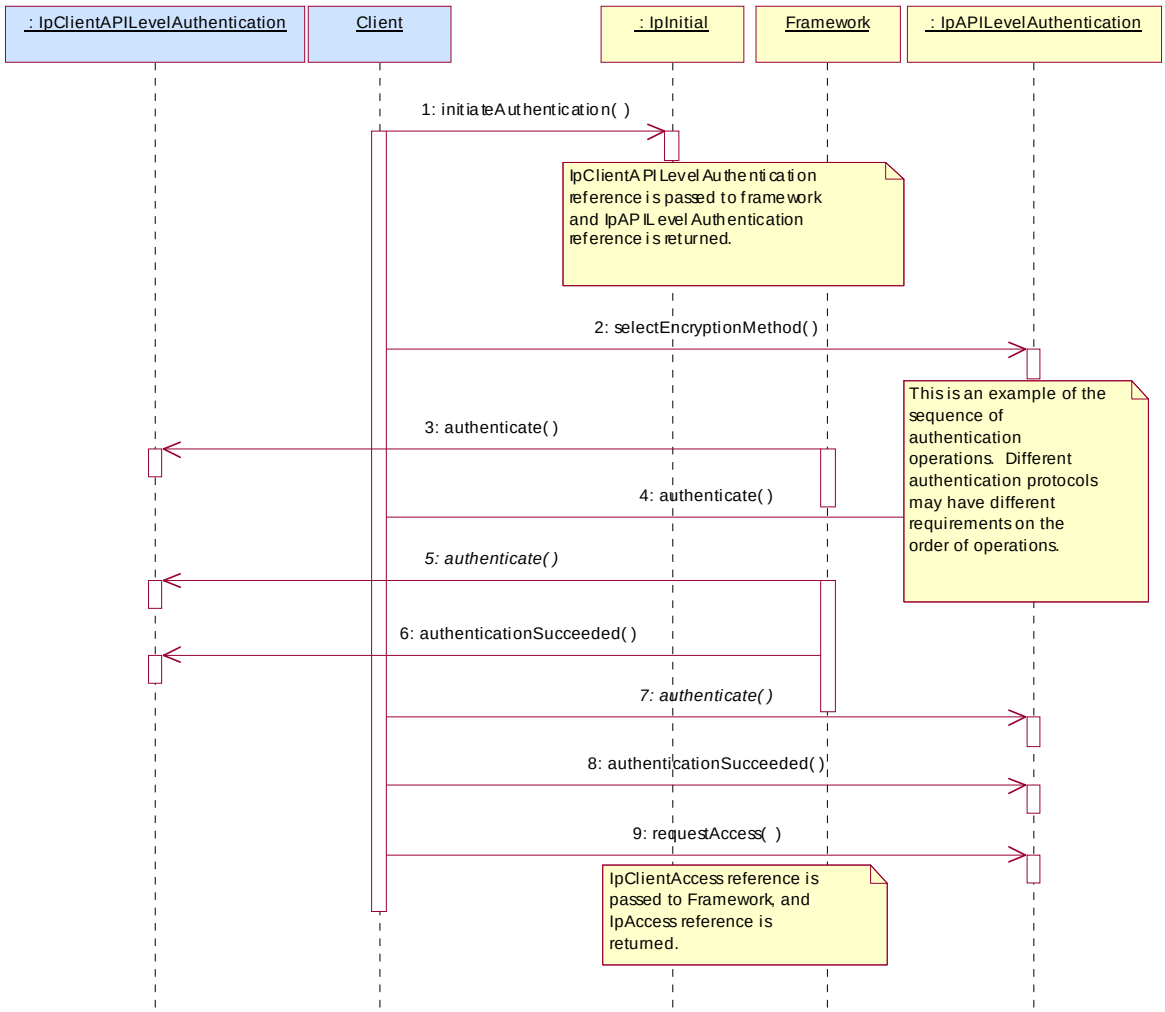
The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

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

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

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

Note that at any point during the access session, either side can request re-authentication. Re-authentication does not have to be mutual.



6.2 Class Diagrams

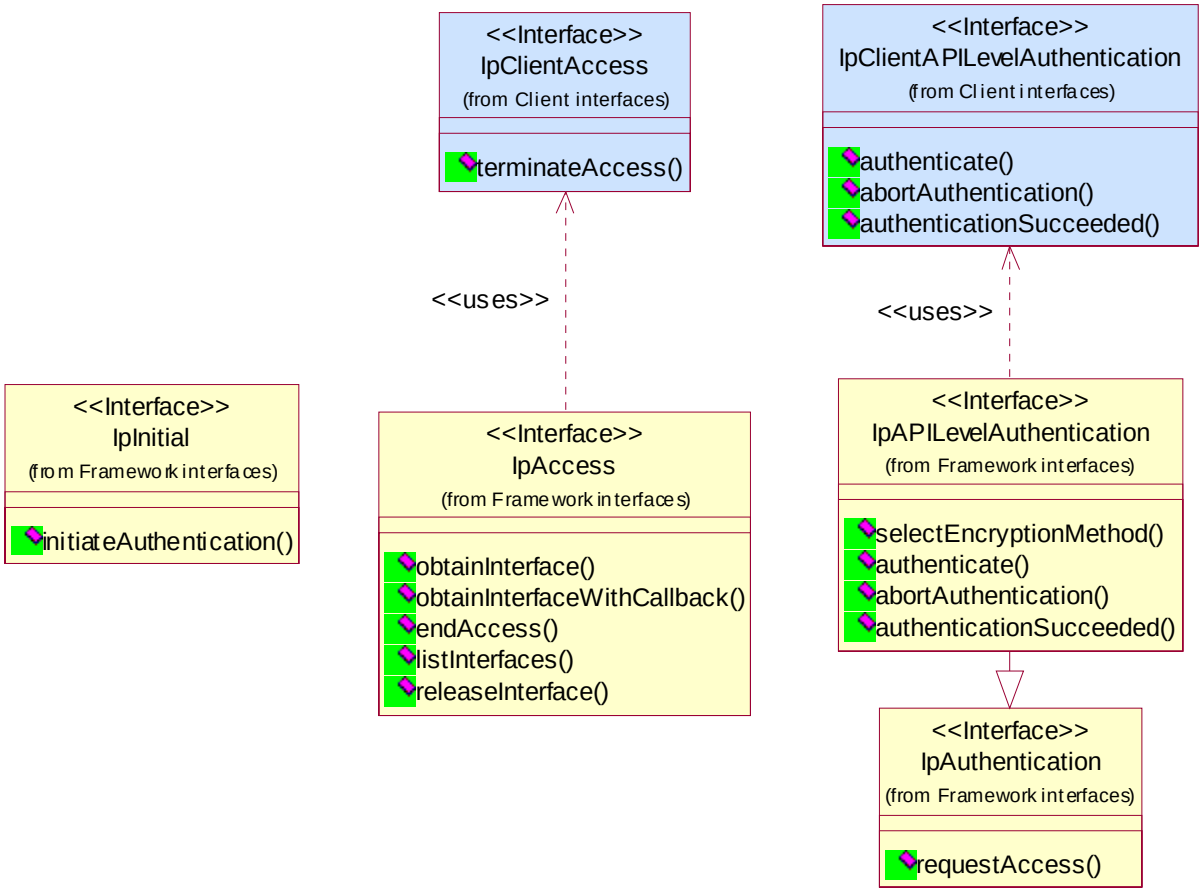


Figure 1: Trust and Security Management Package Overview

6.3 Interface Classes

6.3.1 Trust and Security Management Interface Classes

The Trust and Security Management Interfaces provide:

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

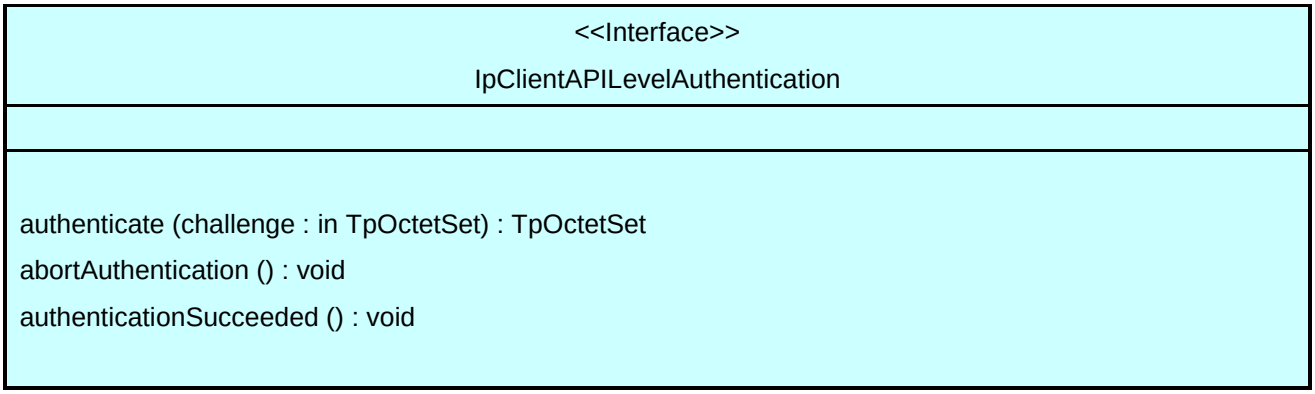
The process by which the client accesses the Framework provider has been separated into 3 stages, each supported by a different Framework interface:

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

6.3.1.1 Interface Class IpClientAPILevelAuthentication

Inherits from: IpInterface.

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



Method

`authenticate()`

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

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

Parameters

`challenge : in TpOctetSet`

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

Returns

`TpOctetSet`

Method

`abortAuthentication()`

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

Parameters

No Parameters were identified for this method.

Method
authenticationSucceeded()

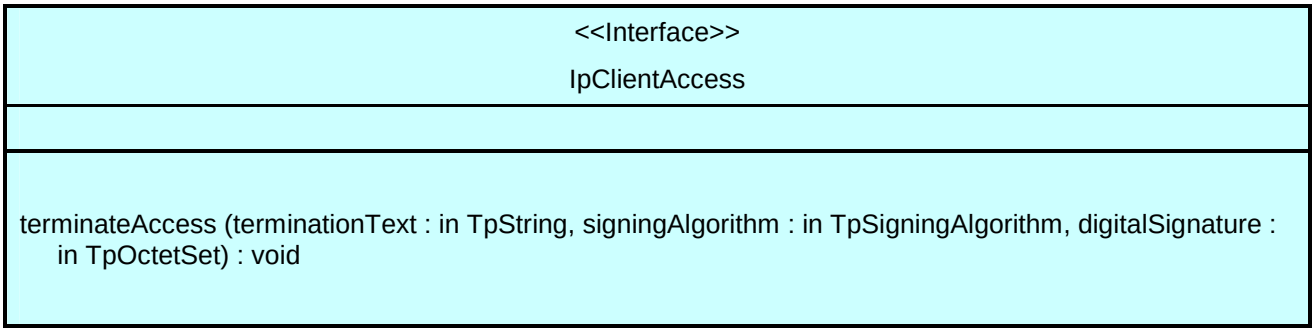
The Framework uses this method to inform the client of the success of the authentication attempt.

Parameters
No Parameters were identified for this method.

6.3.1.2 Interface Class IpClientAccess

Inherits from: IpInterface.

IpClientAccess interface is offered by the client to the framework to allow it to initiate interactions during the access session. This interface and the terminateAccess() method shall be implemented by a client.



Method
terminateAccess()

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

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

Parameters

terminationText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

digitalSignature : in TpOctetSet

This is a signed version of a hash of the termination text. If no signing algorithm is used, the digitalSignature is the octet sequence of the termination text itself. The framework uses this to confirm its identity to the client. The client can check that the terminationText has been signed by the framework. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

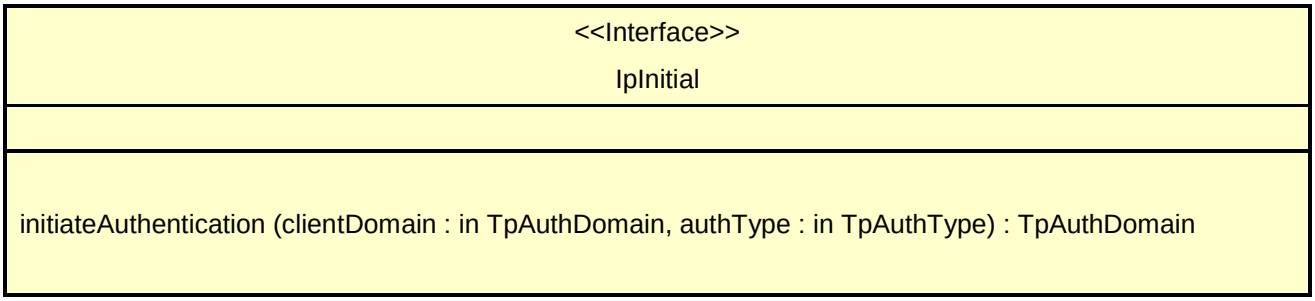
Raises

TpCommonExceptions, P_INVALID_SIGNING_ALGORITHM, P_INVALID_SIGNATURE

6.3.1.3 Interface Class IpInitial

Inherits from: IpInterface.

The Initial Framework interface is used by the client to initiate the mutual authentication with the Framework. This interface and the initiateAuthentication() method shall be implemented by a Framework.



Method

initiateAuthentication()

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

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

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

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

Parameters

clientDomain : in TpAuthDomain

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

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

The domainID parameter is an identifier either for a client application (i.e. TpClientAppID) or for an enterprise operator (i.e. TpEntOpID), or for an instance of a service for which a client application has signed a service agreement (i.e. TpServiceInstanceID), or for a service supplier (i.e. TpServiceSupplierID). It is used to identify the client domain to the framework, (see authenticate() on IpAPILevelAuthentication). If the framework does not recognise the domainID, the framework returns an error code (P_INVALID_DOMAIN_ID).
The authInterface parameter is a reference to call the authentication interface of the client. The type of this interface is defined by the authType parameter. If the interface reference is not of the correct type, the framework returns an error code (P_INVALID_INTERFACE_TYPE).

authType : in TpAuthType

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

Returns

TpAuthDomain

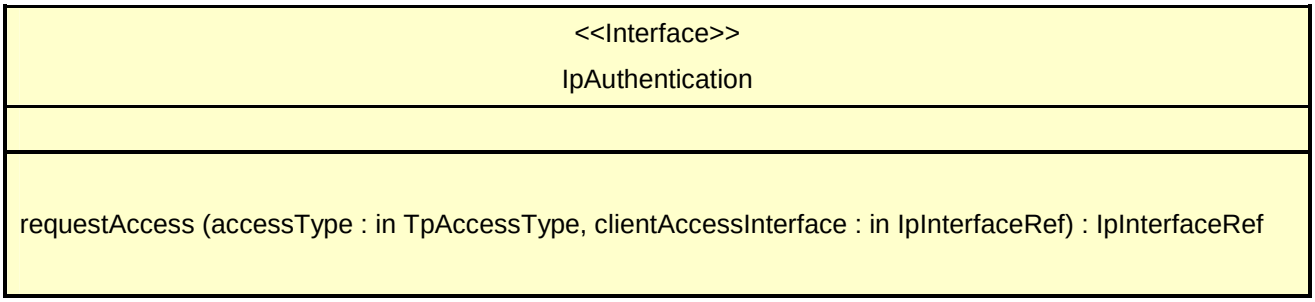
Raises

TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE

6.3.1.4 Interface Class IpAuthentication

Inherits from: IpInterface.

The Authentication Framework interface is used by client to request access to other interfaces supported by the Framework. The mutual authentication process should in this case be done with some underlying distribution technology authentication mechanism, e.g. CORBA Security.
One of IpAuthentication or IpAPILevelAuthentication interfaces shall be implemented by a Framework. The requestAccess() method shall be implemented in each.



Method

requestAccess()

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

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

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

Parameters

accessType : in TpAccessType

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

clientAccessInterface : in IpInterfaceRef

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

Returns

IpInterfaceRef

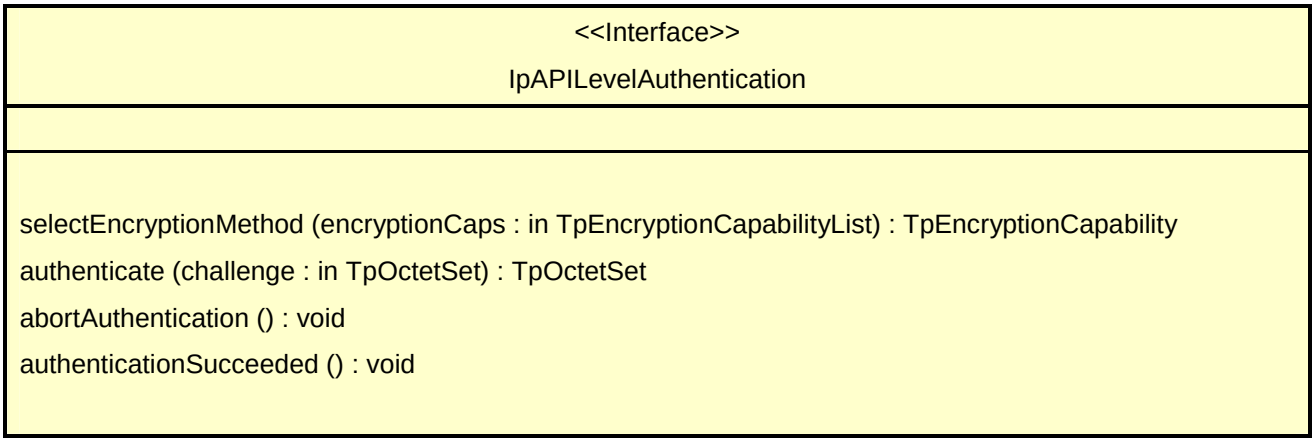
Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_ACCESS_TYPE, P_INVALID_INTERFACE_TYPE

6.3.1.5 Interface Class IpAPILevelAuthentication

Inherits from: IpAuthentication.

The API Level Authentication Framework interface is used by client to perform its part of the mutual authentication process with the Framework necessary to be allowed to use any of the other interfaces supported by the Framework. If the IpAPILevelAuthentication interface is implemented by a Framework, the selectEncryptionMethod(), authenticate(), abortAuthentication() and authenticationSucceeded() methods shall be implemented. IpAPILevelAuthentication inherits the requirements of IpAuthentication, therefore requestAccess() shall be implemented.



Method

selectEncryptionMethod()

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

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

Parameters

encryptionCaps : in TpEncryptionCapabilityList

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

Returns

TpEncryptionCapability

Raises

TpCommonExceptions, P_ACCESS_DENIED,
P_NO_ACCEPTABLE_ENCRYPTION_CAPABILITY

Method
authenticate()

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

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

Parameters

challenge : in TpOctetSet

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

Returns

TpOctetSet

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method
abortAuthentication()

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

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method
authenticationSucceeded()

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

Parameters

No Parameters were identified for this method.

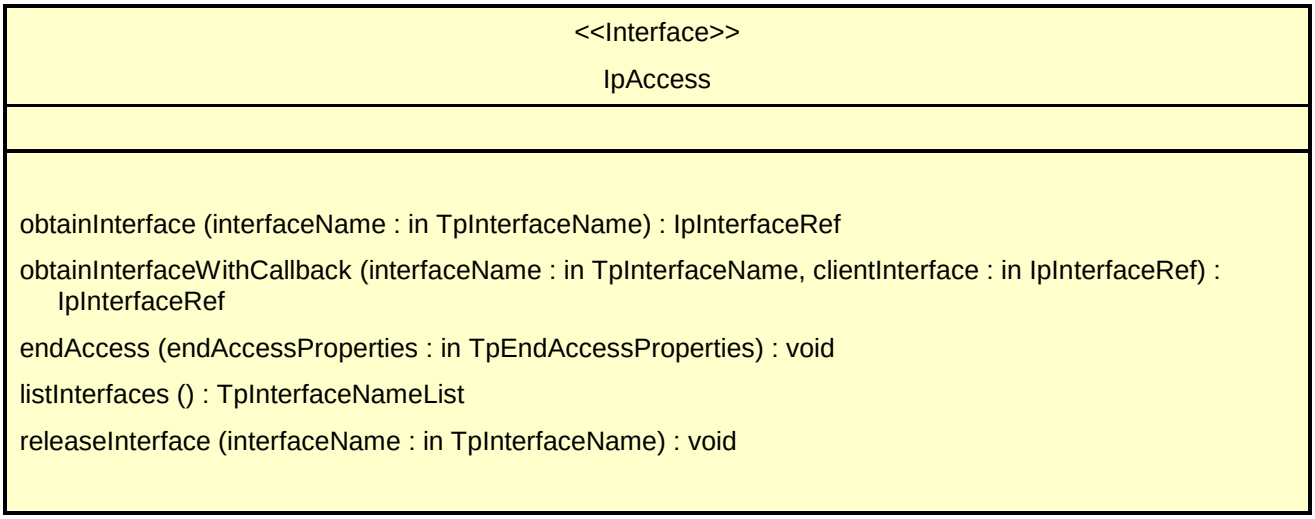
Raises

TpCommonExceptions, P_ACCESS_DENIED

6.3.1.6 Interface Class IpAccess

Inherits from: IpInterface.

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



Method

obtainInterface()

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

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

Parameters

interfaceName : in TpInterfaceName

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

Returns

IpInterfaceRef

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME

Method

obtainInterfaceWithCallback()

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

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

Parameters

interfaceName : in TpInterfaceName

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

clientInterface : in IpInterfaceRef

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

Returns

IpInterfaceRef

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME, P_INVALID_INTERFACE_TYPE

Method

endAccess()

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

Parameters

endAccessProperties : in TpEndAccessProperties

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

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_PROPERTY

Method

listInterfaces()

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

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

Parameters

No Parameters were identified for this method.

Returns

TpInterfaceNameList

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method
releaseInterface()

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

Parameters

interfaceName : in TpInterfaceName

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

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_INTERFACE_NAME

6.4 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 client are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the client also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

6.4.1 Trust and Security Management State Transition Diagrams

6.4.1.1 State Transition Diagrams for IpInitial

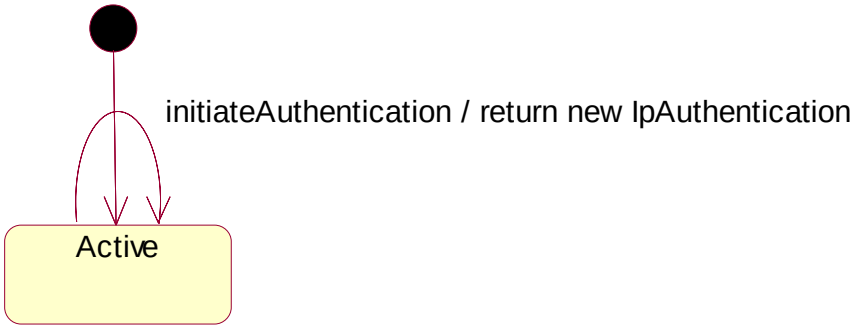


Figure 2: State Transition Diagram for IpInitial

6.4.1.2 State Transition Diagrams for IpAPILevelAuthentication

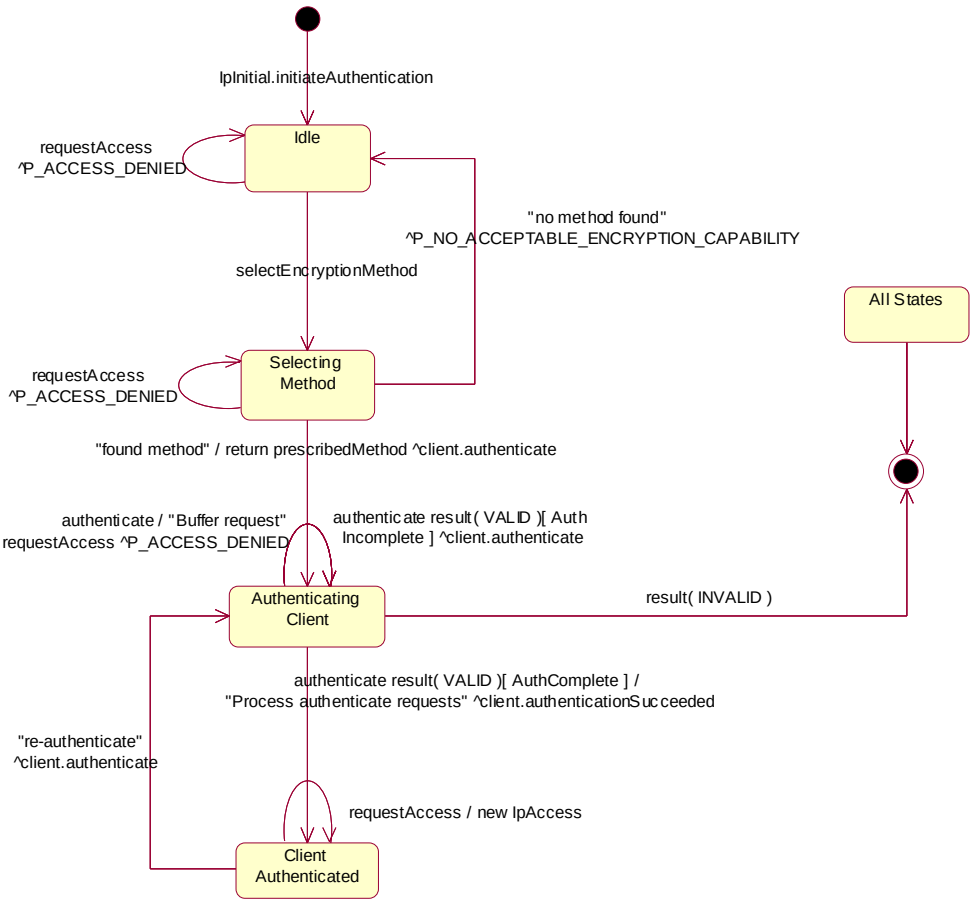


Figure 3: State Transition Diagram for IpAPILevelAuthentication

6.4.1.2.1 Idle State

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

6.4.1.2.2 Selecting Method State

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

6.4.1.2.3 Authenticating Client State

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

6.4.1.2.4 Client Authenticated State

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

6.4.1.3 State Transition Diagrams for IpAccess

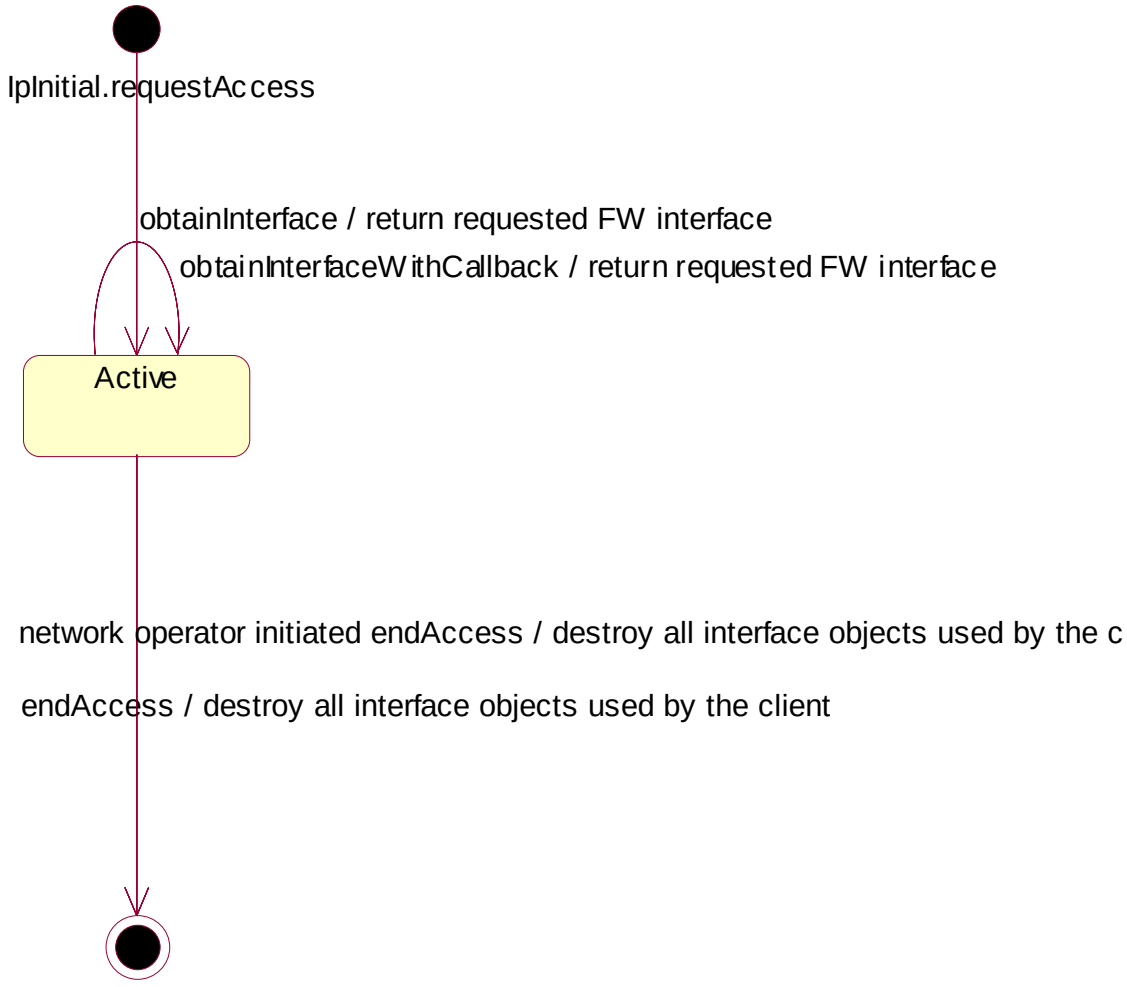


Figure 4: State Transition Diagram for IpAccess

6.4.1.3.1 Active State

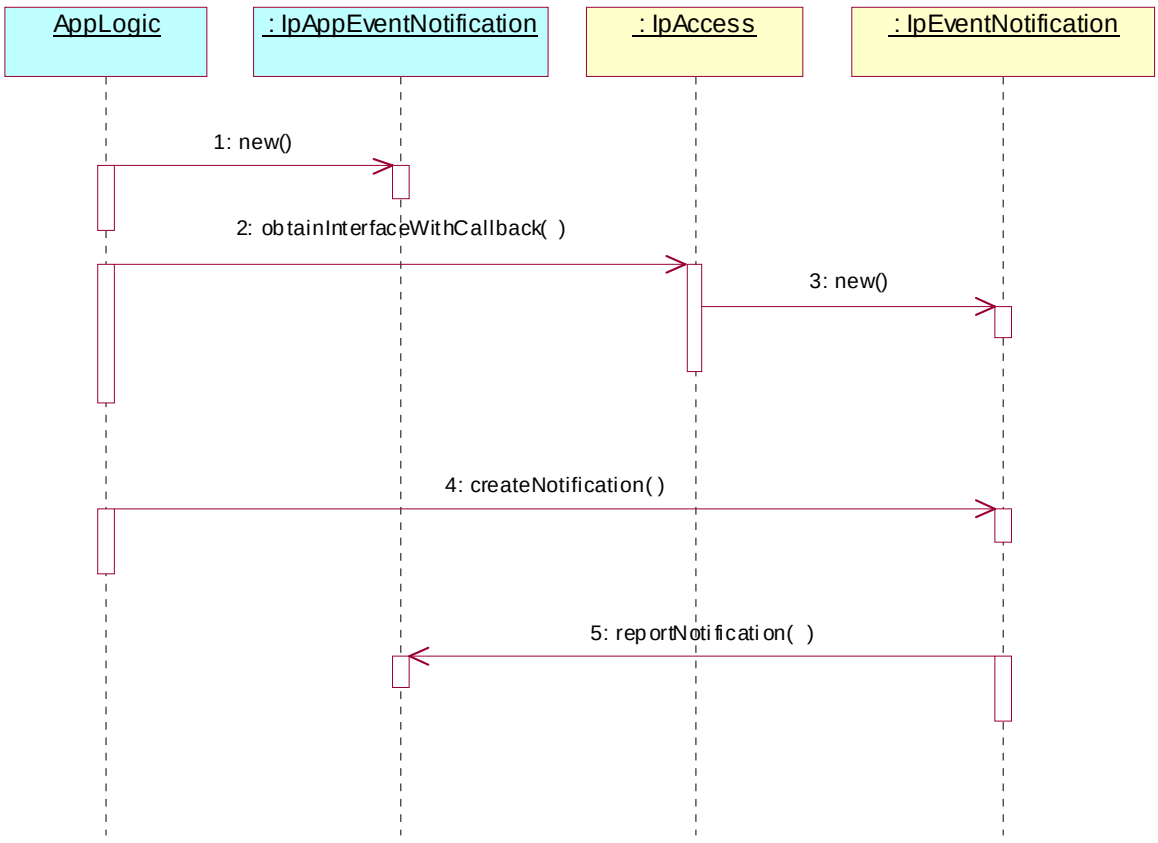
When the client requests access to the Framework on the IpInitial interface, an object implementing the IpAccess interface is created. The client can now request other Framework interfaces, including Service Discovery. When the client is no longer interested in using the interfaces it calls the endAccess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

7 Framework-to-Application API

7.1 Sequence Diagrams

7.1.1 Event Notification Sequence Diagrams

7.1.1.1 Enable Event Notification



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

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

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

4: createNotification(eventCriteria : in TpFwEventCriteria): TpAssignmentID.

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

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

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

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

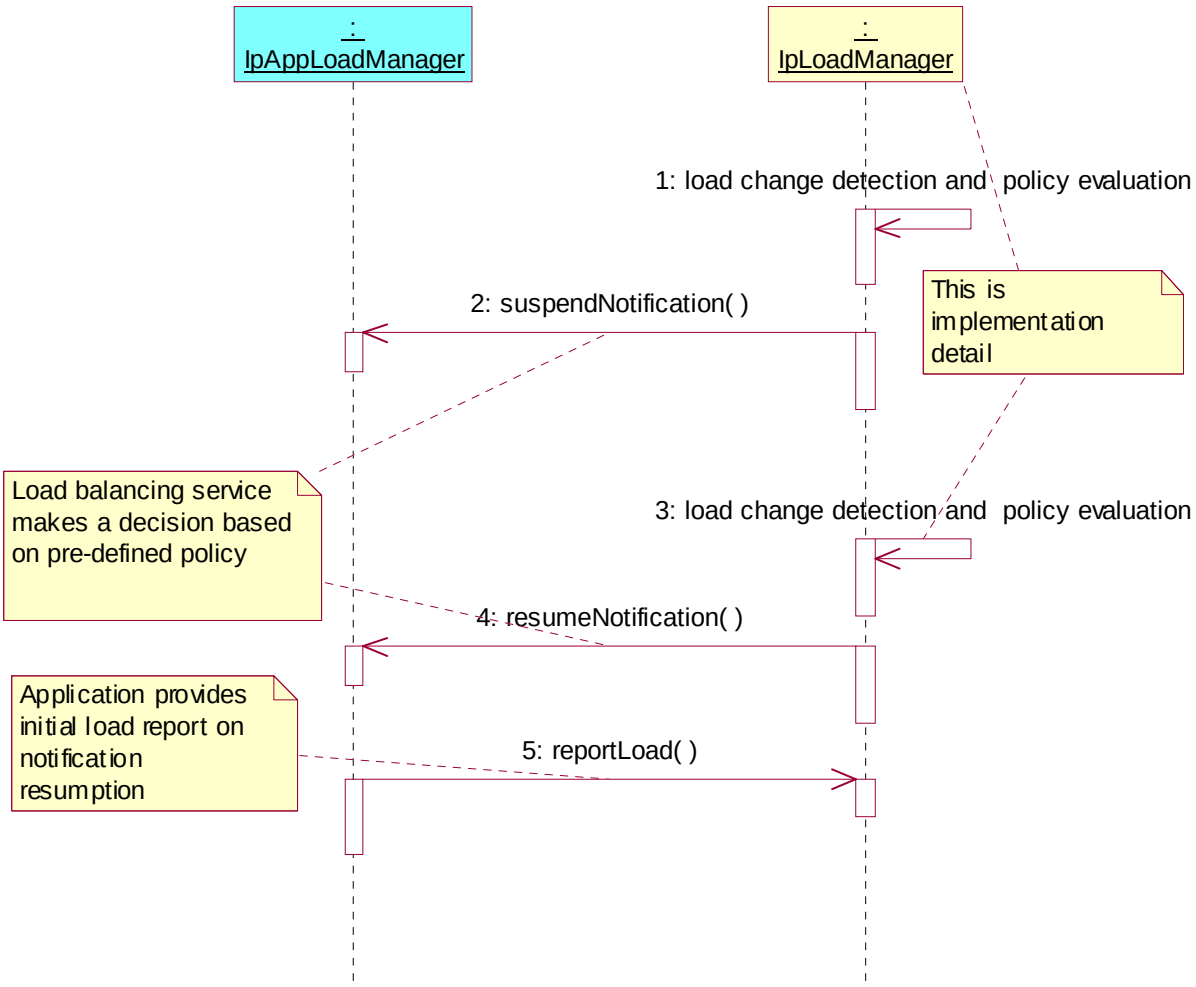
Alternatively, or additionally, the application can request to be informed of SCFs becoming unavailable.

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

7.1.2 Integrity Management Sequence Diagrams

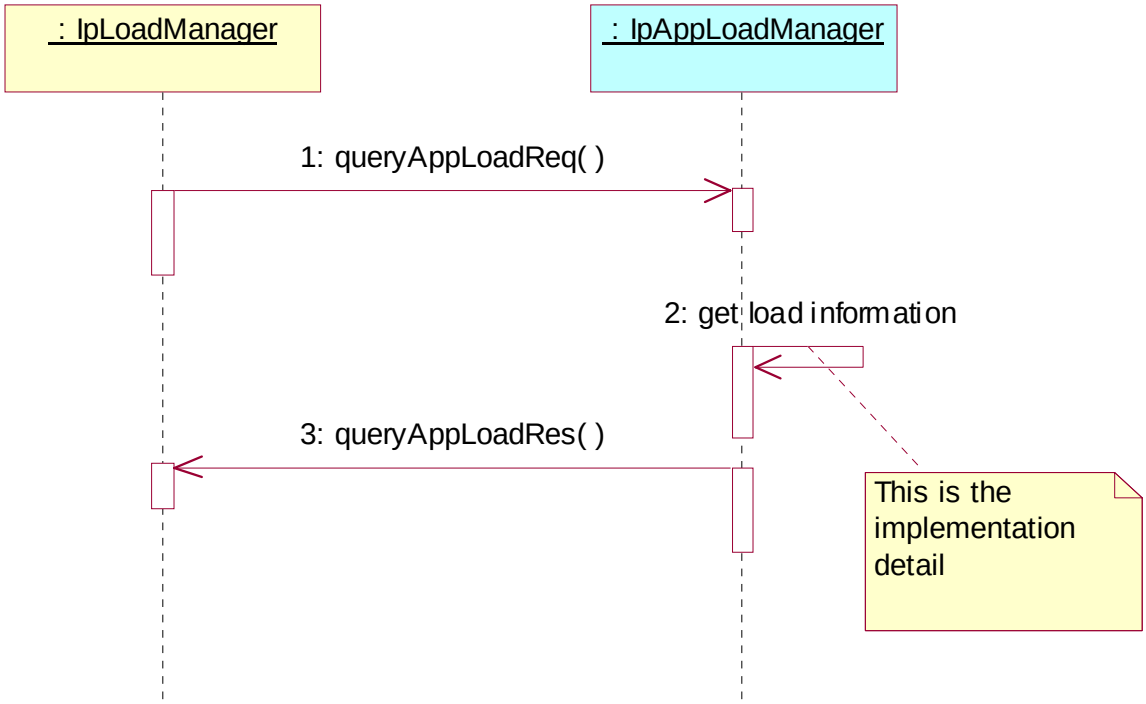
7.1.2.1 Load Management: Suspend/resume notification from application

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



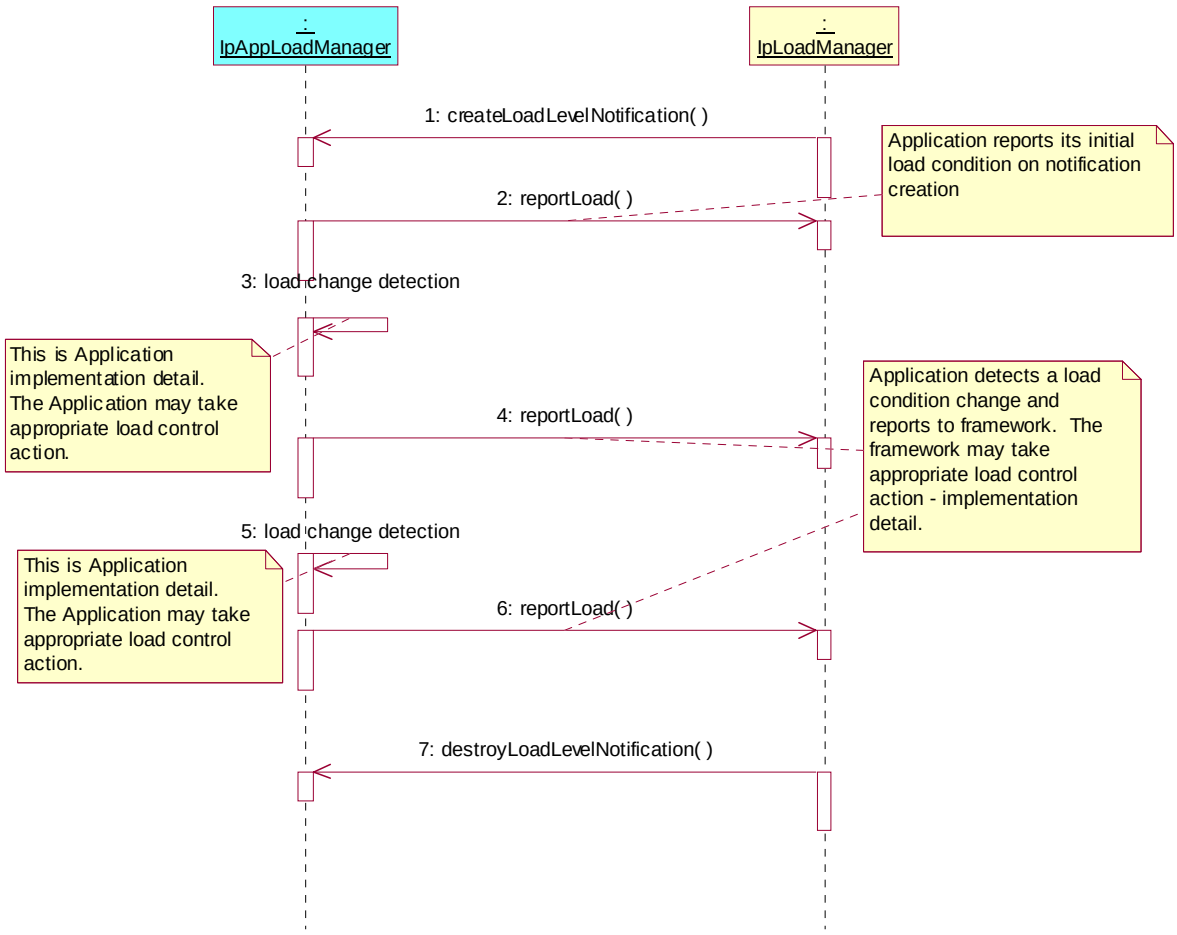
7.1.2.2 Load Management: Framework queries load statistics

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



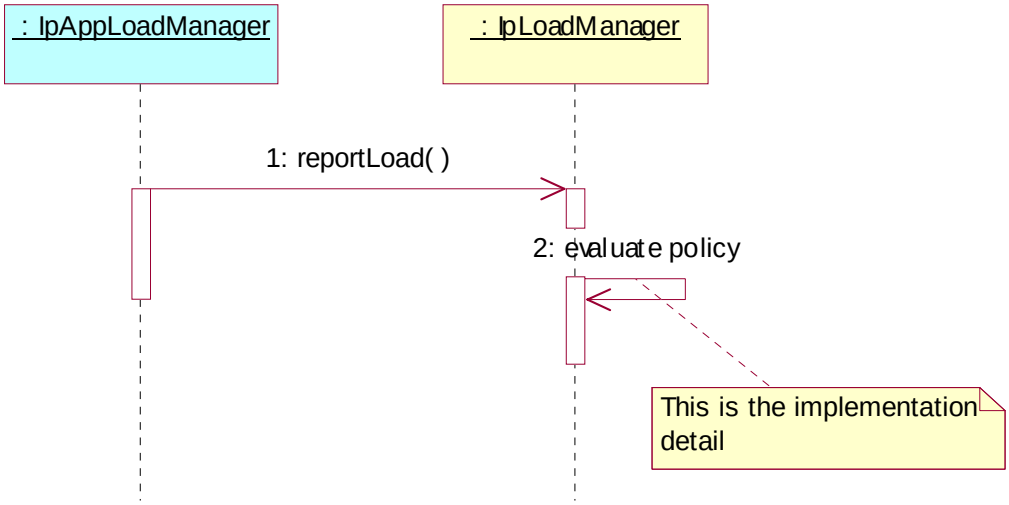
7.1.2.3 Load Management: Framework callback registration and Application load control

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



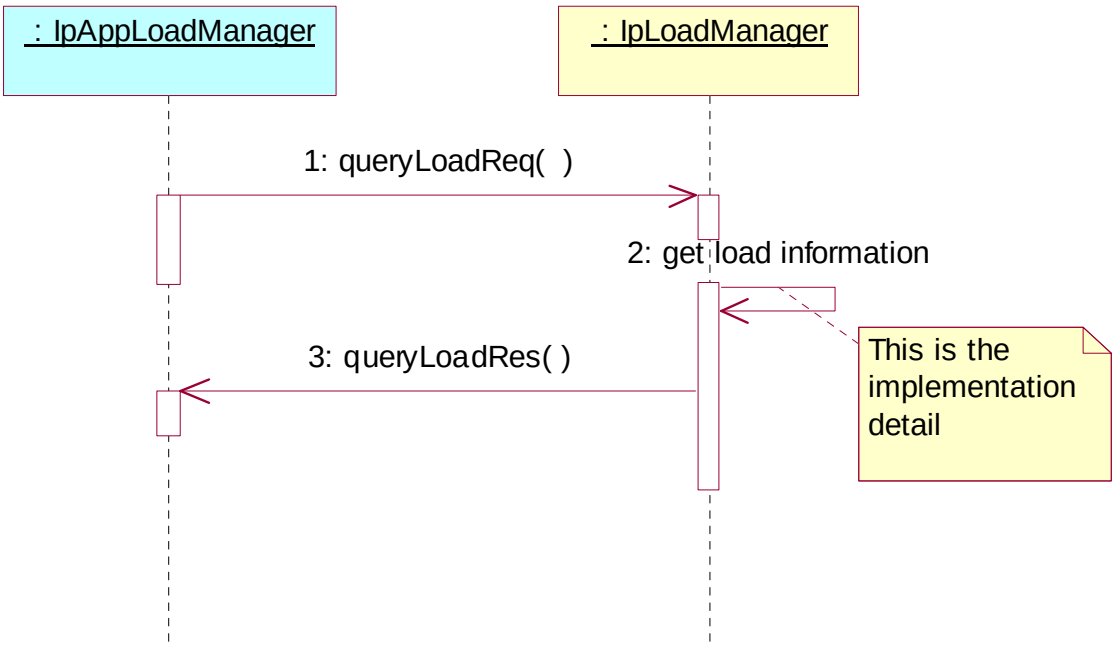
7.1.2.4 Load Management: Application reports current load condition

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



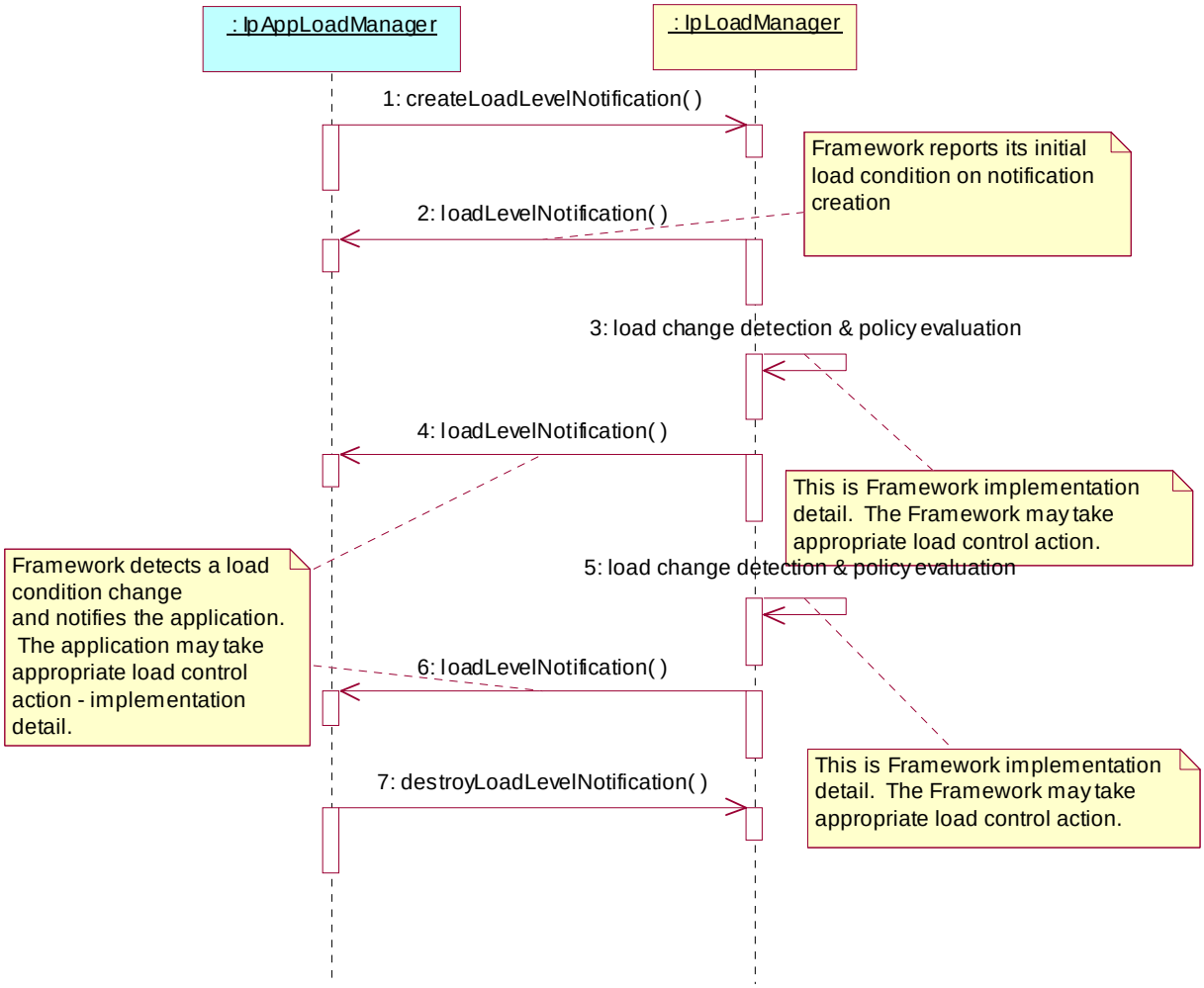
7.1.2.5 Load Management: Application queries load statistics

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



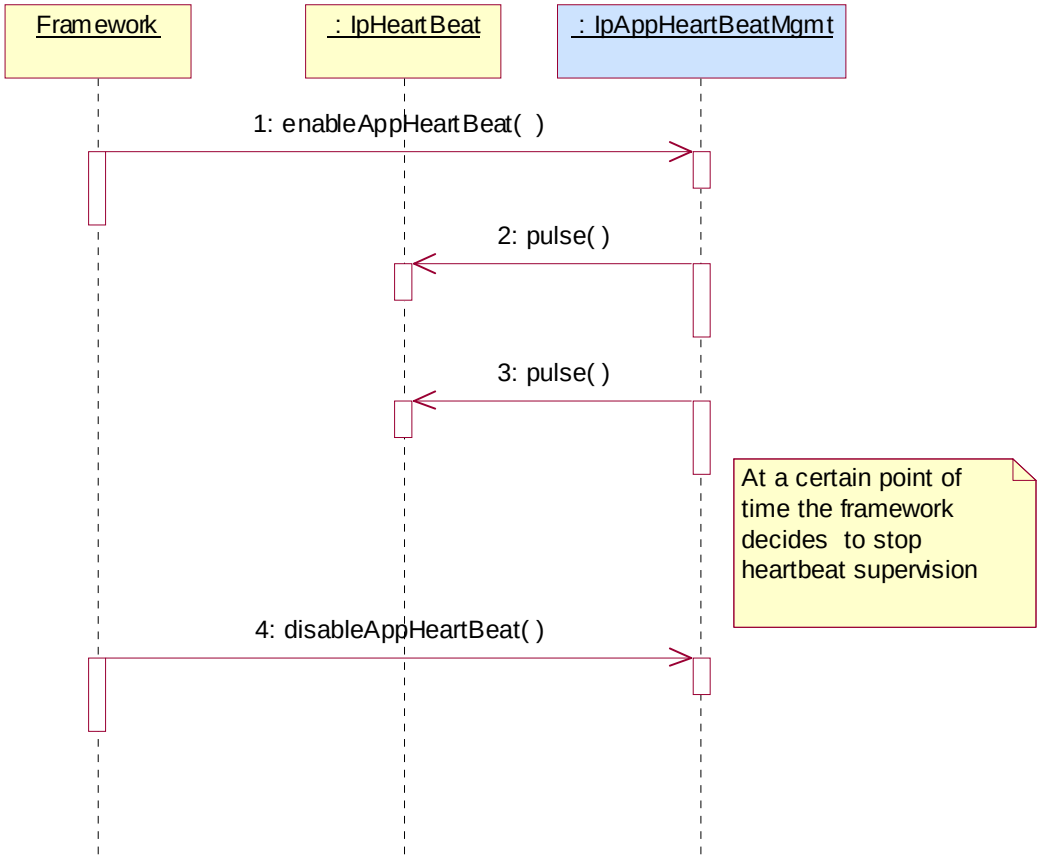
7.1.2.6 Load Management: Application callback registration and load control

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



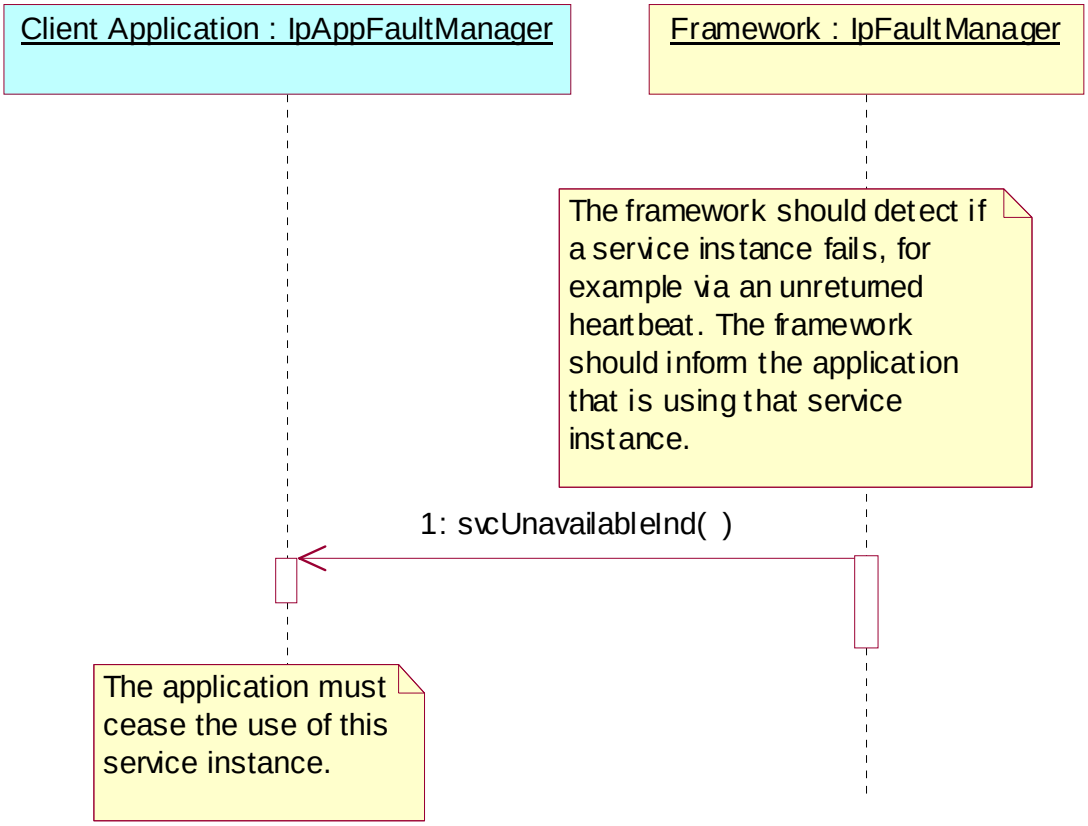
7.1.2.7 Heartbeat Management: Start/perform/end heartbeat supervision of the application

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



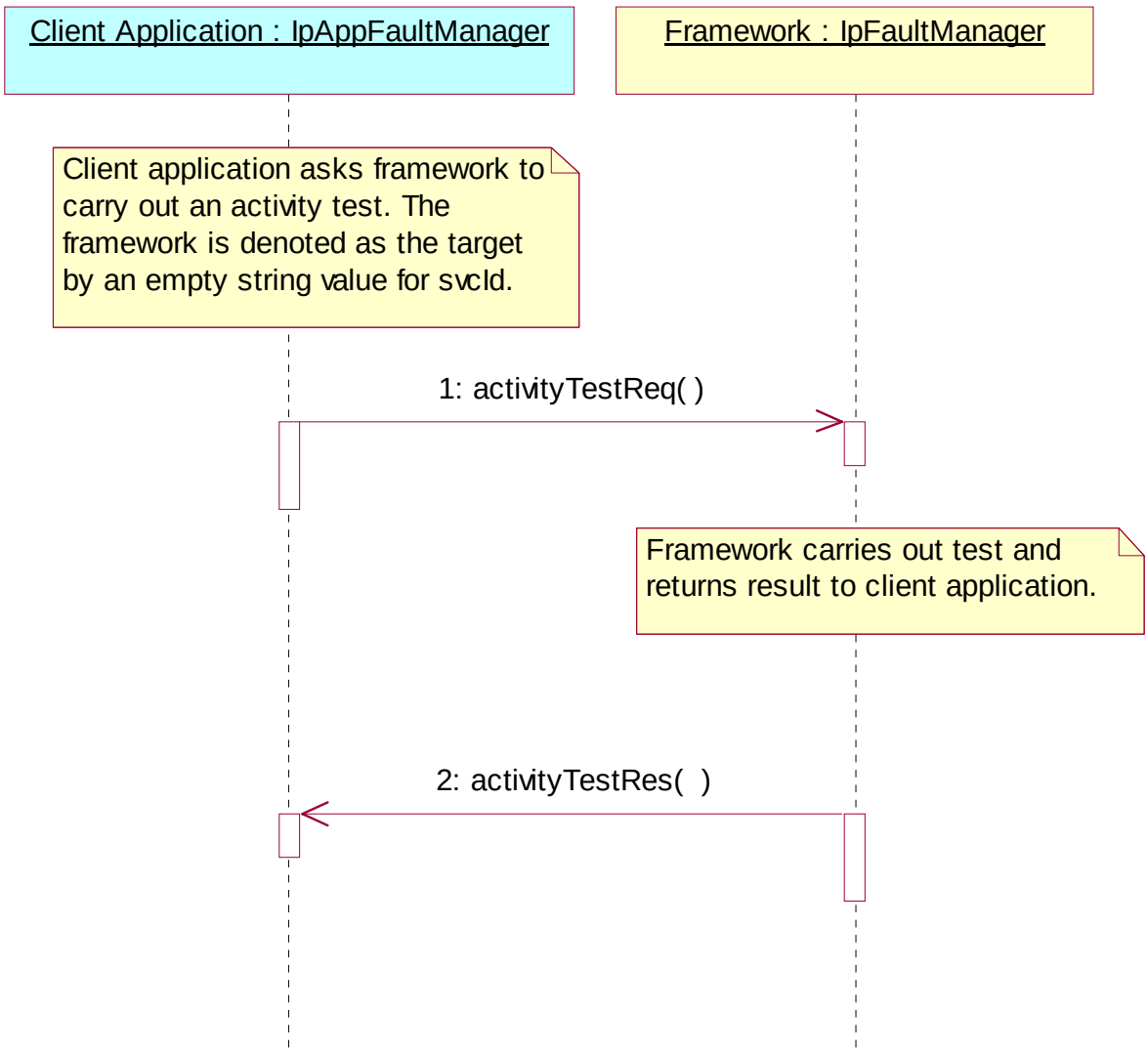
7.1.2.8 Fault Management: Framework detects a Service failure

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



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

7.1.2.9 Fault Management: Application requests a Framework activity test



1: The client application asks the framework to do an activity test. The client identifies that it would like the activity test done for the framework, rather then a service, by supplying an empty string value for the svcId parameter.

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

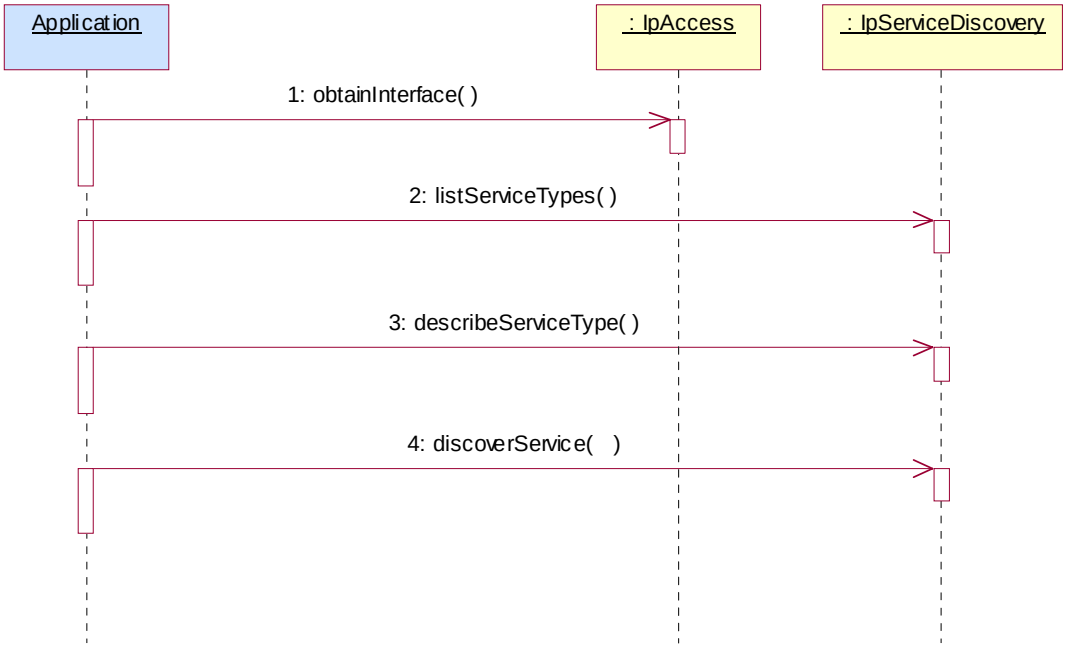
7.1.3 Service Discovery Sequence Diagrams

7.1.3.1 Service Discovery

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

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

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



2: Discovery: first step - list service types

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

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

- out listTypes.

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

3: Discovery: second step - describe service type

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

The following input is necessary:

- in name.

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

And the output is:

- out serviceTypeDescription.

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

- the property names associated with the SCF;
- the corresponding property value types;
- the corresponding property mode (mandatory or read only) associated with each SCF property;
- the names of the super types of this type; and
- whether the type is currently enabled or disabled.

4: Discovery: third step - discover service

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

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

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

Input parameters are:

- in serviceTypeName.

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

- in desiredPropertyList.

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

The following parameter is necessary as input:

- in max.

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

And the output is:

- out serviceList.

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

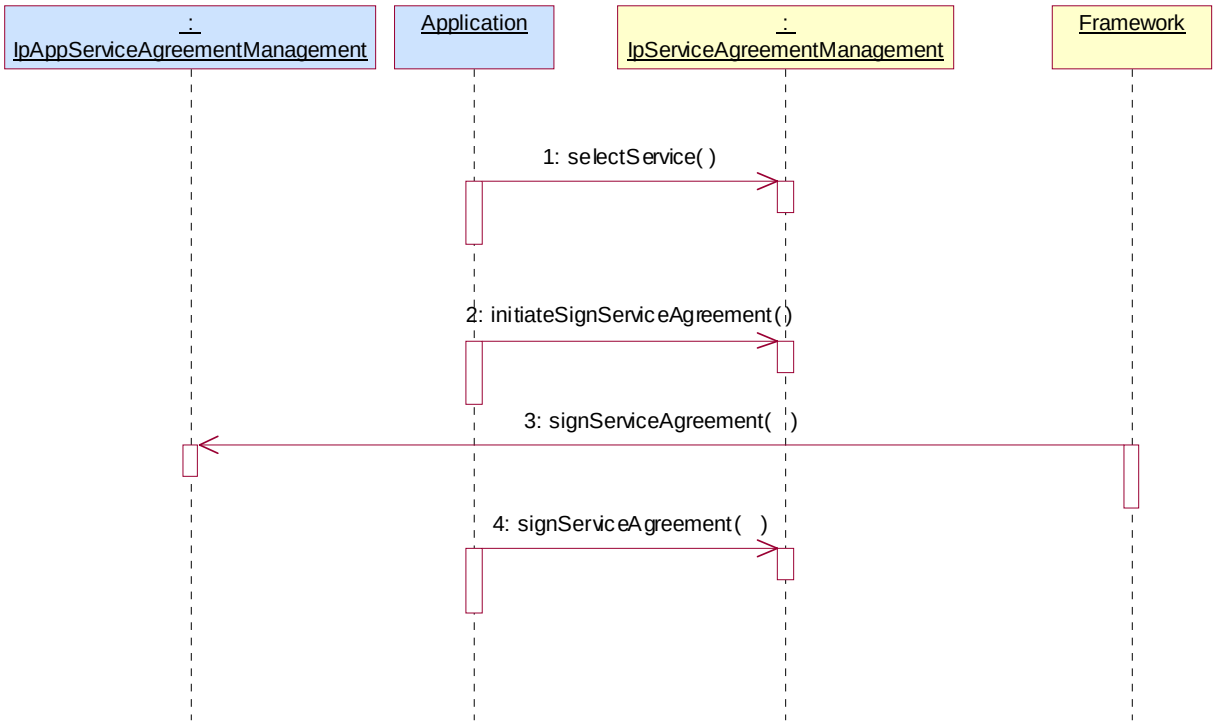
7.1.4 Service Agreement Management Sequence Diagrams

7.1.4.1 Service Selection

The following figure shows the process of selecting an SCF.

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

This is achieved by the following two steps:



1: Service Selection: first step - selectService

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

Input is:

- in serviceID.

This identifies the SCF required.

And output:

- out serviceToken.

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

2: Service Selection: second step - signServiceAgreement

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

Input:

- in serviceToken.

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

- in agreementText.

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

- in signingAlgorithm.

This is the algorithm used to compute the digital signature.

Output:

- out signatureAndServiceMgr.

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

7.2 Class Diagrams

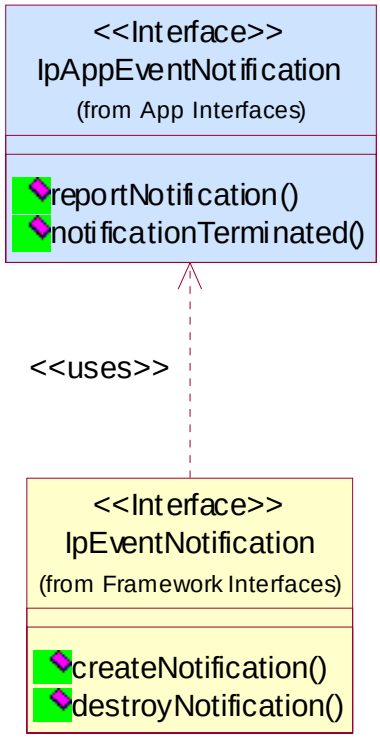


Figure 5: Event Notification Class Diagram

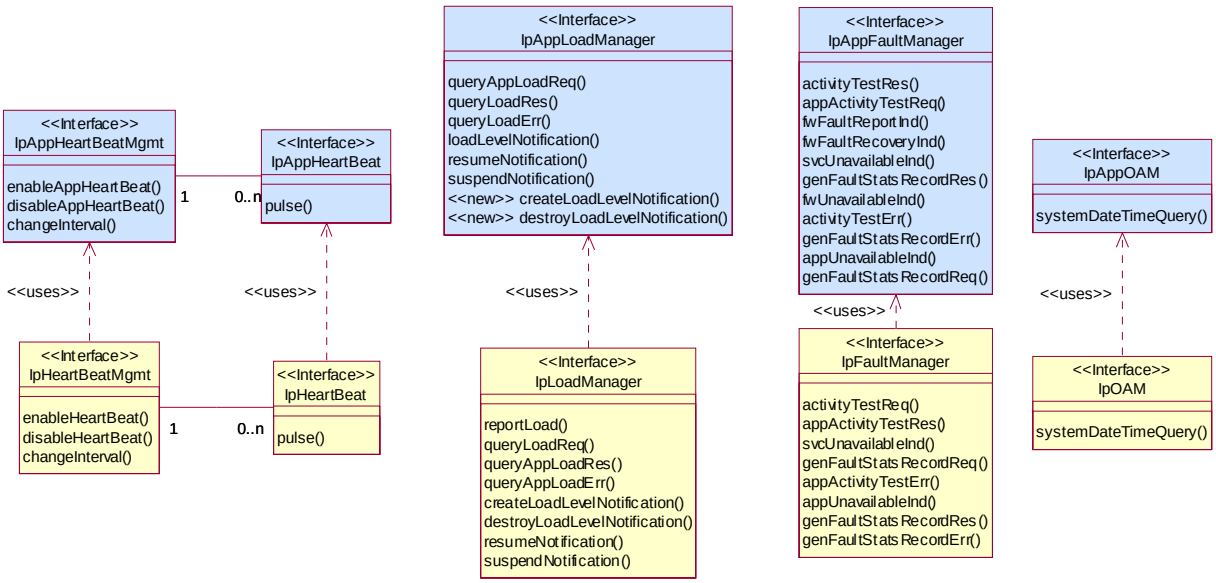


Figure 6: Integrity Management Package Overview

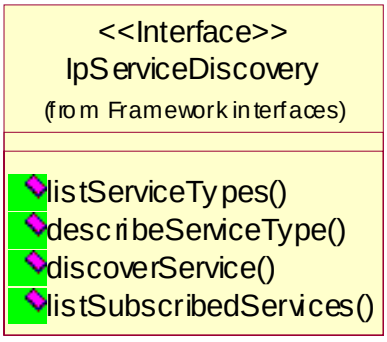


Figure 7: Service Discovery Package Overview

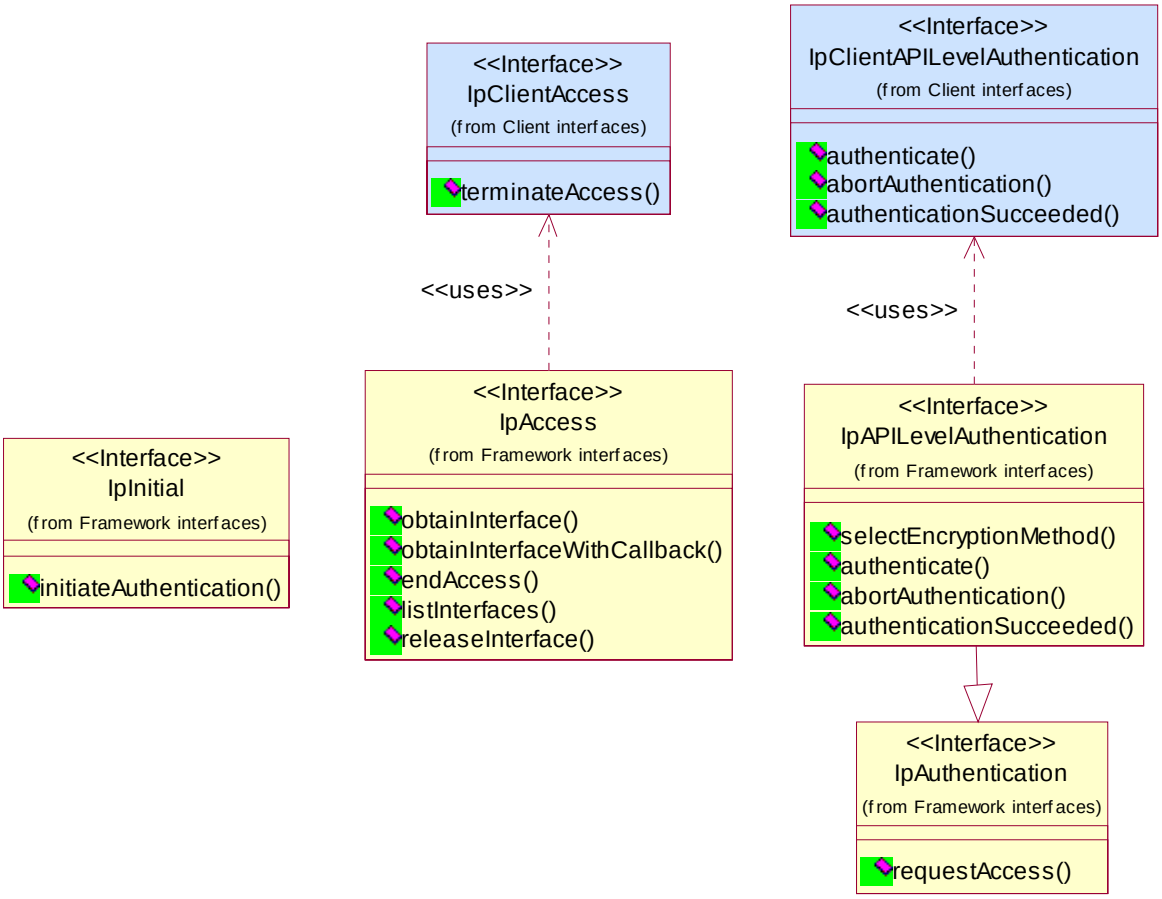


Figure 8: Trust and Security Management Package Overview

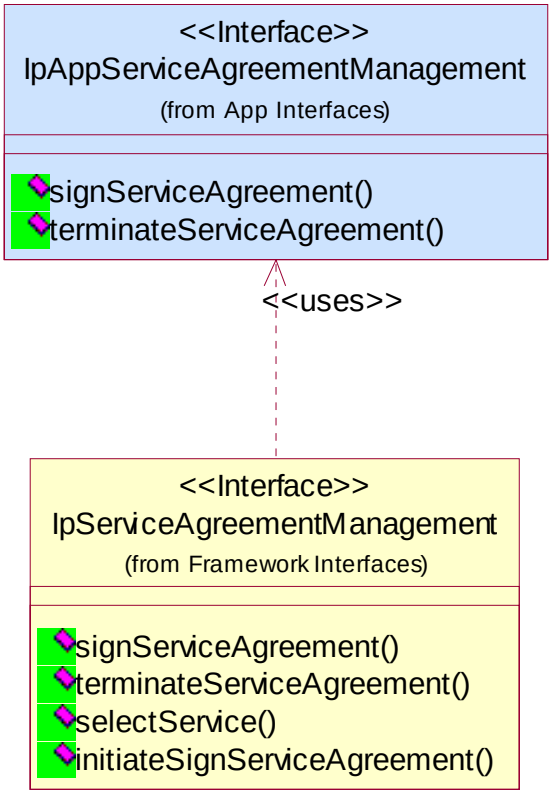


Figure 9: Service Agreement Management Package Overview

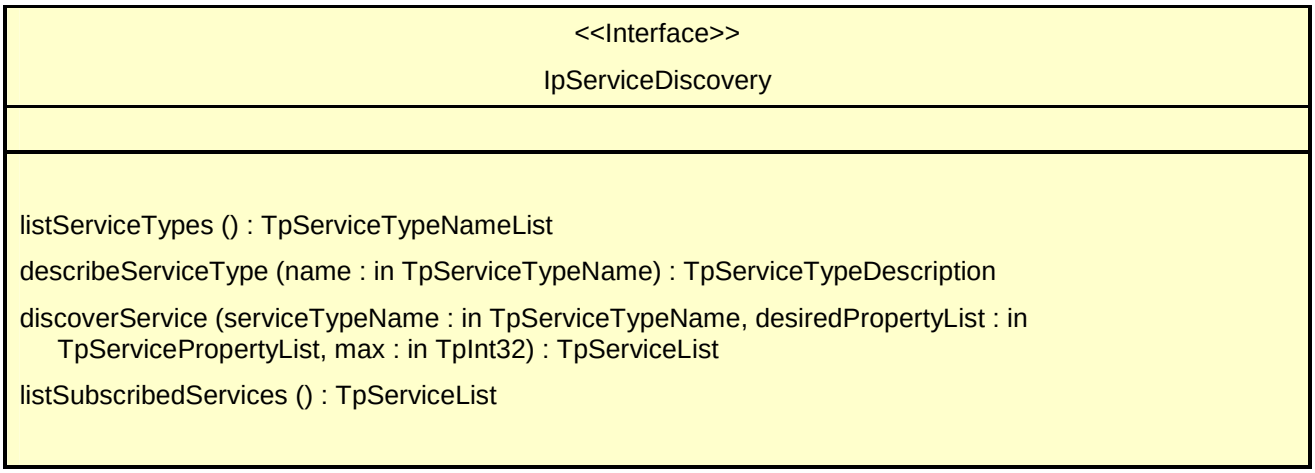
7.3 Interface Classes

7.3.1 Service Discovery Interface Classes

7.3.1.1 Interface Class IpServiceDiscovery

Inherits from: IpInterface.

The service discovery interface, shown below, consists of four methods. Before a service can be discovered, the enterprise operator (or the client applications) must know what "types" of services are supported by the Framework and what service "properties" are applicable to each service type. The listServiceType() method returns a list of all "service types" that are currently supported by the framework and the "describeServiceType()" returns a description of each service type. The description of service type includes the "service-specific properties" that are applicable to each service type. Then the enterprise operator (or the client applications) can discover a specific set of registered services that both belong to a given type and possess the desired "property values", by using the "discoverService() method". Once the enterprise operator finds out the desired set of services supported by the framework, it subscribes to (a sub-set of) these services using the Subscription Interfaces. The enterprise operator (or the client applications in its domain) can find out the set of services available to it (i.e. the service that it can use) by invoking "listSubscribedServices()". The service discovery APIs are invoked by the enterprise operators or client applications. They are described below.
This interface shall be implemented by a Framework with as a minimum requirement the listServiceTypes(), describeServiceType() and discoverService() methods.



Method
listServiceTypes()

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

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

Parameters
No Parameters were identified for this method.

Returns

TpServiceTypeNameList

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method
describeServiceType()

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

Returns <serviceTypeDescription> : The description of the specified service type. The description provides information about:

- the service properties associated with this service type: i.e. a list of service property {name, mode and type} tuples;
- the names of the super types of this service type; and
- whether the service type is currently available or unavailable.

Parameters

name : in TpServiceTypeName

The name of the service type to be described.

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

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

Returns

TpServiceTypeDescription

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE

Method

discoverService()

The discoverService operation is the means by which a client application is able to obtain the service IDs of the services that meet its requirements. The client application passes in a list of desired service properties to describe the service it is looking for, in the form of attribute/value pairs for the service properties. The client application also specifies the maximum number of matched responses it is willing to accept. The framework must not return more matches than the specified maximum, but it is up to the discretion of the Framework implementation to choose to return less than the specified maximum. The discoverService() operation returns a serviceID/Property pair list for those services that match the desired service property list that the client application provided. The service properties returned will form a complete view of what the client application will be able to do with the service, as per the service level agreement. If the framework supports service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties.

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

Parameters

serviceTypeName : in TpServiceTypeName

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

If the string representation of the "type" does not obey the rules for service type identifiers, then the P_ILLEGAL_SERVICE_TYPE exception is raised.

If the "type" is correct syntactically but is not recognised as a service type within the Framework, then the P_UNKNOWN_SERVICE_TYPE exception is raised.

The framework may return a service of a subtype of the "type" requested. A service sub-type can be described by the properties of its supertypes.

desiredPropertyList : in TpServicePropertyList

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

max : in TpInt32

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

Returns

TpServiceList

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE, P_INVALID_PROPERTY

Method

listSubscribedServices()

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

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

Parameters

No Parameters were identified for this method.

Returns

TpServiceList

Raises

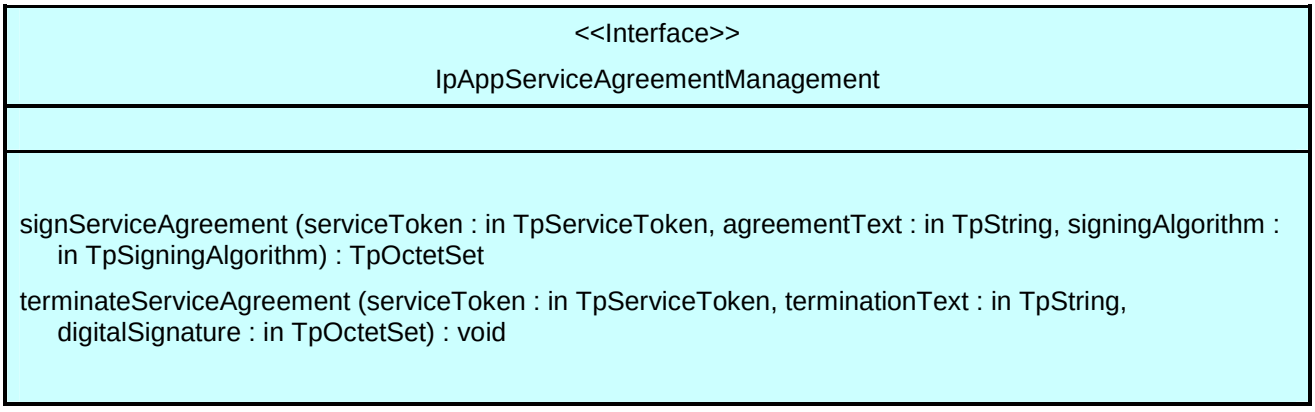
TpCommonExceptions, P_ACCESS_DENIED

7.3.2 Service Agreement Management Interface Classes

7.3.2.1 Interface Class IpAppServiceAgreementManagement

Inherits from: IpInterface.

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



Method

signServiceAgreement()

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

Returns <digitalSignature> : The digitalSignature is the signed version of a hash of the service token and agreement text given by the framework. If no signing algorithm is used, the digitalSignature is the octet sequence of the service token and the agreement text. If the signature is incorrect the serviceToken will be expired immediately.

Parameters

serviceToken : in TpServiceToken

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

agreementText : in TpString

This is the agreement text that is to be signed by the client application using the private key of the client application. If the agreementText is invalid, then the P_INVALID_AGREEMENT_TEXT exception is thrown.

signingAlgorithm : in TpSigningAlgorithm

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

Returns

TpOctetSet

Raises

TpCommonExceptions, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM

Method

terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpOctetSet

This is a signed version of a hash of the service token and the termination text. If no signing algorithm is used, the digitalSignature is the octet sequence of the termination text itself. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the service agreement is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

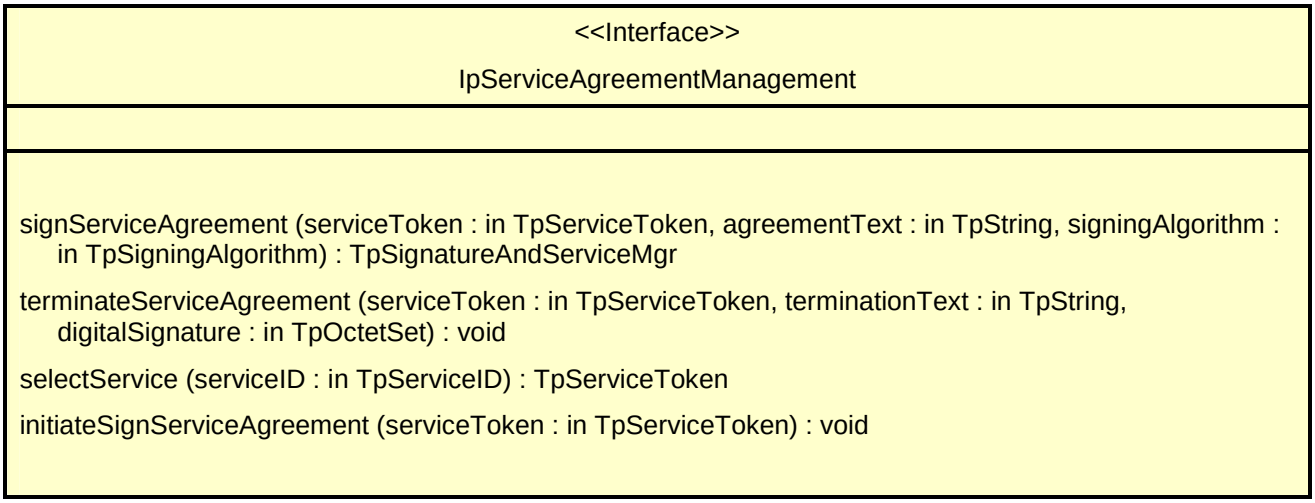
Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

7.3.2.2 Interface Class IpServiceAgreementManagement

Inherits from: IpInterface.

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



Method

signServiceAgreement()

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

Returns <signatureAndServiceMgr> : This contains the digital signature of the framework for the service agreement, and a reference to the service manager interface of the service.

```
structure TpSignatureAndServiceMgr {
        digitalSignature: TpOctetSet;
        serviceMgrInterface: IpServiceRef;
};
```

The digitalSignature is the signed version of a hash of the service token and agreement text given by the client application. If no signing algorithm is used, the digitalSignature is the octet sequence of the service token and the agreement text given by the client application.

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

Parameters

serviceToken : in TpServiceToken

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

agreementText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

Returns

TpSignatureAndServiceMgr

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM, P_SERVICE_ACCESS_DENIED

Method

terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpOctetSet

This is a signed version of a hash of the service token and the termination text. If no signing algorithm is used, the digitalSignature is the octet sequence of the termination text itself. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement().The framework uses this to check that the terminationText has been signed by the client application. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

Method
selectService()

This method is used by the client application to identify the service that the client application wishes to use. If the client application is not allowed to access the service, then the P_SERVICE_ACCESS_DENIED exception is thrown. The P_SERVICE_ACCESS_DENIED exception is also thrown if the client attempts to select a service for which it has already signed a service agreement for, and therefore obtained an instance of. This is because there must be only one service instance per client application.

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

Parameters

serviceID : in TpServiceID

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

Returns

TpServiceToken

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_SERVICE_ACCESS_DENIED

Method
initiateSignServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

Raises

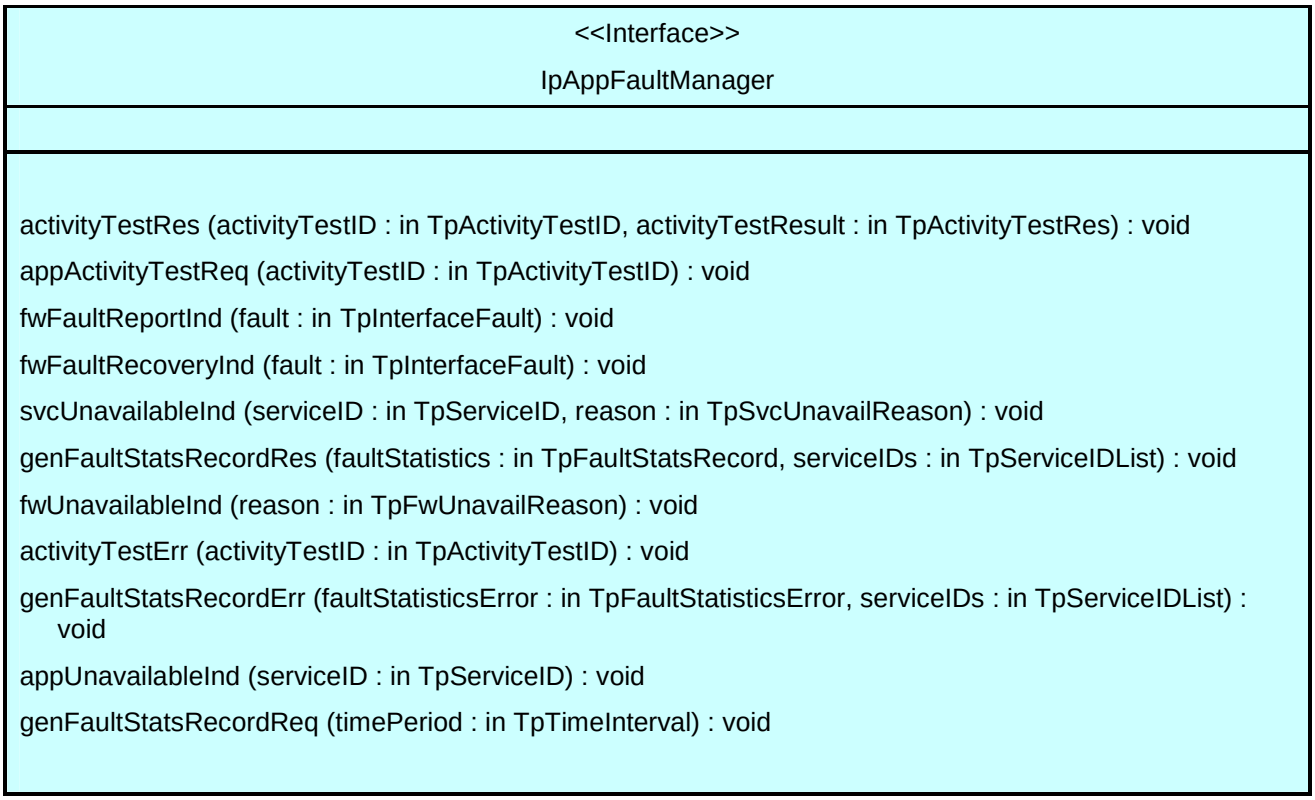
TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_SERVICE_ACCESS_DENIED

7.3.3 Integrity Management Interface Classes

7.3.3.1 Interface Class IpAppFaultManager

Inherits from: IpInterface.

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



Method

activityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Method

appActivityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

Method

fwFaultReportInd()

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

Parameters

`fault : in TpInterfaceFault`

Specifies the fault that has been detected by the framework.

Method

fwFaultRecoveryInd()

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

Parameters

`fault : in TpInterfaceFault`

Specifies the fault from which the framework has recovered.

Method

svcUnavailableInd()

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

Parameters

`serviceID : in TpServiceID`

Identifies the affected service.

`reason : in TpSvcUnavailReason`

Identifies the reason why the service is no longer available.

Method

genFaultStatsRecordRes()

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

Parameters

`faultStatistics : in TpFaultStatsRecord`

The fault statistics record.

`serviceIDs : in TpServiceIDList`

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

Method

fwUnavailableInd()

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

Parameters

`reason : in TpFwUnavailReason`

Identifies the reason why the framework is no longer available.

Method
activityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Method
genFaultStatsRecordErr()

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

Parameters

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

serviceIDs : in TpServiceIDList

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

Method
appUnavailableInd()

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

Parameters

serviceID : in TpServiceID

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

Method
genFaultStatsRecordReq()

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

Parameters

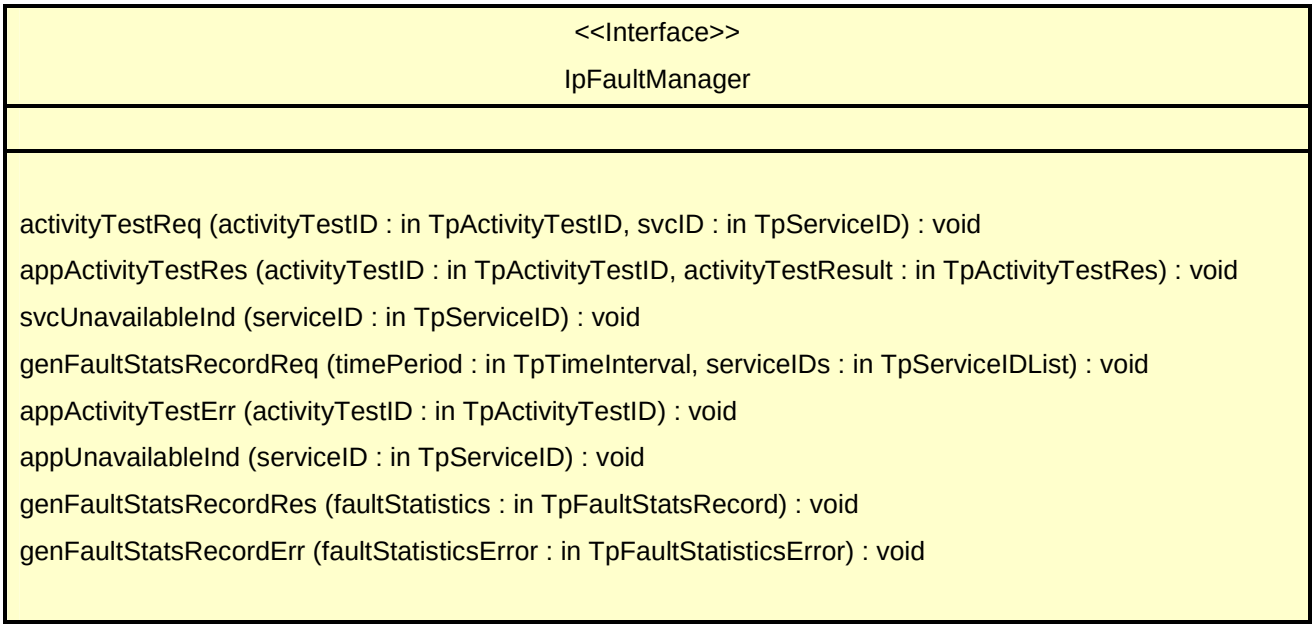
timePeriod : in TpTimeInterval

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

7.3.3.2 Interface Class IpFaultManager

Inherits from: IpInterface.

This interface is used by the application to inform the framework of events that affect the integrity of the framework and services, and to request information about the integrity of the system. The fault manager operations do not exchange callback interfaces as it is assumed that the client application supplies its Fault Management callback interface at the time it obtains the Framework's Fault Management interface, by use of the obtainInterfaceWithCallback operation on the IpAccess interface.
If the IpFaultManager interface is implemented by a Framework, at least one of these methods shall be implemented. If the Framework is capable of invoking the IpAppFaultManager.appActivityTestReq() method, it shall implement appActivityTestRes() and appActivityTestErr() in this interface. If the Framework is capable of invoking IpAppFaultManager.genFaultStatsRecordReq(), it shall implement genFaultStatsRecordRes() and genFaultStatsRecordErr() in this interface.



Method
activityTestReq()

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

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

Parameters

activityTestID : in TpActivityTestID

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

svcID : in TpServiceID

Identifies either the framework or a service for testing. The framework is designated by an empty string.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

appActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_INVALID_ACTIVITY_TEST_ID

Method

svcUnavailableInd()

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

Parameters

serviceID : in TpServiceID

Identifies the service that the application can no longer use.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

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

serviceIDs : in TpServiceIDList

Specifies either the framework or services to be included in the general fault statistics record. If this parameter is not an empty list, the fault statistics records of the client's instances of the specified services are returned, otherwise the fault statistics record of the framework is returned.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

appActivityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

appUnavailableInd()

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

Parameters

serviceID : in TpServiceID

Identifies the affected application.

Raises

TpCommonExceptions

Method

genFaultStatsRecordRes()

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

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

Raises

TpCommonExceptions

Method

genFaultStatsRecordErr()

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

Parameters

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

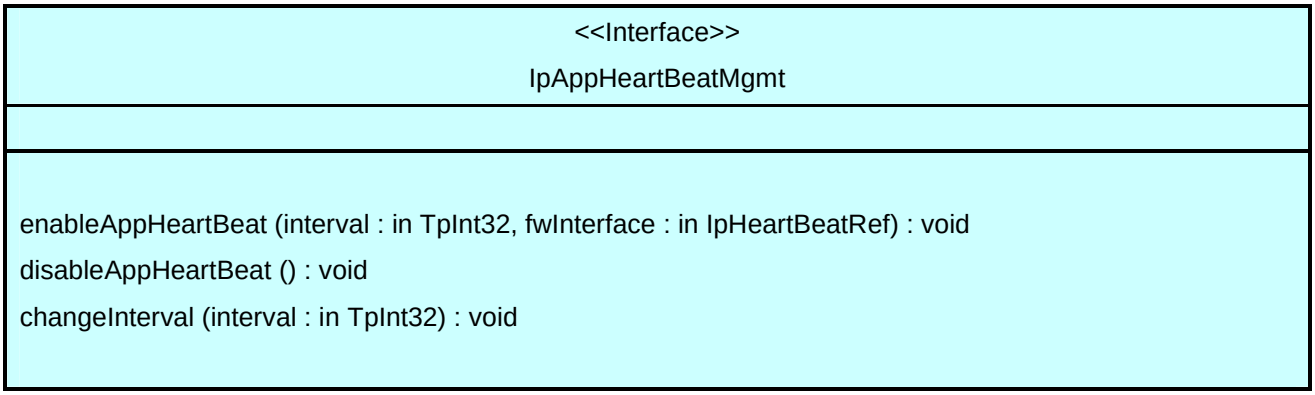
Raises

TpCommonExceptions

7.3.3.3 Interface Class IpAppHeartBeatMgmt

Inherits from: IpInterface.

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



Method

enableAppHeartBeat()

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

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

fwInterface : in IpHeartBeatRef

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

Method

disableAppHeartBeat()

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

Parameters

No Parameters were identified for this method.

Method

changeInterval()

Allows the administrative change of the heartbeat interval.

Parameters

`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

7.3.3.4 Interface Class IpAppHeartBeat

Inherits from: IpInterface.

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

<<Interface>> IpAppHeartBeat
pulse () : void

Method

`pulse()`

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

Parameters

No Parameters were identified for this method.

7.3.3.5 Interface Class IpHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the framework by a client application. If the IpHeartBeatMgmt interface is implemented by a Framework, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.

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

Method

`enableHeartBeat()`

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

Parameters

`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

`appInterface : in IpAppHeartBeatRef`

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

Raises

`TpCommonExceptions`

Method

`disableHeartBeat()`

Instructs the framework to cease the sending of its heartbeat.

Parameters

No Parameters were identified for this method.

Raises

`TpCommonExceptions`

Method

`changeInterval()`

Allows the administrative change of the heartbeat interval.

Parameters

`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

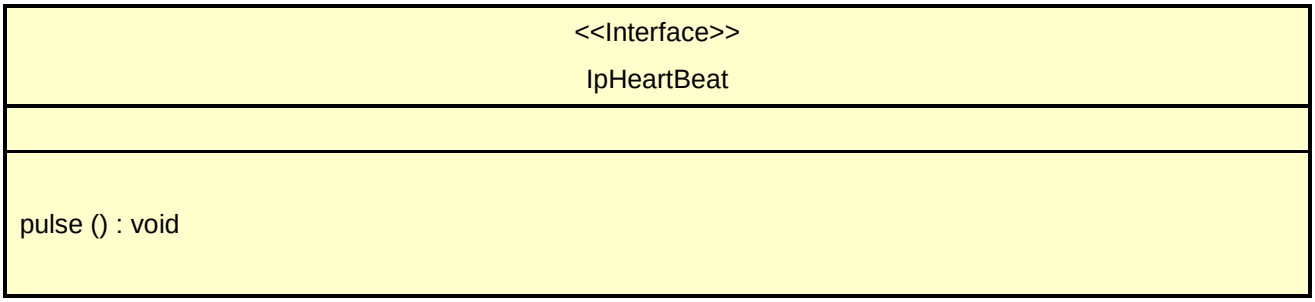
Raises

`TpCommonExceptions`

7.3.3.6 Interface Class IpHeartBeat

Inherits from: IpInterface.

The Heartbeat Framework interface is used by the client application to send its heartbeat. If a Framework is capable of invoking IpAppHeartBeatMgmt.enableHeartBeat(), it shall implement IpHeartBeat and the pulse() method.



Method
pulse()

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

Parameters
No Parameters were identified for this method.

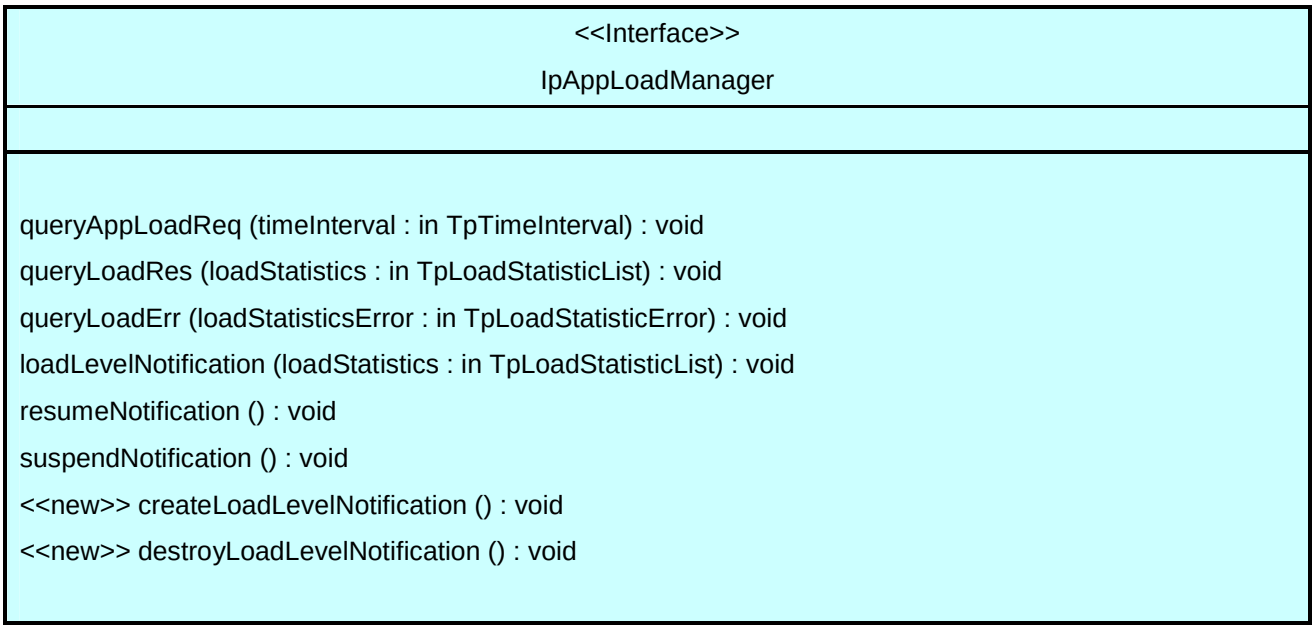
Raises

TpCommonExceptions

7.3.3.7 Interface Class IpAppLoadManager

Inherits from: IpInterface.

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



Method
queryAppLoadReq()

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

Parameters

timeInterval : in TpTimeInterval

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

Method
queryLoadRes()

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

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics.

Method

queryLoadErr()

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

Parameters

loadStatisticsError : in TpLoadStatisticError

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

Method

loadLevelNotification()

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

Parameters

loadStatistics : in TpLoadStatisticList

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

Method

resumeNotification()

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

Parameters

No Parameters were identified for this method.

Method

suspendNotification()

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

Parameters

No Parameters were identified for this method.

Method

<<new>> createLoadLevelNotification()

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

Parameters
No Parameters were identified for this method.

Method
<<new>> destroyLoadLevelNotification()

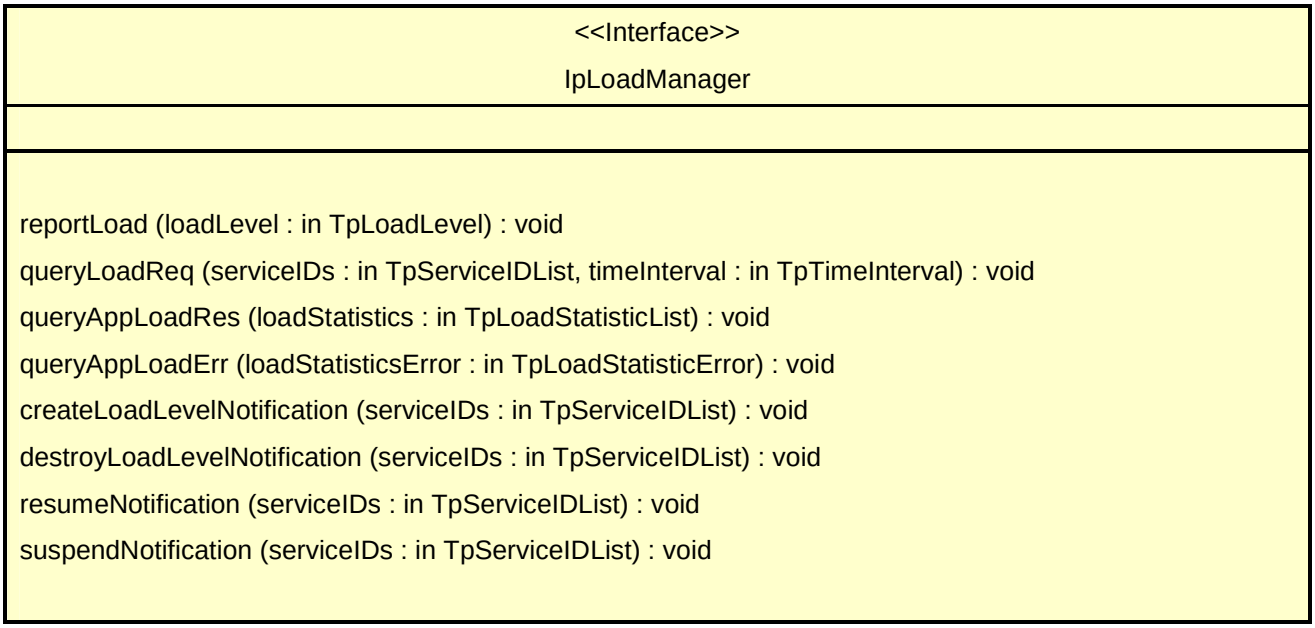
The framework uses this method to unregister for notifications of load level changes associated with the application.

Parameters
No Parameters were identified for this method.

7.3.3.8 Interface Class IpLoadManager

Inherits from: IpInterface.

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



Method
reportLoad()

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

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

Parameters

loadLevel : in TpLoadLevel

Specifies the application's load level.

Raises

TpCommonExceptions

Method
queryLoadReq()

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

Parameters

serviceIDs : in TpServiceIDList

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

timeInterval : in TpTimeInterval

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

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

Method
queryAppLoadRes()

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

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the application-supplied load statistics.

Raises

TpCommonExceptions

Method

queryAppLoadErr()

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

Parameters

loadStatisticsError : in TpLoadStatisticError

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

Raises

TpCommonExceptions

Method

createLoadLevelNotification()

The client application uses this method to register to receive notifications of load level changes associated with either the framework or with its instances of the individual services used by the application. If the application does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID. Upon receipt of this method the framework shall inform the client application of the current framework or service instance load using the loadLevelNotification method on the corresponding IpAppLoadManager.

Parameters

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

destroyLoadLevelNotification()

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

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which load level changes should no longer be reported. To unregister for framework load control, the serviceIDs parameter must be an empty list.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

resumeNotification()

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

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which the sending of notifications of load level changes by the framework should be resumed. To resume notifications for the framework, the serviceIDs parameter must be an empty list.

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

Method

suspendNotification()

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

Parameters

serviceIDs : in TpServiceIDList

Specifies the framework or the services for which the sending of notifications by the framework should be suspended. To suspend notifications for the framework, the serviceIDs parameter must be an empty list.

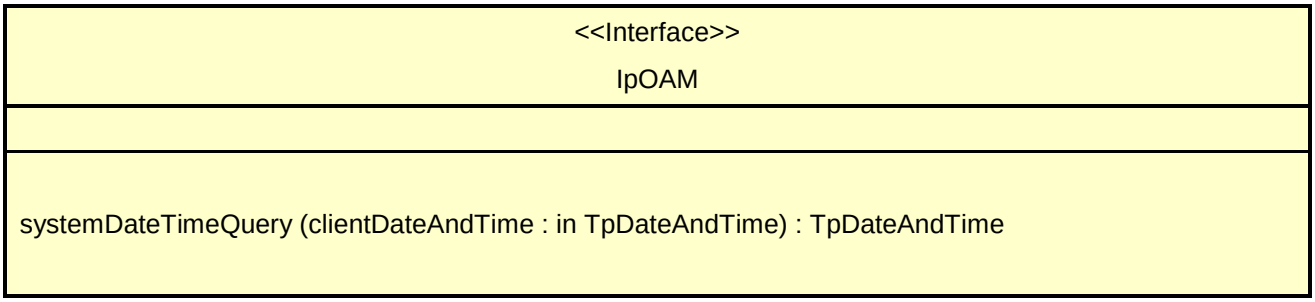
Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_SERVICE_NOT_ENABLED, P_UNAUTHORISED_PARAMETER_VALUE

7.3.3.9 Interface Class IpOAM

Inherits from: IpInterface.

The OAM interface is used to query the system date and time. The application and the framework can synchronise the date and time to a certain extent. Accurate time synchronisation is outside the scope of the OSA APIs. This interface and the systemDateTimeQuery() method are optional.



Method

systemDateTimeQuery()

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

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

Parameters

clientDateAndTime : in TpDateAndTime

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

Returns

TpDateAndTime

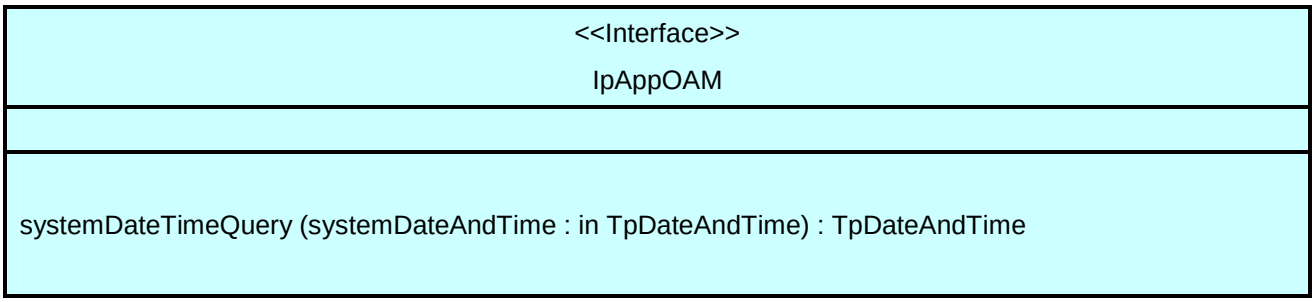
Raises

TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT

7.3.3.10 Interface Class IpAppOAM

Inherits from: IpInterface.

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



Method

systemDateTimeQuery()

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

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

Parameters

systemDateAndTime : in TpDateAndTime

This is the system date and time of the framework.

Returns

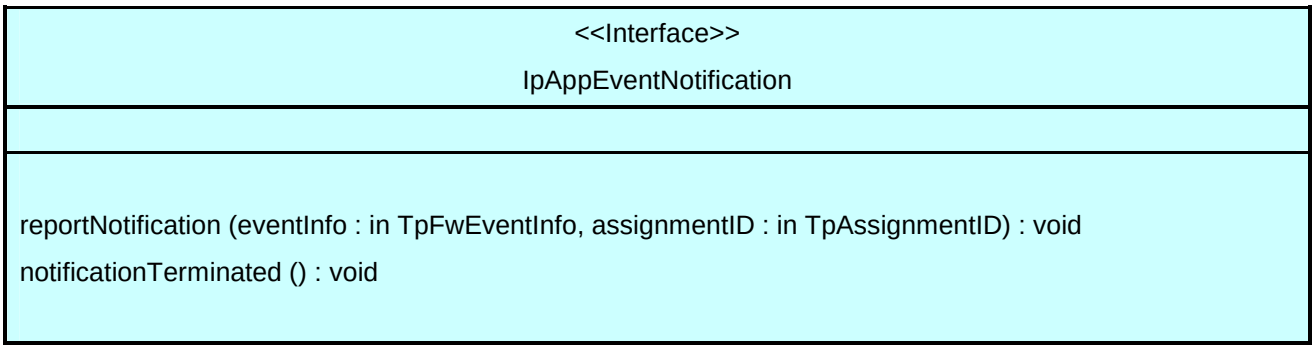
TpDateAndTime

7.3.4 Event Notification Interface Classes

7.3.4.1 Interface Class IpAppEventNotification

Inherits from: IpInterface.

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



Method
`reportNotification()`

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

Parameters

`eventInfo : in TpFwEventInfo`

Specifies specific data associated with this event.

`assignmentID : in TpAssignmentID`

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

Method
`notificationTerminated()`

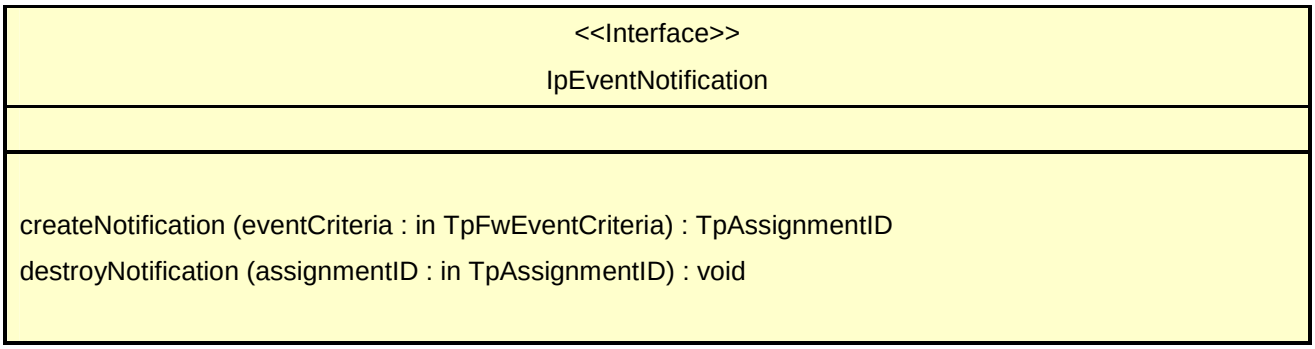
This method indicates to the application that all generic event notifications have been terminated (for example, due to faults detected).

Parameters
No Parameters were identified for this method.

7.3.4.2 Interface Class IpEventNotification

Inherits from: IpInterface.

The event notification mechanism is used to notify the application of generic service related events that have occurred. If Event Notifications are supported by a Framework, this interface and the createNotification() and destroyNotification() methods shall be implemented.



Method

createNotification()

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

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

Parameters

eventCriteria : in TpFwEventCriteria

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

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CRITERIA, P_INVALID_EVENT_TYPE

Method

destroyNotification()

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

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID given by the framework when the previous createNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the framework will return the error code P_INVALID_ASSIGNMENTID.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_ASSIGNMENT_ID

7.4 State Transition Diagrams

This 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.

7.4.1 Service Discovery State Transition Diagrams

7.4.1.1 State Transition Diagrams for IpServiceDiscovery

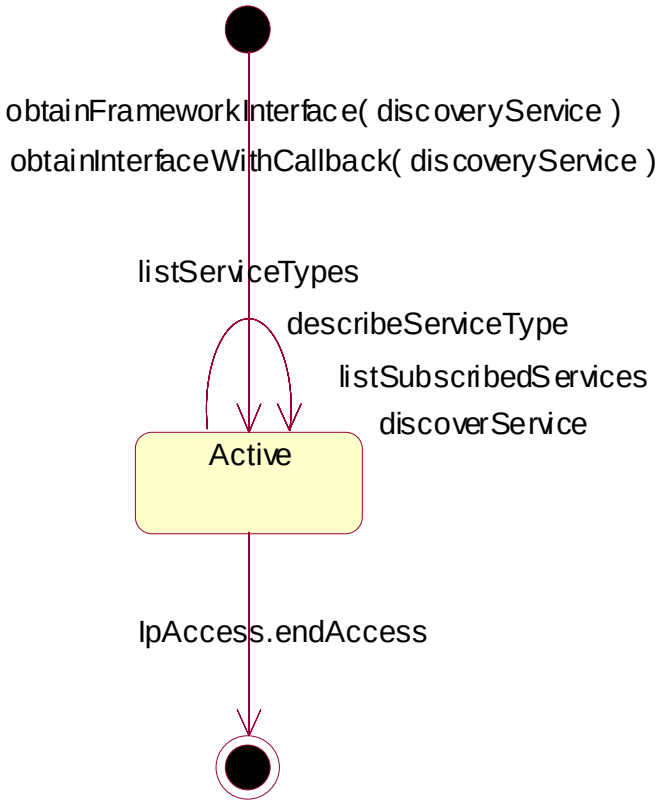


Figure 10: State Transition Diagram for IpServiceDiscovery

7.4.1.1.1 Active State

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

7.4.2 Service Agreement Management State Transition Diagrams

There are no State Transition Diagrams defined for Service Agreement Management.

7.4.3 Integrity Management State Transition Diagrams

7.4.3.1 State Transition Diagrams for IpLoadManager

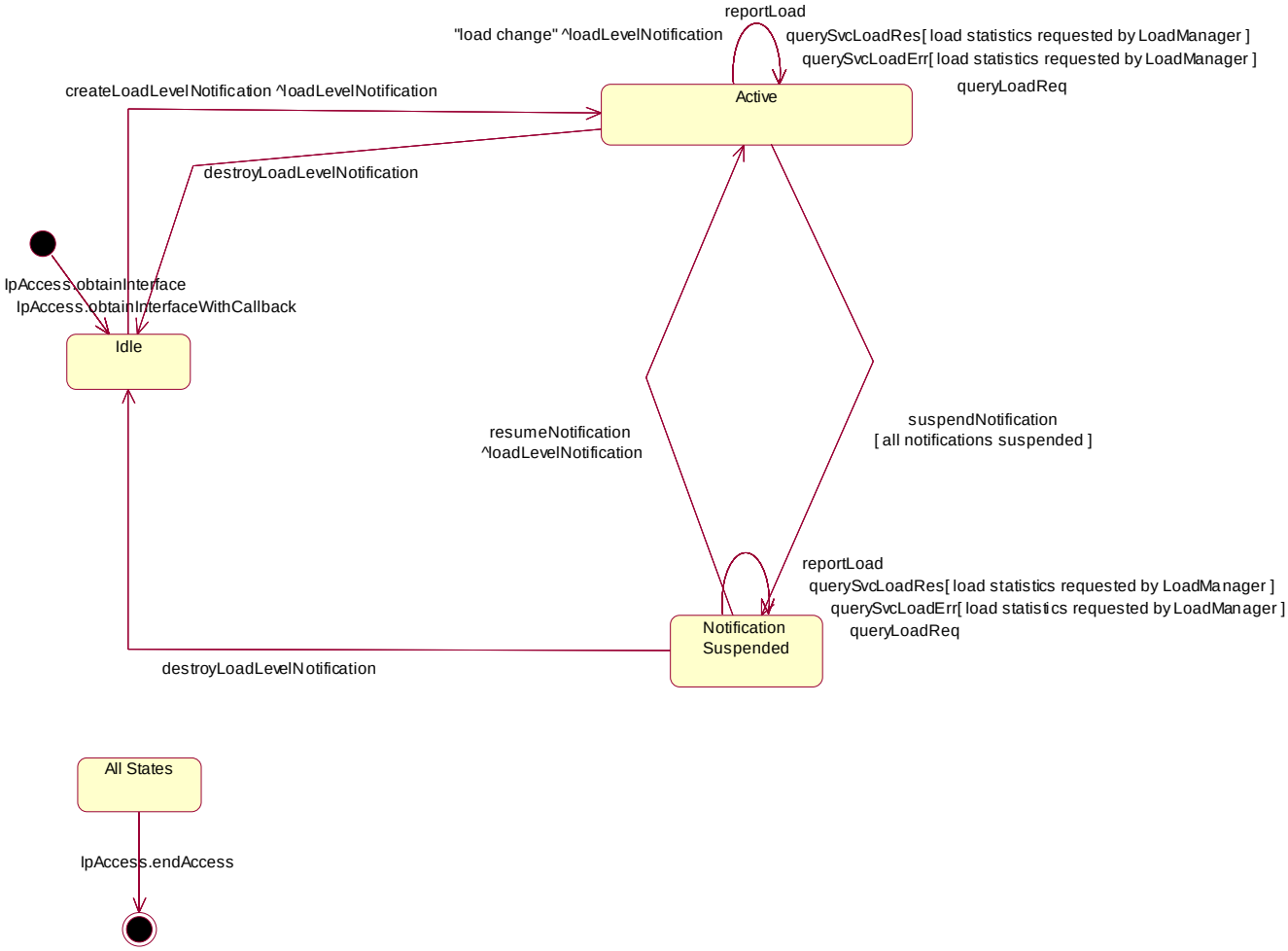


Figure 11: State Transition Diagram for IpLoadManager

7.4.3.1.1 Idle State

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

7.4.3.1.2 Notification Suspended State

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

7.4.3.1.3 Active State

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

7.4.3.2 State Transition Diagrams for LoadManagerInternal

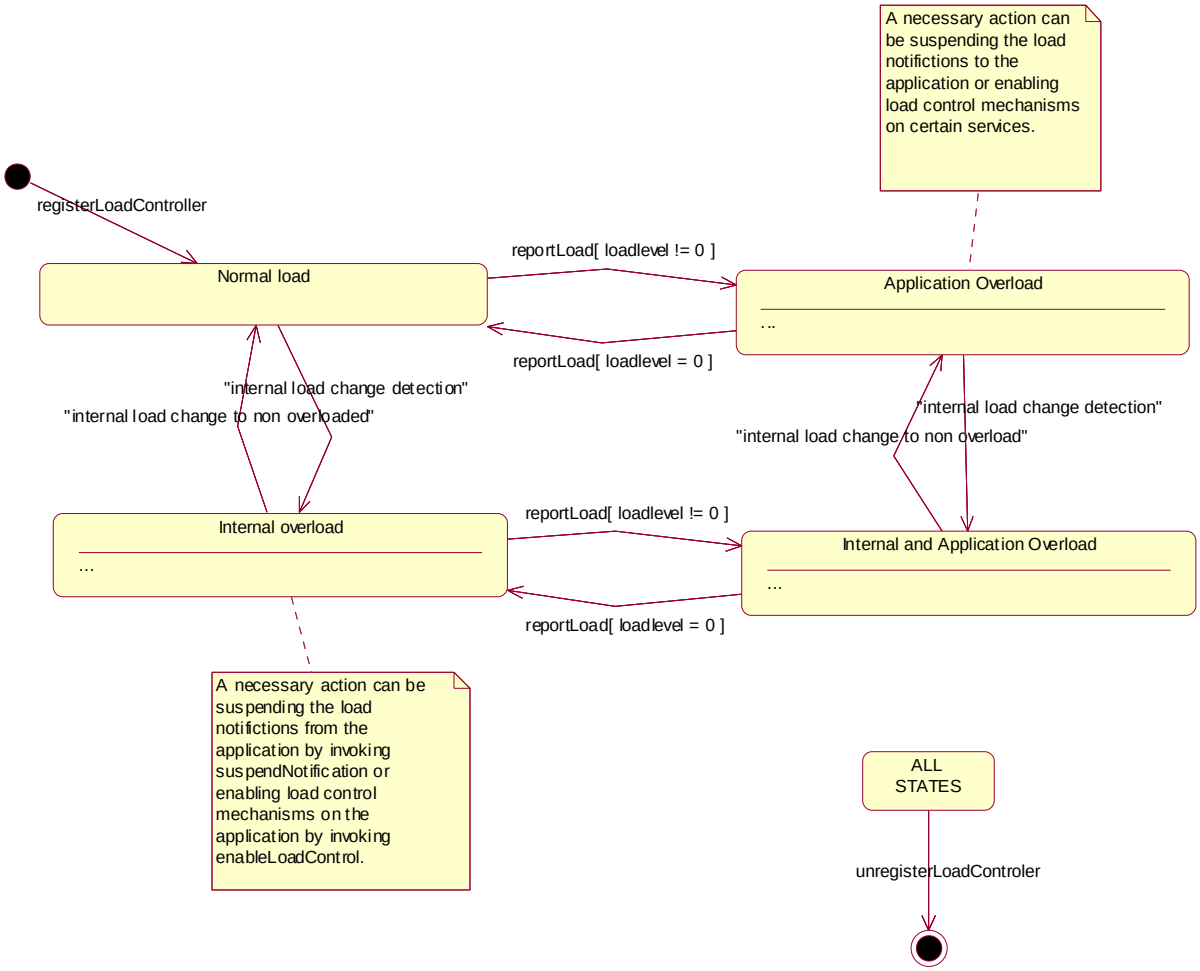


Figure 12: State Transition Diagram for LoadManagerInternal

7.4.3.2.1 Normal load State

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

7.4.3.2.2 Application Overload State

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

7.4.3.2.3 Internal overload State

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

7.4.3.2.4 Internal and Application Overload State

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

7.4.3.3 State Transition Diagrams for IpOAM

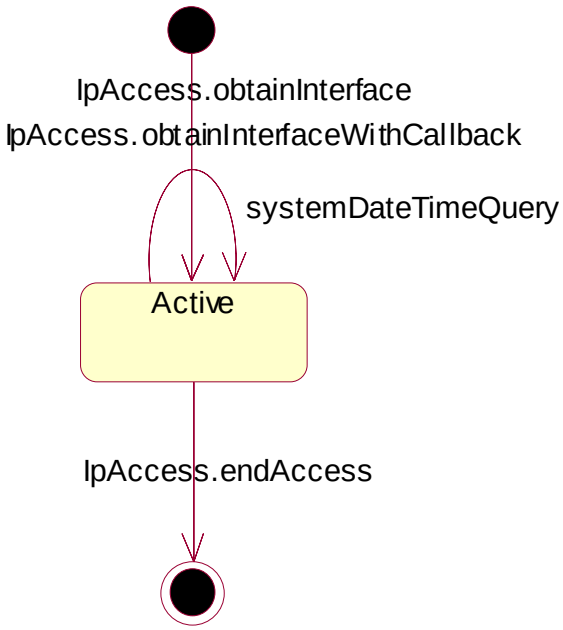


Figure 13: State Transition Diagram for IpOAM

7.4.3.3.1 Active State

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

7.4.3.4 State Transition Diagrams for IpFaultManager

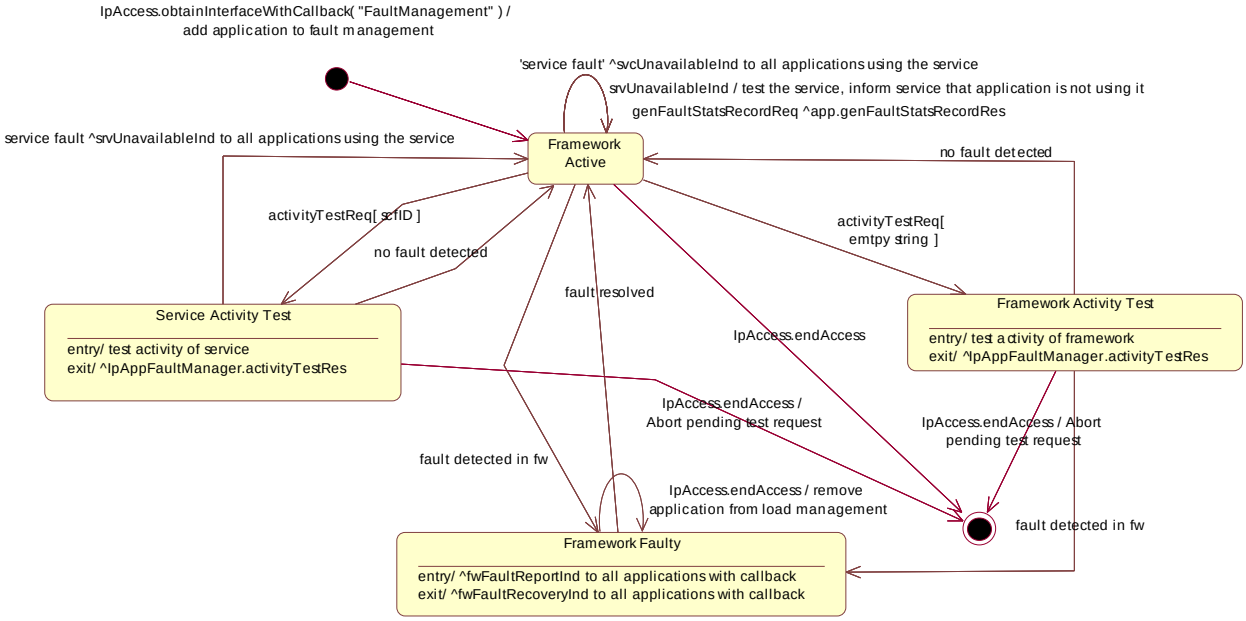


Figure 14: State Transition Diagram for IpFaultManager

7.4.3.4.1 Framework Active State

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

7.4.3.4.2 Framework Faulty State

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

7.4.3.4.3 Framework Activity Test State

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

7.4.3.4.4 Service Activity Test State

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

7.4.4 Event Notification State Transition Diagrams

7.4.4.1 State Transition Diagrams for IpEventNotification

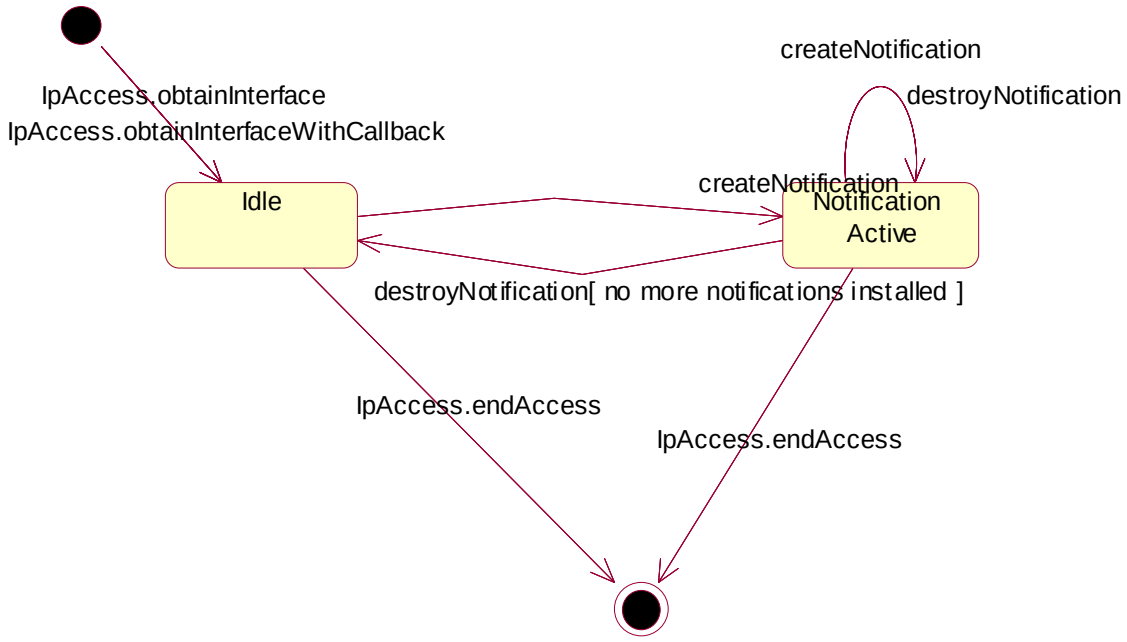


Figure 15: State Transition Diagram for IpEventNotification

8 Framework-to-Enterprise Operator API

In some cases, the client applications (or the enterprise operators on behalf of these applications) must explicitly subscribe to the services before the client applications can access those services. To accomplish this, they use the service subscription function of the Framework for subscribing or un-subscribing to services. Subscription represents a contractual agreement between the enterprise operator and the Framework operator. In general, an entity acting in the role of a *customer/subscriber* subscribes to the services provided by the Framework on behalf of the *users/consumers* of the service.

In this model, the enterprise operators act in the role of *subscriber/customer* of services and the client applications act in the role of *users or consumers* of *services*. The framework itself acts in the role of *retailer* of services. The following examples illustrate these roles:

- Service (to be subscribed): Call Centre Service, Mobility Service, etc.
- Framework Operator: AT&T, BT, etc.
- Enterprise Operator: A Financial institution such as a Bank or Insurance Company, or possibly an Application Service Provider (Such an enterprise has a conformant Subscription Application in its domain which "talks" to its peer in the Framework).
- User/Consumer: Client Applications (or their associated users) in the enterprise domain that use the Call Centre Service or the Mobility Service.

The Service Subscription interface is used by an enterprise operator to subscribe to new services and is required before a client application of the enterprise can use the new service. In general, the service subscription is performed after an off-line negotiation of a set of services and the associated price between the framework operator and the enterprise operator. The service subscription is performed online by the enterprise operator in the frame of an existing off-line negotiated contract between the framework operator and the enterprise. The on-line service subscription function is used for subscriber, client application, and service contract management. The following clause describes a service subscription model.

Subscription Business Model

The following figure shows the subscription business model with respect to the business roles involved in the service subscription process. The subscription process involves the enterprise operator (which acts in the role of service *subscriber*) and the Framework (which acts in the role of *provider* or *retailer* of a service).

Services may be provided to the Enterprise Operator directly by a service provider or indirectly through a retailer, such as the Framework. An enterprise operator represents an organisation or a company which will be hosting client applications. Before a service can be used by the *client applications* in the enterprise operator's domain, subscription to the service must take place. An enterprise operator subscribes to a service by (electronically) signing a contract about the service usage with the Framework, using an on-line subscription interface provided by the Framework. The Framework provides the service according to the service contract. The Enterprise Operator authorises the client application in his/her domain for the service usage. Finally a subscribed service can be used by a particular client application.

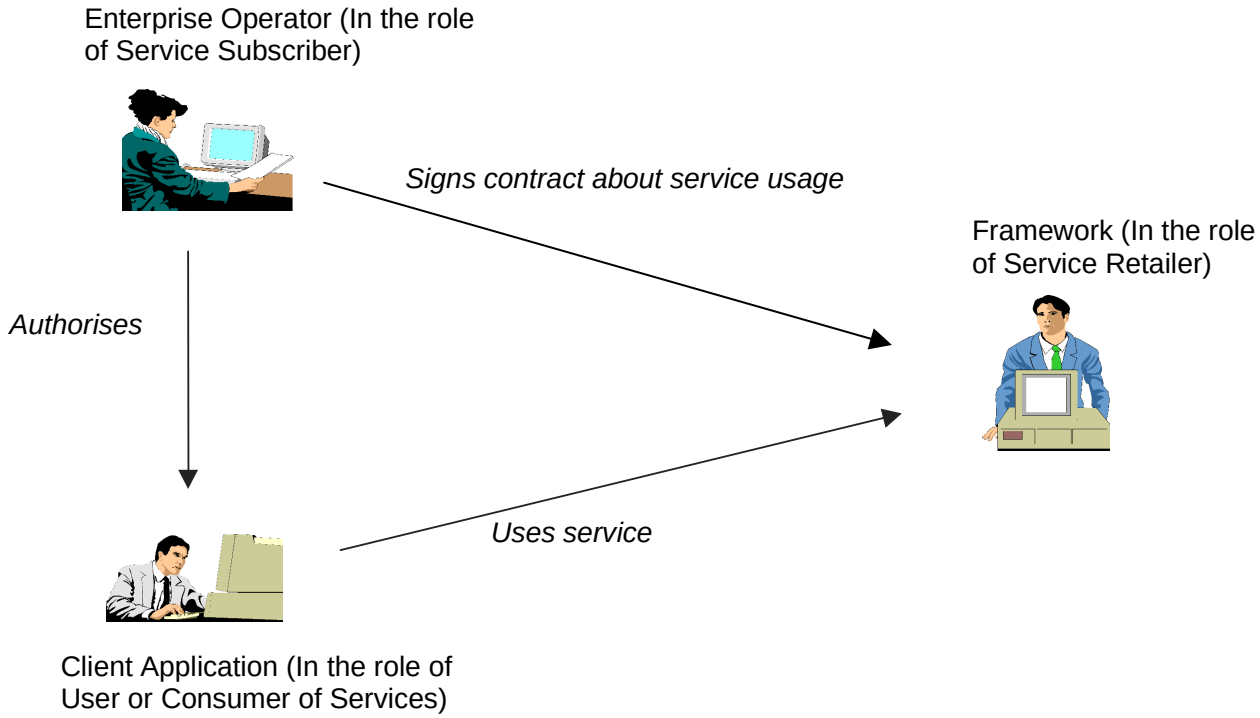


Figure 16: Subscription Business Model

The interfaces between an enterprise operator and the client applications in its domain are outside the scope of this API.

The enterprise operator provides to the Framework the data about the client applications in its domain and the type of services each of them should be allowed access to, using the subscription interfaces offered by the Framework. The Framework provides (to the enterprise operator) the subscription interfaces for subscriber, client application and service contract management. This gives the enterprise operators the capability to dynamically create, modify and delete, in the framework domain, the client applications and service contracts belonging to its domain.

The enterprise operator is represented in the Framework domain as an EntOp *object*. The EntOp object is identified by a unique *ID* and contains the *enterprise operator properties*. The EntOp ID is a unique identifier of an enterprise operator in the Framework domain. It is created by the Framework Operator during the pre-subscription off-line negotiation of services (and their price, etc.) phase. The enterprise operator properties contain information such as the name and address of the enterprise operator (individual or organisation), service charge payment information, etc. The enterprise operator domain has one or more client applications associated with it. The enterprise operator may group a sub-set of client applications in its domain in order to assign the same set of service features to the group. Such a group is called a Subscription Assignment Group (SAG). An enterprise operator may have multiple SAGs in its domain. A SAG relates a client application to the features of a service. A client application may be a member of multiple SAGs, one for each service subscribed for the client application by its enterprise operator.

The enterprise operator subscribes to a number of services by creating a *service contract* with the Framework for each service. Each service subscription is described by a *service contract* which defines the conditions for the service provision. A *service contract* restricts the usage of a service at subscription time. A service contract contains one or more *Service Profile*s, one for each SAG in the enterprise operator domain. A *Service Profile* contains the service parameters which further restrict the corresponding parameters in the service contract in order to adapt the service to the Sag's needs. A service profile is therefore a restriction of the service contract in order to provide restricted service features to a SAG. It is identified by a unique *ID* (within the framework domain) and contains a set of *service properties*, which defines the restricted usage of service allowed for that SAG (and its client applications).

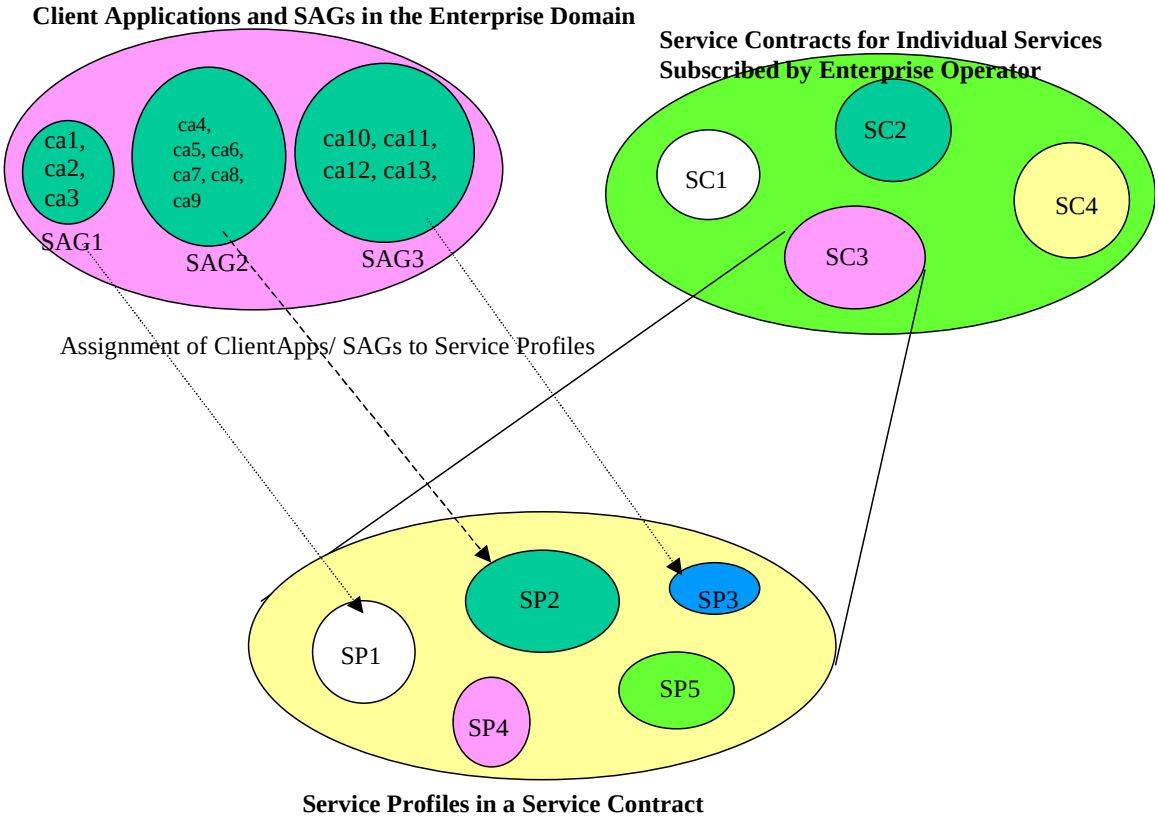


Figure 17: Relationship between Client Applications/SAG, Service Contract and Service Profiles

The *client application* is related to the *enterprise operator* for the usage of a service. The client application is represented in the Framework domain as a *clientApp* object. The *clientApp* object is identified by a unique ID and contains a set of *client application properties* describing the client application relevant information for subscription. Each *client application* is part of at least one *SAG*, which can contain one or more client applications. Each SAG has one s*ervice profile* per service that defines the preferences of the SAG members for the usage of that service. A SAG can have multiple Service Profiles associated with it, one for each service subscribed by the enterprise operator on behalf of the SAG members. The figure above shows the relationship between client application objects, SAGs, service contracts and service profiles.

An enterprise operator may not want to grant all client applications in its domain the same service characteristics and usage permissions. In this case the enterprise operator can group them in a set of SAGs and assign a particular Service Profile to each group. A client application can be assigned to more than one service profile for a given service, as long as the dates within those service profiles do not overlap. The figure below illustrates this. Here the client is assigned to two SAGs. One of these SAGs uses ServiceProfile1 to control access to service 1. The other uses ServiceProfile3 to control access to service 1. If the dates in the two service profiles overlap, as is the case here, then it cannot be determined when the client signs the service agreement which service profile should be used. For example, if the client application signed the service agreement on February the 8th, then it could not be determined which of service profile 1 or service profile 3 would apply. If the dates are not overlapping then there is not a problem with identifying which of the service profiles to use. A SAG may have multiple service profiles, one for each subscribed service, associated with it.

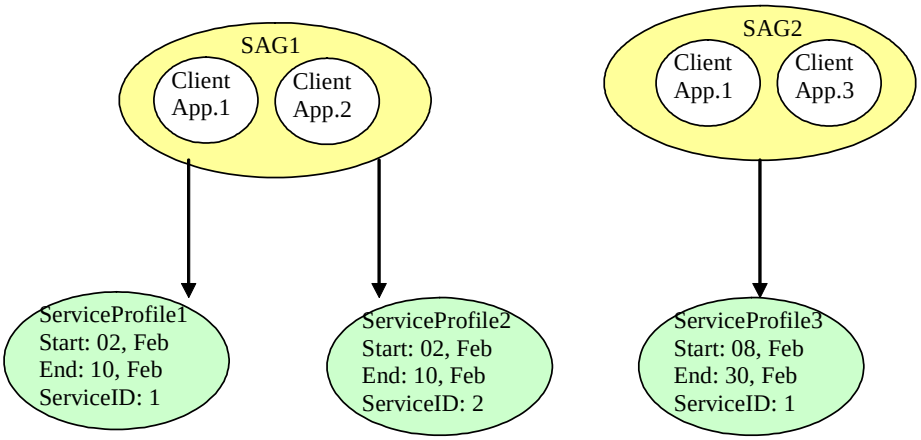


Figure 18: Overlapping date fields in service profiles

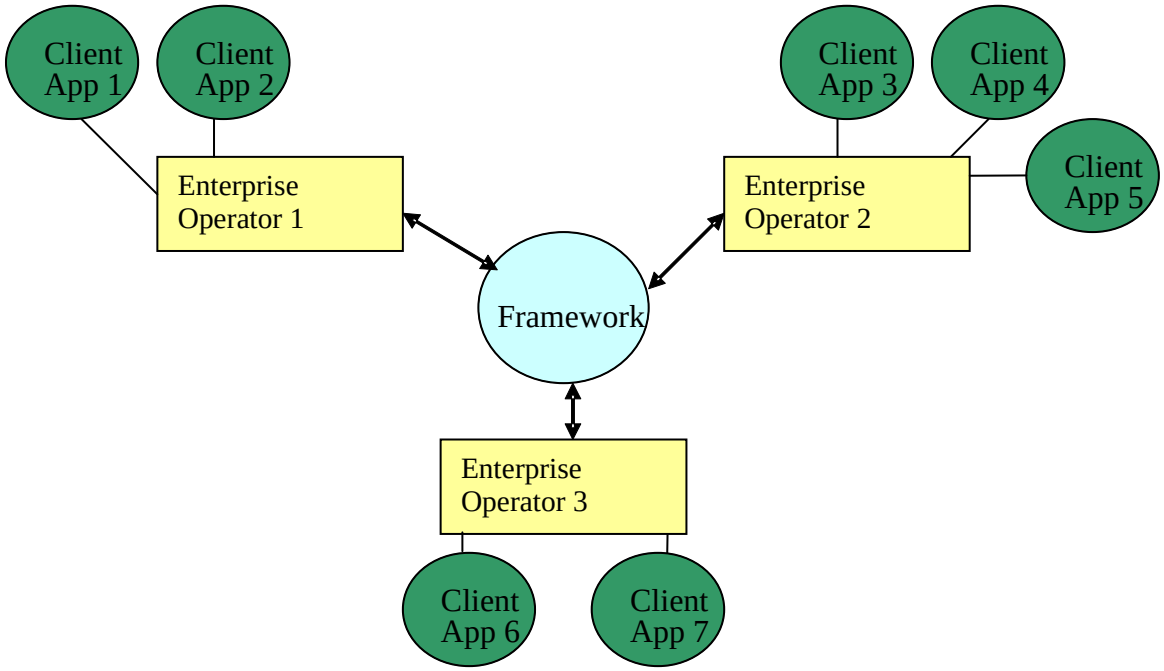


Figure 19: Multiple Enterprise Operators

The figure above illustrates that the framework can offer its services to applications in the domains of many enterprise operators. An enterprise operator could be an Application Service Provider, a corporation, or even the network operator (if the services offered through the framework belong to a single network – it is even possible that the network operator will be the only enterprise operator). It is possible, however, that each service registered with the framework could actually be in a different network. The client application IDs have to be unique within the framework. The framework operator could decide to allocate a block of application IDs to each enterprise operator, or even negotiate with the enterprise operators to provide a set of client application IDs that are meaningful to them.

Service subscription and subscription management requires a careful delineation of subscription-related functions. The service subscription interfaces are classified in the following categories:

- Enterprise Operator Account Management.
- Enterprise Operator Account Query.
- Service Contract Management.
- Service Contract Query.
- Service Profile Management.
- Service Profile Query.
- Client Application Management.
- Client Application Query.

Only the enterprise operator, which is registered and later on authenticated, is allowed to use these interfaces.

8.1 Sequence Diagrams

8.1.1 Service Subscription Sequence Diagrams

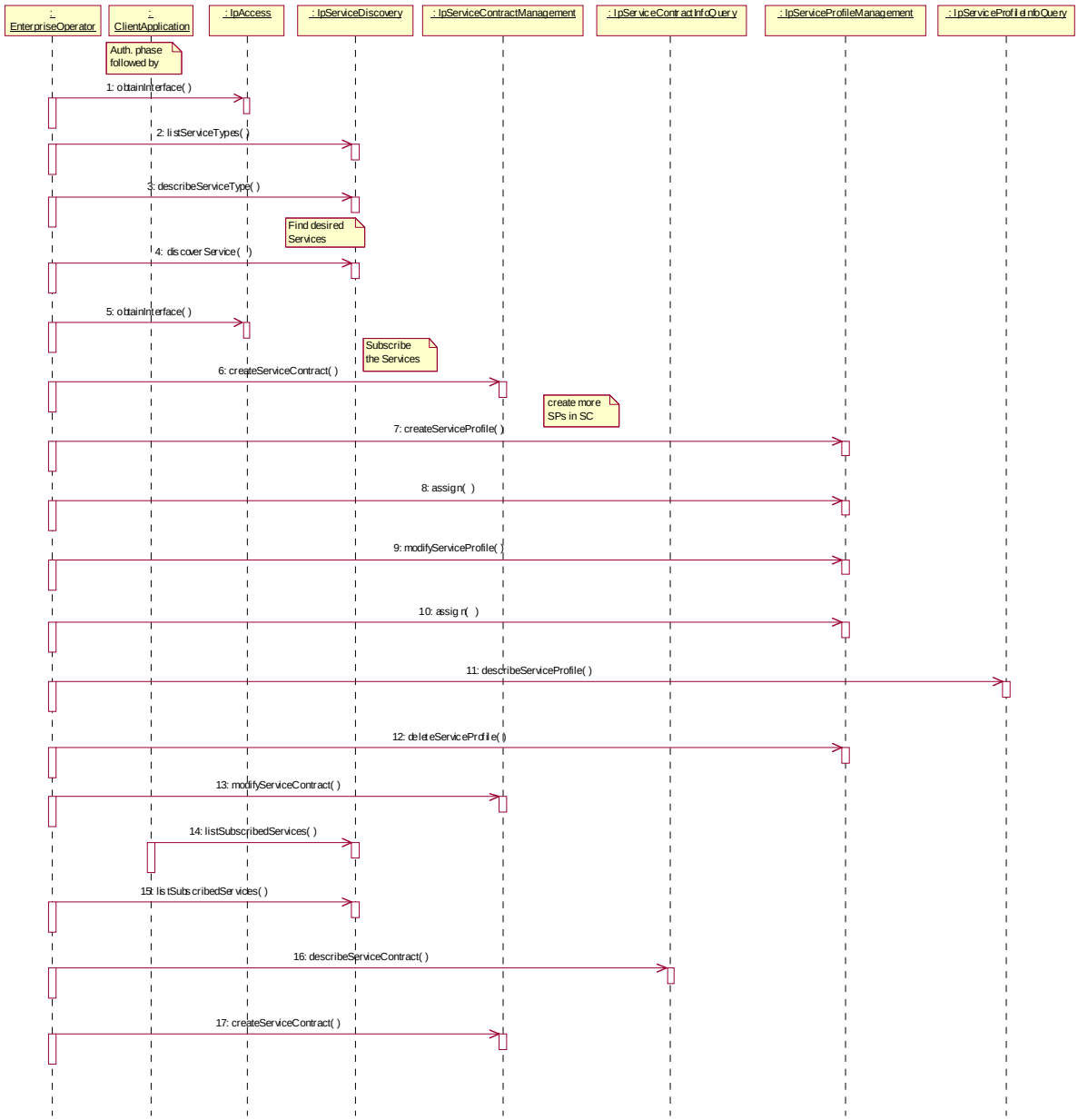
8.1.1.1 Service Discovery and Subscription Scenario

This scenario is shown in the sequence diagram below. Services are subscribed to by the enterprise operator on behalf of the client applications which then use these services. Before an enterprise operator can subscribe to a service, it must have knowledge of the existence of that service in the framework. The enterprise operator discovers the set of services provided by the framework using the IpServiceDiscovery interface. Initially, the enterprise operator obtains a list of service types supported by the framework by invoking listServiceTypes() on IpServiceDiscovery interface. Then it obtains the description of a service type using describeServiceType() to find out the set of properties applicable to a particular service type. Subsequently it invokes discoverService() to discover the services of a given type which supports the desired set of property values. The discoverService() method returns a list of "serviceIDs" and their associated property values. The service discovery phase is followed by the service subscription phase. The enterprise operator uses the IpServiceContractManagement and IpServiceProfileManagement interfaces to perform service subscription.

The enterprise operator invokes the createServiceContract() on IpServiceContractManagement interface to subscribe to a service. Depending upon the Framework Operator's policy, the services may be subscribed by identifying them by their "serviceID" or by their service type. In the former case only the specific service can be used by the enterprise operator and its client applications. In the latter case, all registered services of the given type can be used. The enterprise operator may create multiple service profiles (each of which are a restriction of the service contract) by invoking createServiceProfile() on IpServiceProfileManagement interface and assign each service profile to a different Subscription Assignment Group (SAG), using assign() method. This allows an enterprise operator to assign different service privileges to different client application groups. During the life time of a service contract, the enterprise operator may perform service contract and service profile management functions, such as modifying the service profiles (modifyServiceProfile()) and service contract (modifyServiceContract()), re-assigning the service profiles to a SAG (assign()), obtaining information about a service profile (getServiceProfile()), deleting service profiles (deleteServiceProfile()), etc. These methods may be interleaved in any logical order. The enterprise operator or the client applications, can at any time obtain a list of currently subscribed services by invoking listSubscribedServices() method on the IpServiceDiscovery interface. This method returns a list of serviceIDs of the set of subscribed services.

The service contract ceases to exist after the specified end date. The deleteServiceContract deletes the service contract object held in the framework. It is up to the discretion of the Framework operator to allow the enterprise operator to delete a service contract before its specified end date.

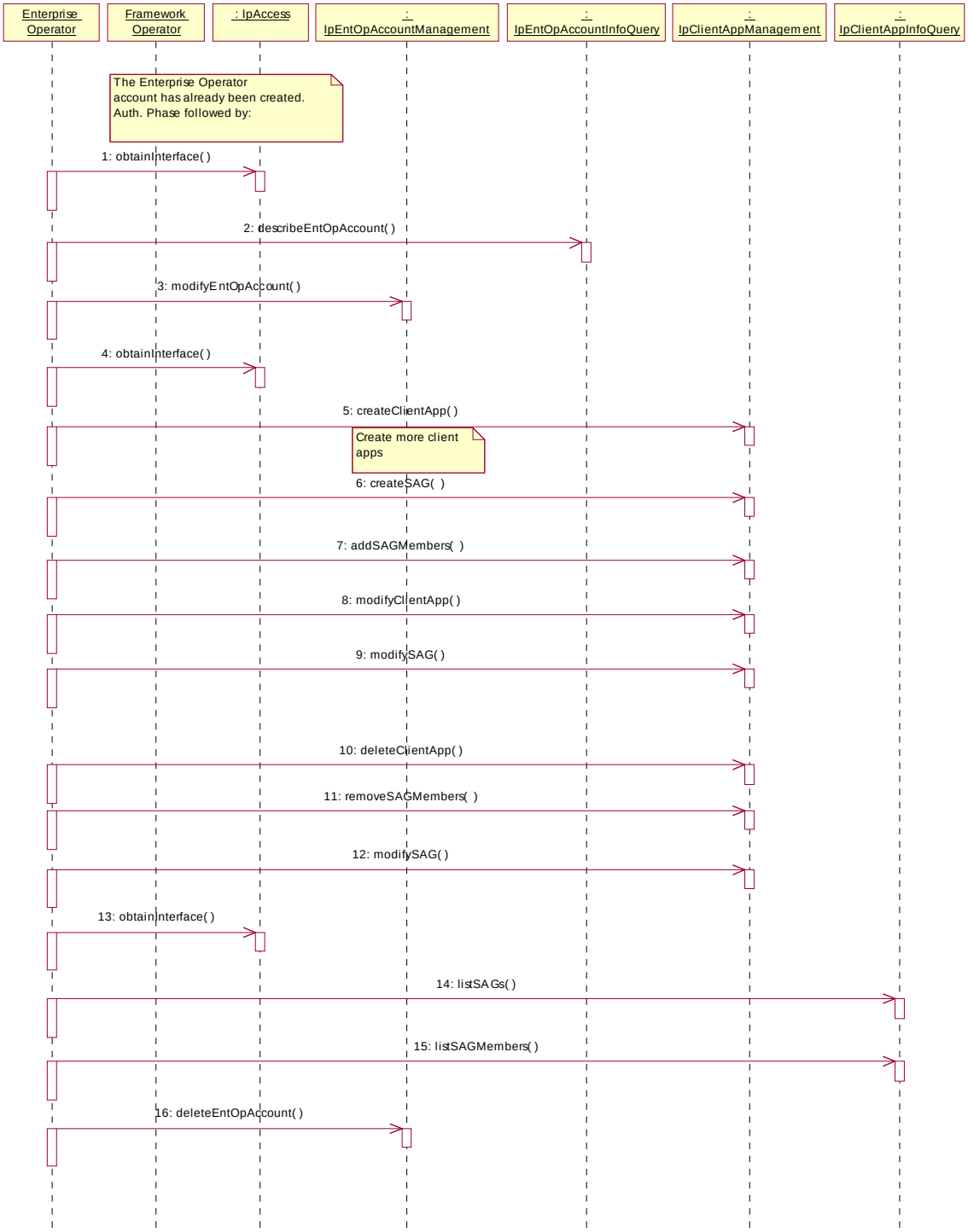
After the service subscription is performed the client applications can access and use the set of subscribed services in addition to the set of freely available services. In order to start a service, the interface reference of the service is required. The discoverService() method or the listSubscribedServices() method, described above, return the "serviceID". The interface reference of the service is obtained in the service access phase. The service access phase begins with the client applications selecting the service, via the selectService() method, and signing a service agreement, via the signServiceAgreement() method. The selectService() method is used by the client application to identify the service that it wants to initiate. The input to the selectService() is the "serviceID" returned by the discoverService() or the listSubscribedServices() and the output is a "serviceToken". The serviceToken is free format text token returned by the framework, which can be used as part of a service agreement. If the service is not subscribed by the enterprise operator, then a "service not subscribed" exception is raised. The signServiceAgreement() is invoked by the client application on the framework to sign an agreement on the service. The input to this method is the service token returned by the selectService() method. The sign service agreement is used as a way of non-repudiation of the intention to use the service by the client application. The successful completion of the signServiceAgreement() returns the interface reference to the service (or to its service manager). The client application can then use this interface reference to start the service.



8.1.1.2 Enterprise Operator and Client Application Subscription Management Sequence Diagram

The first step in the service subscription process is the creation of an account for the enterprise operator. The creation of enterprise operator accounts is performed by the Framework Operator via interfaces outside of the present document. When the enterprise operator's account has been created they are allowed to use the framework. The enterprise operator (acting in the role of service subscriber) can then create accounts within the framework for all of the client applications in its domain. The enterprise operator obtains the reference to the IpEntOpManagement interface by invoking obtainInterface() on the IpAccess interface. The enterprise operator at any time may inspect its subscription account by invoking describeEntOpAccount on the IpEntOpAccountInfoQuery interface and modify the subscriber-related information contained in its subscription account by invoking modifyEntOpAccount() on IpEntOpAccountManagement interface.

An enterprise operator usually has many client applications in its enterprise domain. These client applications must be registered within the framework so that the set of services subscribed to by the enterprise operator (through createServiceContract()) can be assigned to these client applications by associating a service profile (a restriction of service contracts) with a group of client applications, called a Subscription Assignment Group (SAG). In order to create an account for individual client applications, the enterprise operator invokes createClientApp() on IpClientAppManagement interface. The enterprise operator groups a related set of client applications in a SAG so that the same service profile can be assigned to them. The enterprise operator may create an empty SAG by invoking createSAG() on IpClientAppManagement interface. The enterprise operator adds client applications to the newly created SAG by invoking addSAGMembers() on IpClientAppManagement interface. The enterprise operator also performs other client application/SAG management functions such as modifyClientApp(), deleteClientApp(), modifySAG(), listSAGs(), listSAGMembers(), addSAGmembers(), removeSAGmembers()etc. These methods can be interleaved in any logical order. Finally, the enterprise operator (or the framework operator) can delete its subscription account by invoking deleteEntOpAccount() on IpEntOpAccountManagement interface.


Enterprise Operator
Framework Operator
: IpAccess
: IpEntOpAccountManagement
: IpEntOpAccountInfoQuery
: IpClientAppManagement
: IpClientAppInfoQuery
The Enterprise Operator account has already been created. Auth. Phase followed by:
1: obtainInterface( )
2: describeEntOpAccount( )
3: modifyEntOpAccount( )
4: obtainInterface( )
5: createClientApp( )
Create more client apps
6: createSAG( )
7: addSAGMembers( )
8: modifyClientApp( )
9: modifySAG( )
10: deleteClientApp( )
11: removeSAGMembers( )
12: modifySAG( )
13: obtainInterface( )
14: listSAGs( )
15: listSAGMembers( )
16: deleteEntOpAccount( )

8.2 Class Diagrams

Figure 20: Service Subscription Package Overview

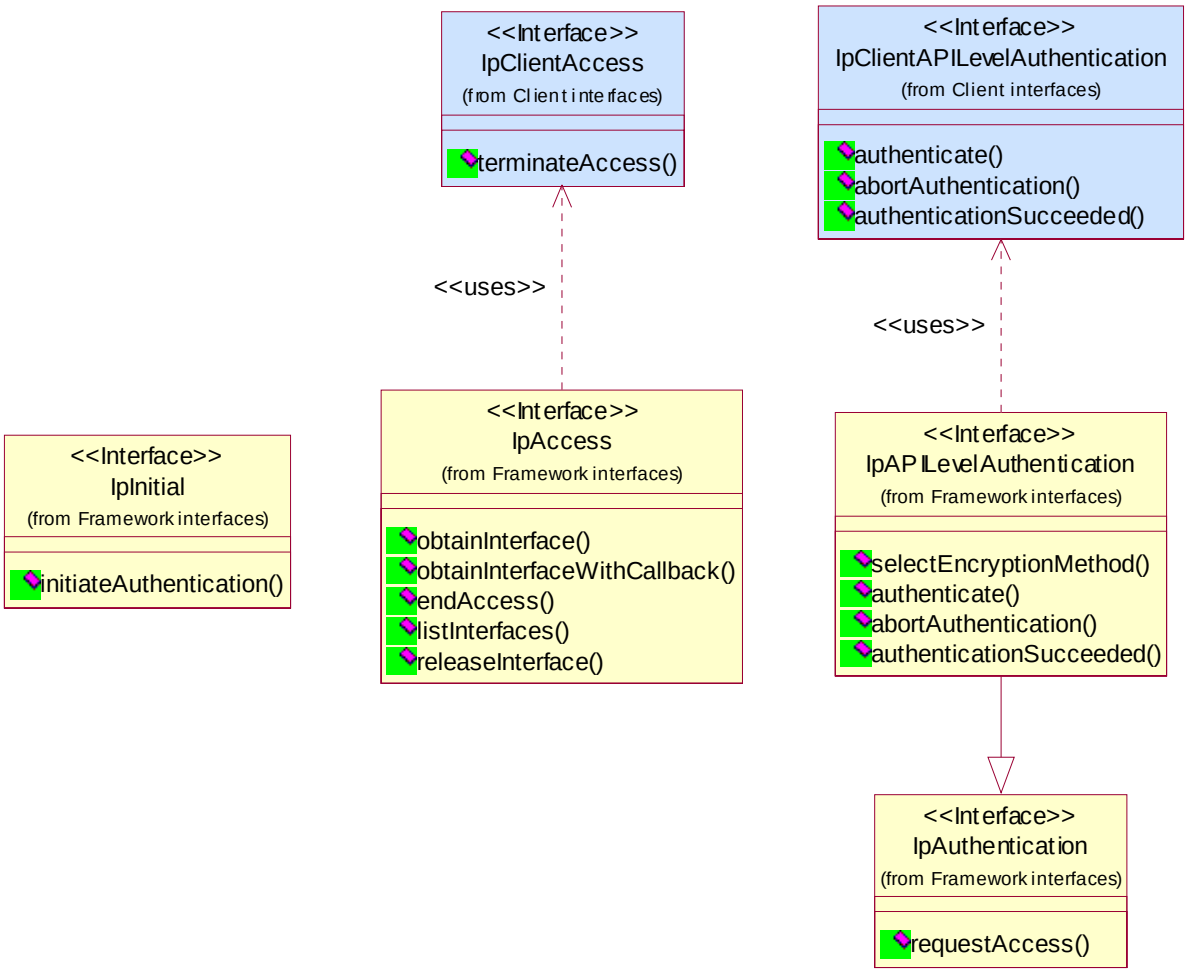


Figure 21: Trust and Security Management Package Overview

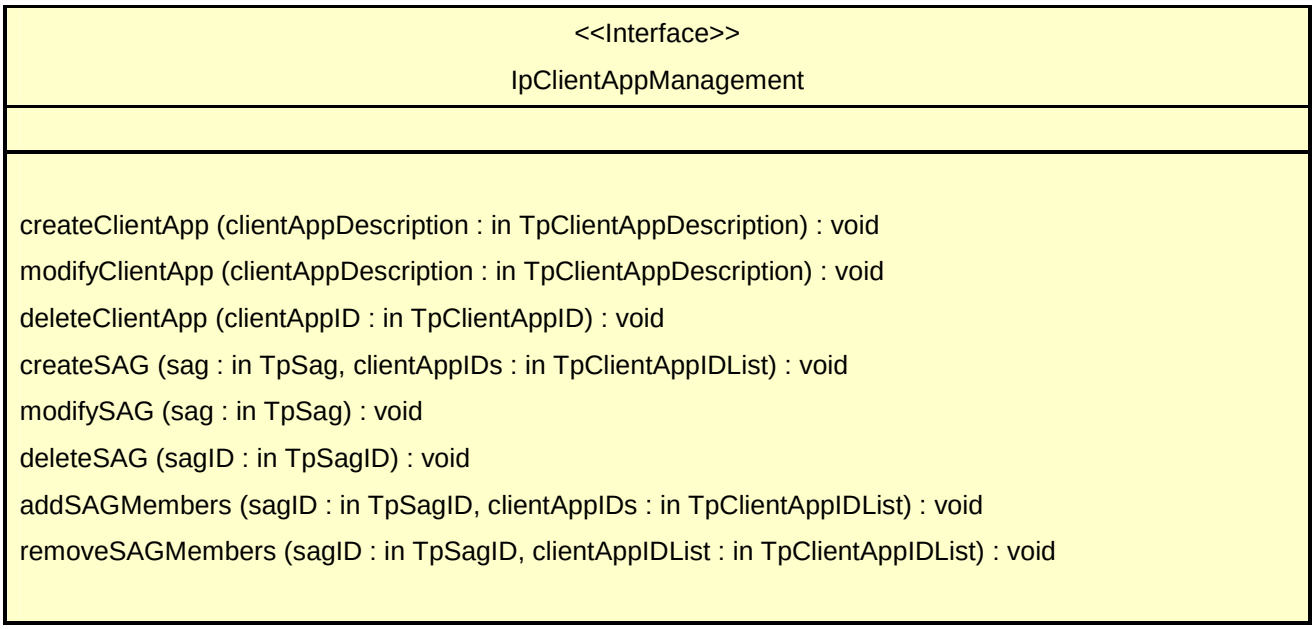
8.3 Interface Classes

8.3.1 Service Subscription Interface Classes

8.3.1.1 Interface Class IpClientAppManagement

Inherits from: IpInterface.

If the enterprise operator wants the client applications in its domain to access the subscribed services in name of the enterprise, then (s)he has to register these client applications in the Framework domain. For this the enterprise operator must use the client application management interface, to which (s)he can subscribe as a privileged user. The client application management interface is intended for cases where an organisation wants to allow several client applications to register with a Framework as service consumers. It allows enterprise operators to dynamically add new client applications and SAGs, delete them and to modify subscription related information concerning the client applications and the SAGs. Client applications use the subscribed services in the enterprise operator's name. The main task of client application management is to: register, modify and delete client applications (Client Application Management), manage groups of client applications, called Subscription Assignment Groups (SAG Management).



Method

createClientApp()

A client application is represented in the Framework domain as a "clientApp object". This method creates a new clientApp object associated with the enterprise operator object. Each clientApp object has a clientApp ID and other subscription related client application's properties stored in it.

Parameters

clientAppDescription : in TpClientAppDescription

The "clientAppDescription" parameter contains the clientApp ID that is to be associated with the newly created clientApp object and the subscription-related "client application properties". The clientApp ID must be a unique ID across framework, if the ID already exists, an exception "P_INVALID_CLIENT_APP_ID" would be raised. The client application properties are a list of name/value pairs. The client application properties are an item for bi-lateral agreement between the enterprise operator and the framework operator.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID

Method

modifyClientApp()

Modify the information contained in an existing clientApp object associated with the enterprise operator. An exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

clientAppDescription : in TpClientAppDescription

The "clientAppDescription" parameter contains the modified client application information. If the clientApp ID does not exist, an exception "P_INVALID_CLIENT_APP_ID" would be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID

Method
deleteClientApp()

Delete the specified client application associated with the enterprise operator. If the client application currently has an access session with the framework then this will be terminated, along with any service instances it may have created. An exception of "P_TASK_REFUSED" will be raised if a non-associated enterprise operator invokes this method.

Parameters

clientAppID : in TpClientAppID

The "clientAppID" parameter identifies the client application that is to be deleted. If the clientAppID does not exist, a "P_INVALID_CLIENT_APP_ID" exception will be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID

Method
createSAG()

Create a new SAG associated with the enterprise operator. The SAG object is identified by a SAG - ID and contains SAG - specific description.

Parameters

sag : in TpSag

The "sag" parameter contains the SAG-ID and SAG-specific description. This sagID is particular to the SAG, and must be unique across framework. If the sagID supplied already exists, an exception of type "P_INVALID_SAG_ID" would be raised.

clientAppIDs : in TpClientAppIDList

The "clientAppIDs" parameter contains the list of client application IDs that are to be associated with the newly created SAG.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID, P_INVALID_SAG_ID

Method
modifySAG()

Modify the description of an existing SAG associated with the enterprise operator. An exception of "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

sag : in TpSag

The "sag" parameter contains the modified SAG-specific description. If the SAG ID does not exist, an exception "P_INVALID_SAG_ID" would be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SAG_ID

Method
deleteSAG()

Delete an existing SAG. Only the enterprise operator associated with the SAG is allowed to delete it, an exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

sagID : in TpSagID

The "sagID" parameter identifies the SAG that is to be deleted. If the SAG ID does not exist, an exception "P_INVALID_SAG_ID" is raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SAG_ID

Method

addSAGMembers()

Add the specified client applications to the specified SAG associated with the enterprise operator. Only the enterprise operator associated with the SAG is allowed to assign members to it, an exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

sagID : in TpSagID

The "sagID" parameter identifies the SAG object to which the client applications are to be added. If the SAG ID does not exist, an exception "P_INVALID_SAG_ID" would be raised.

clientAppIDs : in TpClientAppIDList

The "clientAppIDs" parameter contains the list of the clientApp IDs that are to be added to the specified SAG. The clientApp objects are first created using the createClientApp() method. If one or all of the client application IDs in the list does not exist, an exception "P_INVALID_CLIENT_APP_ID" would be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID, P_INVALID_SAG_ID

Method

removeSAGMembers()

Delete specified client applications from the specified SAG object of the enterprise operator. Only the enterprise operator associated with the SAG is allowed to remove members from it, an exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

sagID : in TpSagID

The "sagID" parameter identifies the SAG from which the client applications are to be removed. If the SAG ID does not exist, an exception "P_INVALID_SAG_ID" would be raised.

clientAppIDList : in TpClientAppIDList

The "clientAppIDList" parameter contains the list of the clientApp IDs that are to be removed from the specified SAG. If one or all of the client application IDs in the list does not exist, an exception "P_INVALID_CLIENT_APP_ID" would be raised.

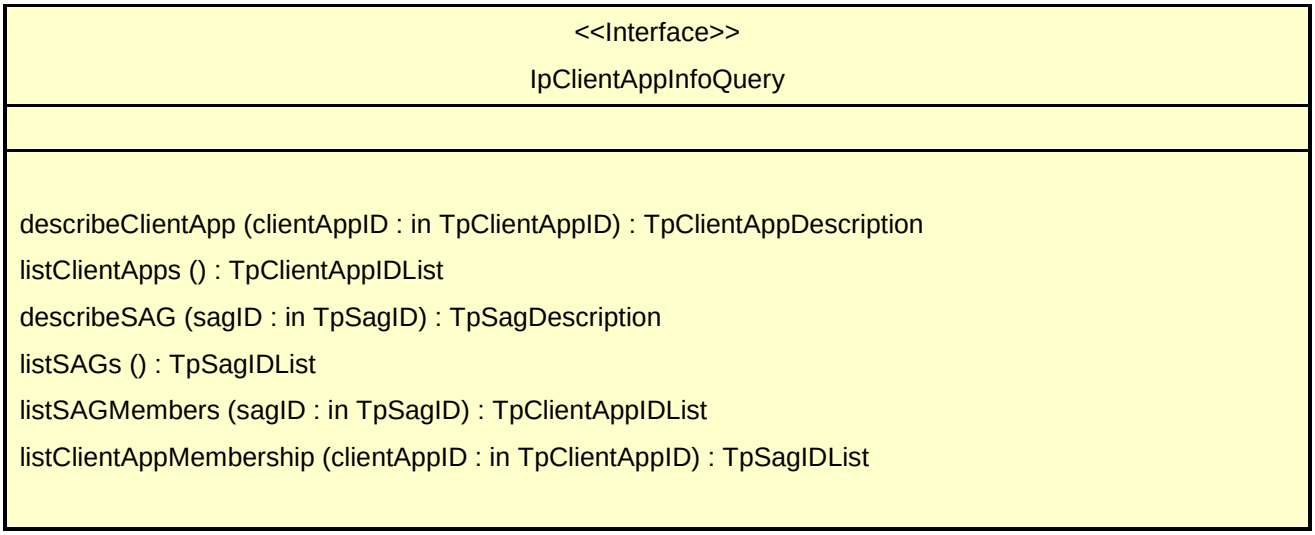
Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID, P_INVALID_SAG_ID

8.3.1.2 Interface Class IpClientAppInfoQuery

Inherits from: IpInterface.

This interface is used by the enterprise operator to list the client applications and the SAGs in its domain and to obtain information about them.



Method
`describeClientApp()`

Query information about the specified client application of the enterprise operator.

Returns <clientAppDescription> : The "clientAppDescription" parameter contains the clientApp description.

Parameters

`clientAppID : in TpClientAppID`

The "clientAppID" parameter identifies the clientApp object whose description is requested.

Returns

`TpClientAppDescription`

Raises

`TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID`

Method
`listClientApps()`

Get a list of all client applications belonging to an enterprise operator.

Returns <clientAppIDs> : The "clientAppIDs" parameter identifies the list of client applications in the enterprise operator domain.

Parameters

No Parameters were identified for this method.

Returns

TpClientAppIDList

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method

describeSAG()

Query information about the specified SAG associated with the enterprise operator.

Returns <SagDescription> : The "sagDescription" parameter returns the SAG-specific description.

Parameters

sagID : in TpSagID

The "sagID" parameter identifies the SAG whose description is required.

Returns

TpSagDescription

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SAG_ID

Method

listSAGs()

Get a list of all SAGs associated with an enterprise operator.

Returns <SagIDList> : The "sagIDList" parameter returns the list of the identifiers of the SAGs associated with the enterprise operator.

Parameters

No Parameters were identified for this method.

Returns

TpSagIDList

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method

listSAGMembers()

Get a list of all client applications associated with the specified SAG.

Returns <clientAppIDList> : The "clientAppIDList" parameter returns the list of the client applications associated with the SAG.

Parameters

sagID : in TpSagID

The "sagID" parameter identifies the SAG whose clientAppID list is required.

Returns

TpClientAppIDList

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SAG_ID

Method

listClientAppMembership()

Obtain a list of the SAGs of which the specified client application is a member.

Returns <sags> : The SAGs of which the client application is a member.

Parameters

clientAppID : in TpClientAppID

The "clientAppID" parameter identifies the clientApp object whose membership details are requested.

Returns

TpSagIDList

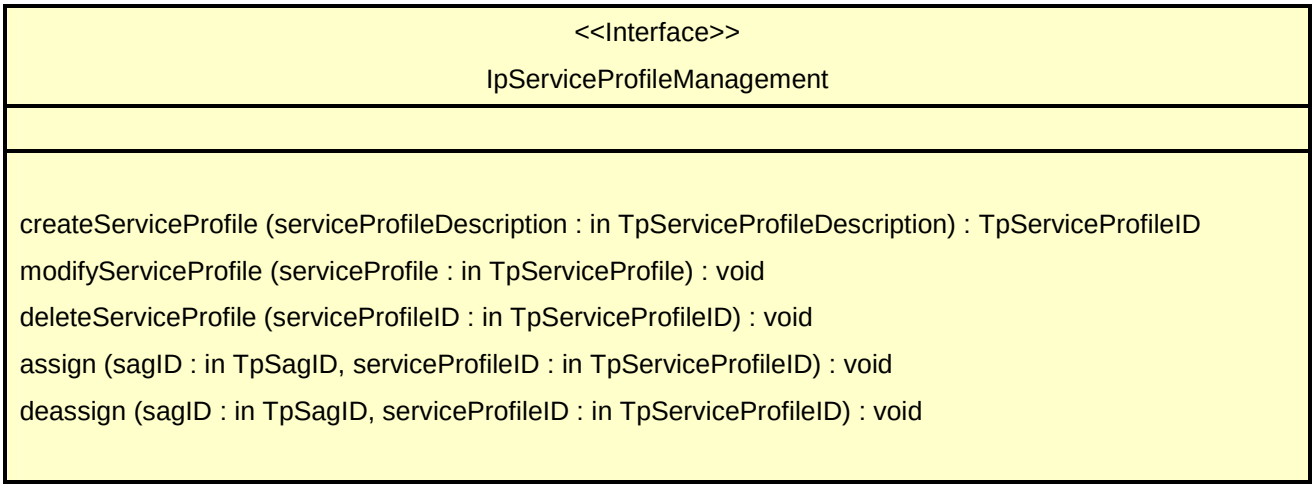
Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_CLIENT_APP_ID

8.3.1.3 Interface Class IpServiceProfileManagement

Inherits from: IpInterface.

This interface is used by the enterprise operator for the management of Service Profiles, which are defined for every subscribed service, and to assign/de - assign the Service Profiles to SAGs.



Method

createServiceProfile()

Creates a new Service Profile for the specified service contract. The service properties within the service profile restrict the service to meet the client application requirements. A Service Profile is a restriction of the corresponding service contract. When the description has been verified, a service profile ID will be generated.

Returns <serviceProfileID> : The service profile ID, generated by the framework, will be used to uniquely identify the service profile within the framework.

Parameters

serviceProfileDescription : in TpServiceProfileDescription

The "serviceProfile" parameter is a structured data type, which contains a subset of the associated service contract information and which may further restrict the value ranges of the service subscription properties.

Returns

TpServiceProfileID

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_PROFILE_ID

Method

modifyServiceProfile()

Modifies the specified Service Profile associated with the enterprise operator. Only the enterprise operator associated with the particular service profile is allowed to modify it, an exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

serviceProfile : in TpServiceProfile

The modified Service Profile. If the serviceProfileID specified in the serviceProfile parameter does not exist, an exception "P_INVALID_SERVICE_PROFILE_ID" would be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_PROFILE_ID

Method

deleteServiceProfile()

Deletes the specified Service Profile. If there are any service instances running that are governed by this profile then they will be terminated. Only the enterprise operator associated with the particular service profile is allowed to delete it, a "P_TASK_REFUSED" exception will be raised if a non-associated enterprise operator invokes this method.

Parameters

serviceProfileID : in TpServiceProfileID

The "serviceProfileID" parameter identifies the Service Profile that is to be deleted. If the serviceProfileID does not exist, a "P_INVALID_SERVICE_PROFILE_ID" exception will be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_PROFILE_ID

Method

assign()

Assign a Service Profile to the specified SAG. Only the enterprise operator associated with the serviceProfileID is allowed to assign it to a SAG, an exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

sagID : in TpSagID

The "sagID" parameter identifies the SAG to which Service Profile is to be assigned. If the SAG ID does not exist, an exception "P_INVALID_SAG_ID" would be raised.

serviceProfileID : in TpServiceProfileID

The "serviceProfileID" parameter identifies the Service Profile that is to be assigned to the SAG. If the serviceProfileID does not exist, an exception "P_INVALID_SERVICE_PROFILE_ID" would be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SAG_ID, P_INVALID_SERVICE_PROFILE_ID

Method

deassign()

De-assign the Service Profile from the specified SAG. Because only the enterprise operator associated with the serviceProfileID is allowed to deassign it from a SAG, an exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

sagID : in TpSagID

The "sagID" parameter identifies the SAG whose Service Profile is to be de-assigned. If the SAG ID does not exist, an exception "P_INVALID_SAG_ID" would be raised.

serviceProfileID : in TpServiceProfileID

The "serviceProfileID" parameter identifies the Service Profile that is to be de-assigned. If the serviceProfileID does not exist, an exception "P_INVALID_SERVICE_PROFILE_ID" would be raised.

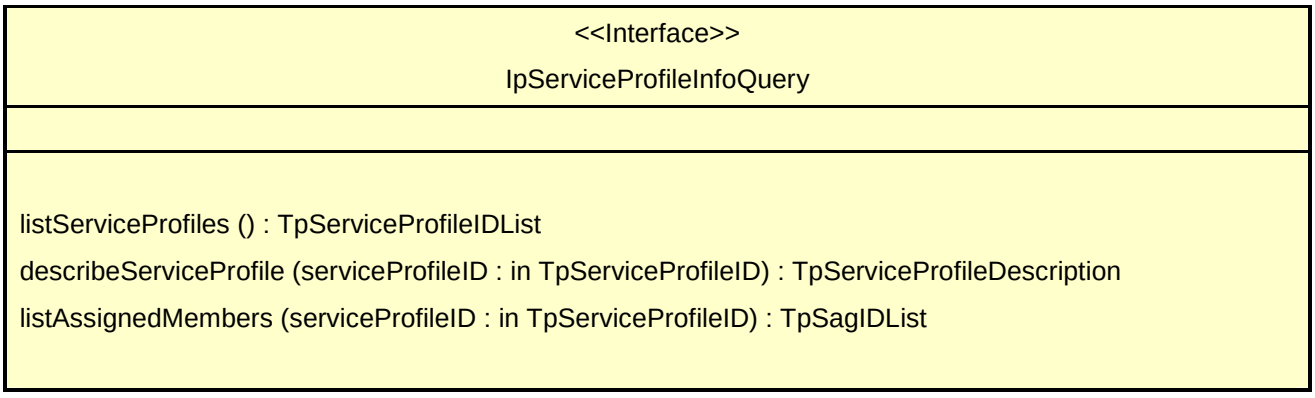
Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SAG_ID, P_INVALID_SERVICE_PROFILE_ID

8.3.1.4 Interface Class IpServiceProfileInfoQuery

Inherits from: IpInterface.

This interface is used by the enterprise operator to obtain information about individual Service Profiles, to find out which SAGs are assigned to a given Service Profile, and to find out what Service Profile is associated with a given client application or SAG.



Method

listServiceProfiles()

Get a list of all service profiles created by the enterprise operator.

Returns <serviceProfileIDList> : The "serviceProfileIDList" is a list of the service profiles associated with the enterprise operator.

Parameters
No Parameters were identified for this method.

Returns

TpServiceProfileIDList

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method
describeServiceProfile()

Query information about a single service profile.

Returns <serviceProfileDescription> : The "serviceProfileDescription" parameter is a structured data type which contains a description for the specified service profile.

Parameters

serviceProfileID : in TpServiceProfileID

The "serviceProfileID" parameter identifies the Service Profile whose description is being requested.

Returns

TpServiceProfileDescription

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_PROFILE_ID

Method
listAssignedMembers()

Get a list of SAGs assigned to the specified service profile.

Returns <sagIDList> : The "sagIDs" parameter is the list of the SAG IDs that are assigned to the specified service profile.

Parameters

serviceProfileID : in TpServiceProfileID

The "serviceProfileID" parameter identifies the Service Profile. If the serviceProfileID is not recognised by the framework, an exception "P_INVALID_SERVICE_PROFILE_ID" would be raised.

Returns

TpSagIDList

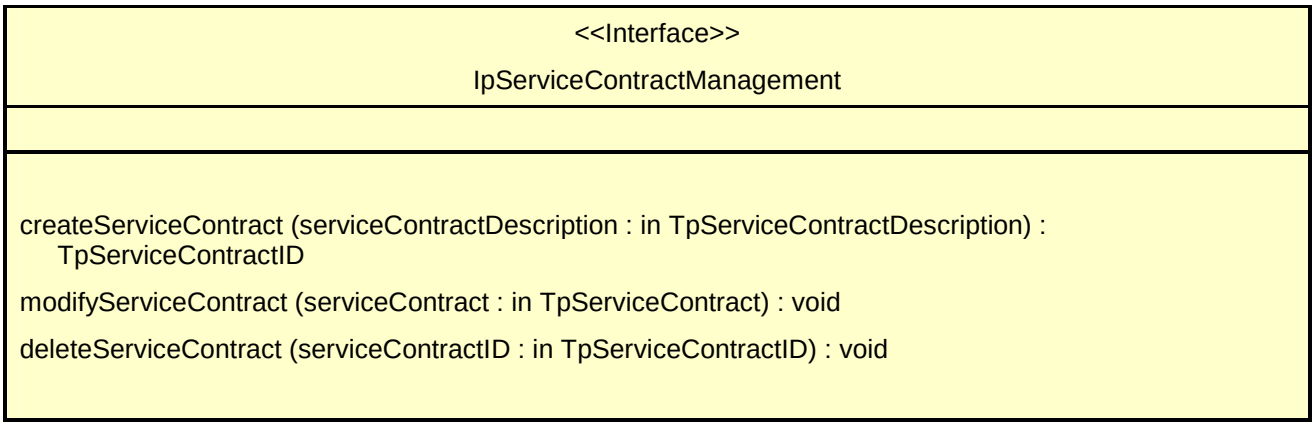
Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_PROFILE_ID

8.3.1.5 Interface Class IpServiceContractManagement

Inherits from: IpInterface.

The enterprise operator uses this interface for service contract management, such as create, modify, and delete service contracts.



Method

createServiceContract()

Create a new service contract for an enterprise operator. The enterprise operator provides the service contract description. This contract should conform to the previously negotiated high - level agreement (regarding the services, their usage and the price, etc.), if any, between the enterprise operator and the framework operator, otherwise the appropriate exception is raised by the framework. When the description has been validated, a service contract ID will be generated.

Returns <serviceContractID> : The service contract ID will be used to uniquely identify the service contract within the framework.

Parameters

serviceContractDescription : in TpServiceContractDescription

The "serviceContractDescription" parameter provides the information contained in the service contract. The service contract is a structured data type, which contains the following information:

a. information about the service requestor, i.e. the enterprise operator,
b. information about the billing contact (person),
c. service start date,
d. service end date,
e. service type (e.g. obtained from listServiceType() method),
f. service ID (e.g. obtained from discoverService() method). For certain services, service type information is sufficient and service ID may not be required. This implies that any service of the type specified above is subscribed and hence accessible to the enterprise operator or to its client applications.
g. list of service subscription properties and their value ranges (service profiles further restrict these value ranges)

Returns

TpServiceContractID

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_INVALID_SERVICE_CONTRACT_ID

Method

modifyServiceContract()

Modify an existing service contract. The service contract can be modified only within the context of a pre-existing off-line negotiated high-level agreement between the enterprise operator and the framework operator. Only the enterprise operator associated with the serviceContract is allowed to modify it, an exception "P_TASK_REFUSED" would be raised if a non-associated enterprise operator invokes this method.

Parameters

serviceContract : in TpServiceContract

The "serviceContract" parameter provides the modified service contract. If the serviceContractID does not exist, an exception "P_INVALID_SERVICE_CONTRACT_ID" would be raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_INVALID_SERVICE_CONTRACT_ID

Method

deleteServiceContract()

Delete an existing service contract. All the Service Profiles associated with the service contract are also deleted. If there are any service instances running that are governed by this contract, or any of the profiles associated with it, then they will be terminated. Only the enterprise operator associated with the serviceContract is allowed to delete it, a "P_TASK_REFUSED" exception will be raised if a non-associated enterprise operator invokes this method.

Parameters

serviceContractID : in TpServiceContractID

The "serviceContractID" parameter identifies the service contract that the enterprise operator wishes to delete. If the serviceContractID does not exist, a "P_INVALID_SERVICE_CONTRACT_ID" exception will be raised.

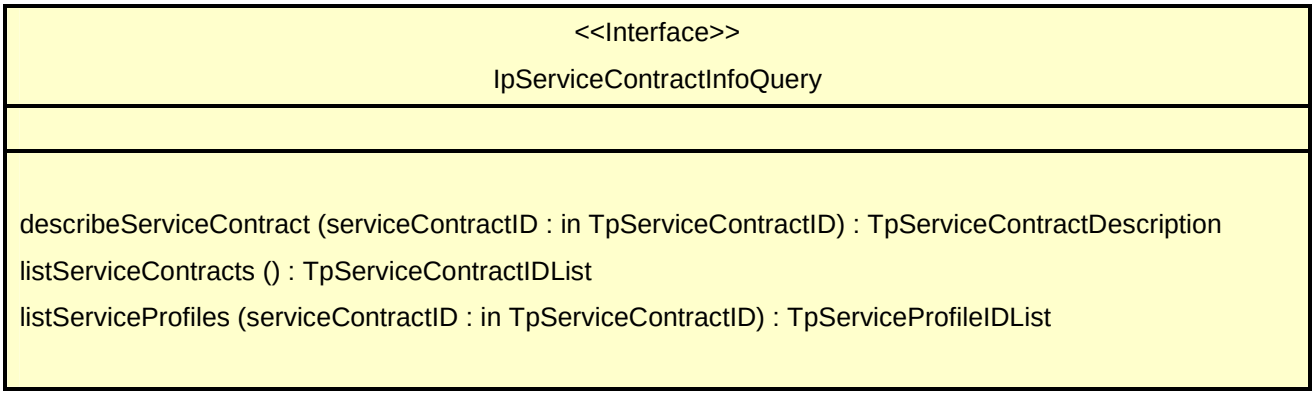
Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_CONTRACT_ID

8.3.1.6 Interface Class IpServiceContractInfoQuery

Inherits from: IpInterface.

The enterprise operator uses this interface to query information about a given service contract.



Method

describeServiceContract()

Query information about the specified service contract. The enterprise operator invokes this operation to obtain information that is stored in the specified service contract. The enterprise operator can only obtain information about the service contracts that it has created.

Returns <serviceContractDescription> : The "serviceContract" parameter contains the description for the specified service contract.

Parameters

serviceContractID : in TpServiceContractID

The "serviceContractID" parameter identifies the service whose description is being requested.

Returns

TpServiceContractDescription

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_CONTRACT_ID

Method

listServiceContracts()

Returns a list of the IDs of service contracts created by the Enterprise Operator.

Returns <serviceContractIDs> : The "serviceContractIDs" parameter will contain a list of IDs for service contracts that the enterprise operator has created.

Parameters

No Parameters were identified for this method.

Returns

TpServiceContractIDList

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method

listServiceProfiles()

The enterprise operator invokes this operation to obtain a list of service profiles that are associated with a particular service contract.

Returns <serviceProfileIDs> : The "serviceContractIDs" parameter contains the service profile members associated with a particular service contract.

Parameters

serviceContractID : in TpServiceContractID

The "serviceContractID" parameter identifies the service contract. If the serviceContractID is not recognised by the framework, an exception "P_INVALID_SERVICE_CONTRACT_ID" would be raised.

Returns

TpServiceProfileIDList

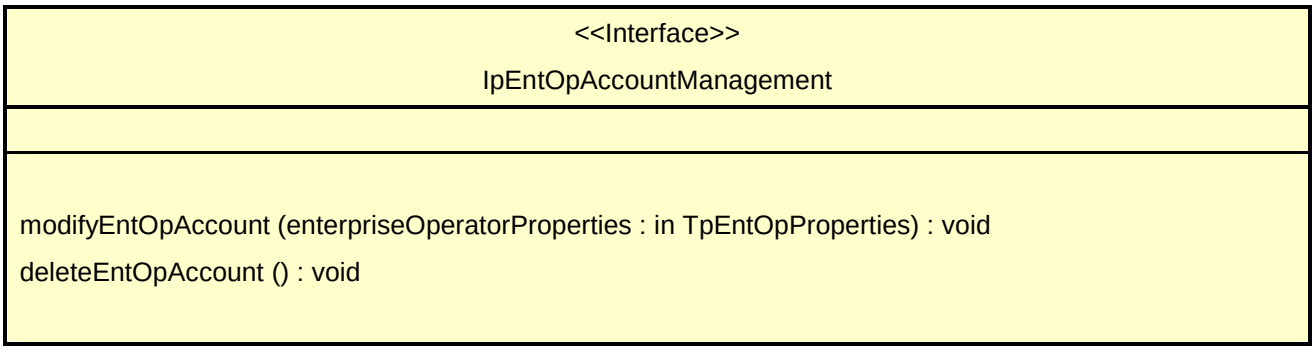
Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_CONTRACT_ID

8.3.1.7 Interface Class IpEntOpAccountManagement

Inherits from: IpInterface.

The enterprise operator, in the role of the service subscriber, uses this interface for the management of enterprise operator subscription accounts, such as modify and delete enterprise operator accounts. The EntOpID will be decided in an off-line agreement between the FW operator and the EntOp, as the EntOp may require the ID to be something more meaningful than a random number. The EntOp account, consisting of the EntOpID, along with the list of valid properties and their modes and prescribed ranges, will be entered via a FW operator interface that is currently outside the scope of the API.



Method

modifyEntOpAccount()

Modification of the enterprise operator information contained in the enterprise operator object.

Parameters

enterpriseOperatorProperties : in TpEntOpProperties

The "enterprise operator properties" parameter conveys the modified/populated information about the enterprise operator. The values of the "enterprise operator properties" can only be modified within the prescribed range, as negotiated earlier (an off-line process) between the enterprise operator and the framework operator, otherwise a P_INVALID_PROPERTY exception is raised.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_PROPERTY

Method

deleteEntOpAccount()

Deletes the specified enterprise operator object. Deletion of the enterprise operator object cannot be performed until the enterprise operator has deleted all the service contracts (and the Service Profiles) associated with it. An attempt to delete the enterprise operator account will result in a P_TASK_REFUSED exception if there are outstanding service contracts (and service profiles).

Parameters

No Parameters were identified for this method.

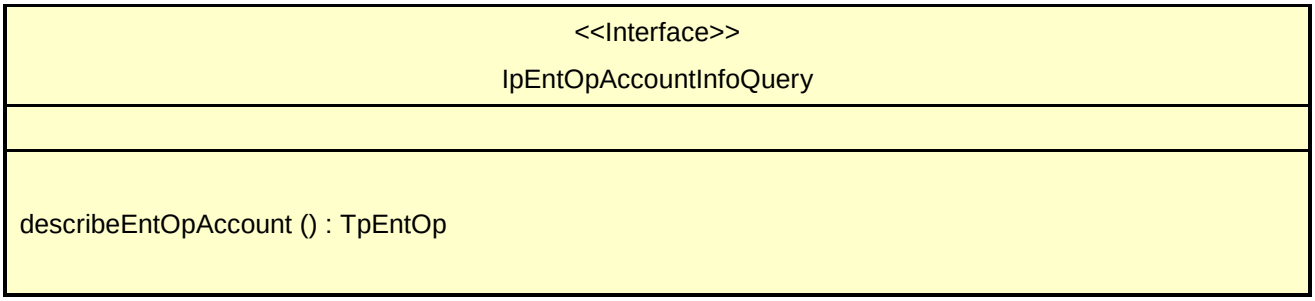
Raises

TpCommonExceptions, P_ACCESS_DENIED

8.3.1.8 Interface Class IpEntOpAccountInfoQuery

Inherits from: IpInterface.

This interface is used by the enterprise operator to query information related to its own subscription account as held within the framework.



Method

`describeEntOpAccount()`

Query information about the enterprise operator. The enterprise operator invokes this operation to find out what information about itself is stored in the enterprise operator account object within the Framework.

Returns <enterpriseOperator> : The "enterpriseOperator" parameter conveys the information stored in the EntOp object about the enterprise operator. It contains the unique "enterprise operator ID" followed by a list of "enterprise operator properties". The enterprise operator properties is a list of name/value pairs which provide enterprise operator related information such as the name, organisation, address, phone, e-mail, fax, payment method (credit card, bank account), etc. to the framework.

Parameters

No Parameters were identified for this method.

Returns

`TpEntOp`

Raises

`TpCommonExceptions, P_ACCESS_DENIED`

8.4 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 client are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the client also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

8.4.1 Service Subscription State Transition Diagrams

There are no State Transition Diagrams defined for Service Subscription.

9 Framework-to-Service API

9.1 Sequence Diagrams

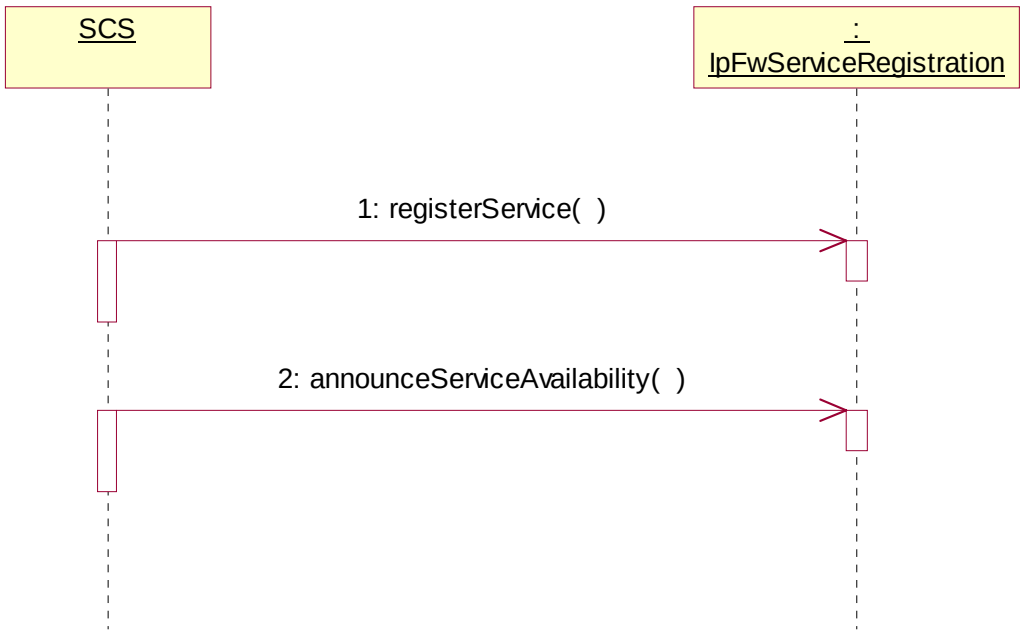
9.1.1 Service Discovery Sequence Diagrams

No Sequence Diagrams exist for Service Discovery.

9.1.2 Service Registration Sequence Diagrams

9.1.2.1 New SCF Registration

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



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 pair of (ServicePropertyName, ServicePropertyValueList).

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

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

The following output parameter results from service registration:

- out serviceID.

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

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

2: Registration: second step - announce service availability

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

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

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

- in serviceID.

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

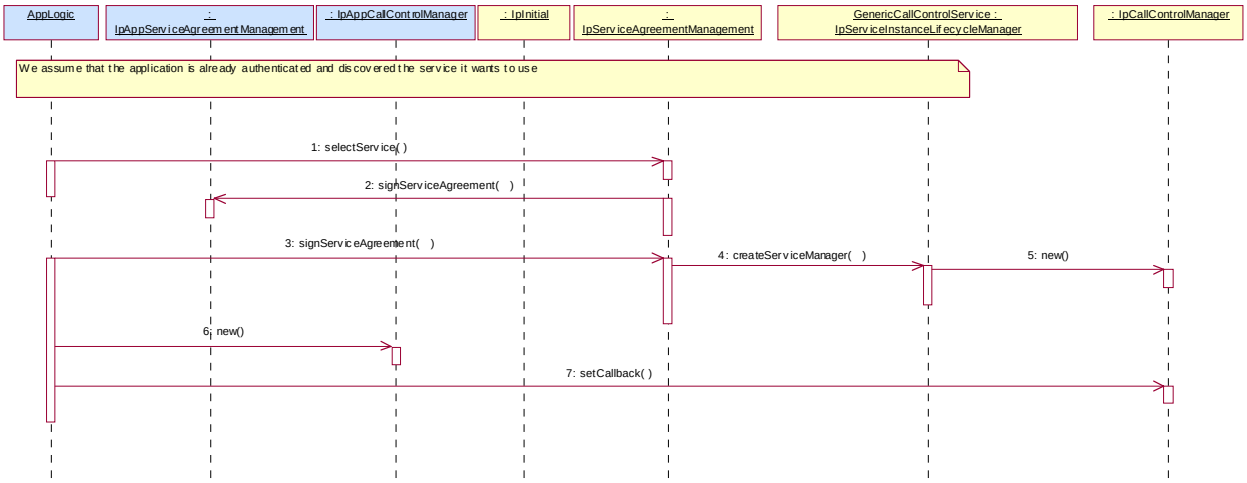
- in serviceInstanceLifecycleManagerRef.

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

9.1.3 Service Instance Lifecycle Manager Sequence Diagrams

9.1.3.1 Sign Service Agreement

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



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 client application signs the service agreement.

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

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

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

Following the creation of the service manager outlined above, a unique instance of the service particular to the application client results. This service instance is assigned a serviceInstanceID by the Framework, which is provided to the Service Instance Lifecycle manager during the createServiceManager operation. If it is necessary that Framework Integrity Management functionality and operations are to be supported between the Framework and the service instance identified by the defined serviceInstanceID, it is then necessary for the new service instance to establish an access session with the Framework. This provides the Framework with the ability to manage and monitor the operation of the service instance that relates to a particular application client. The steps required to establish a Framework access session are outlined in clause 6 of the present document.

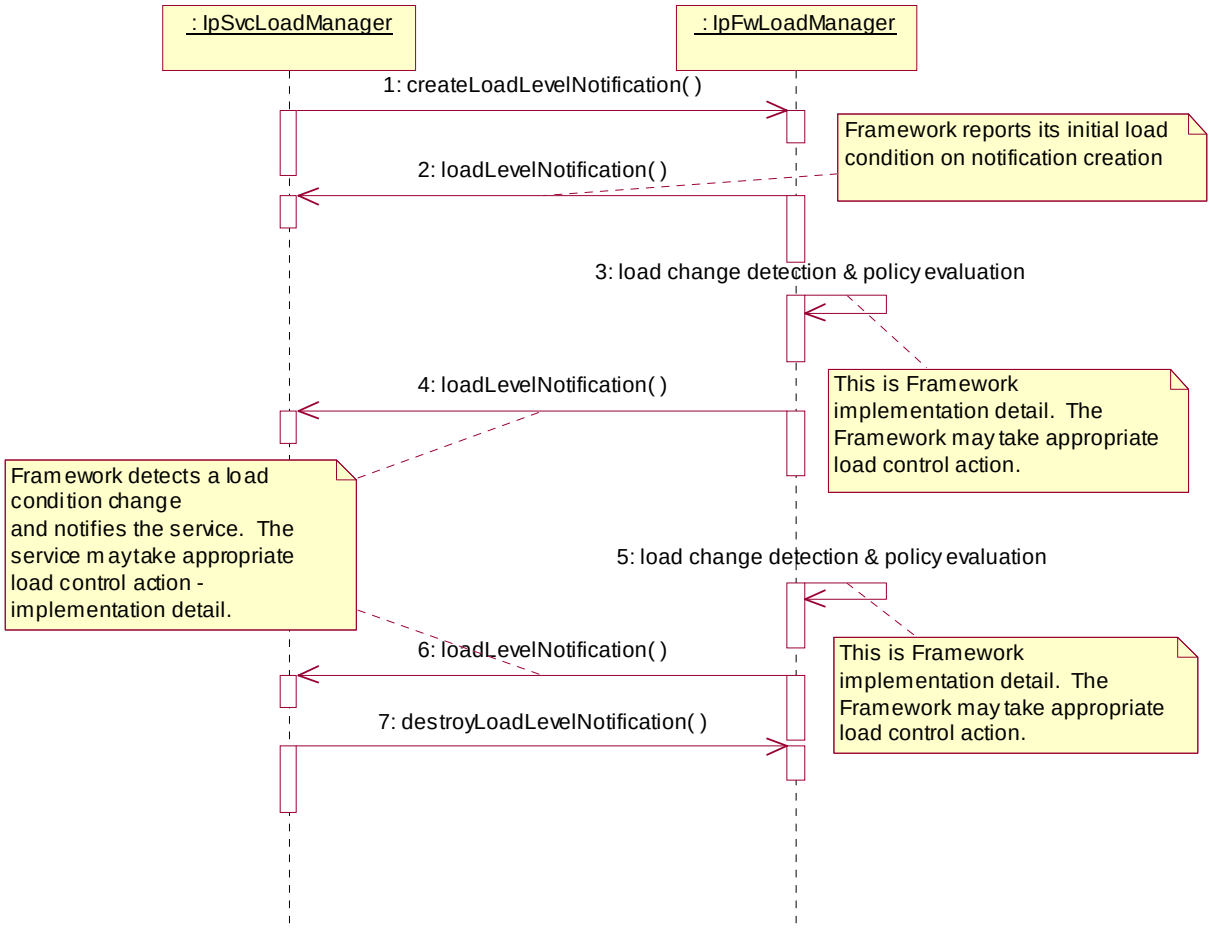
6: The application creates a new IpAppCallControlManager interface to be used for callbacks.

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

9.1.4 Integrity Management Sequence Diagrams

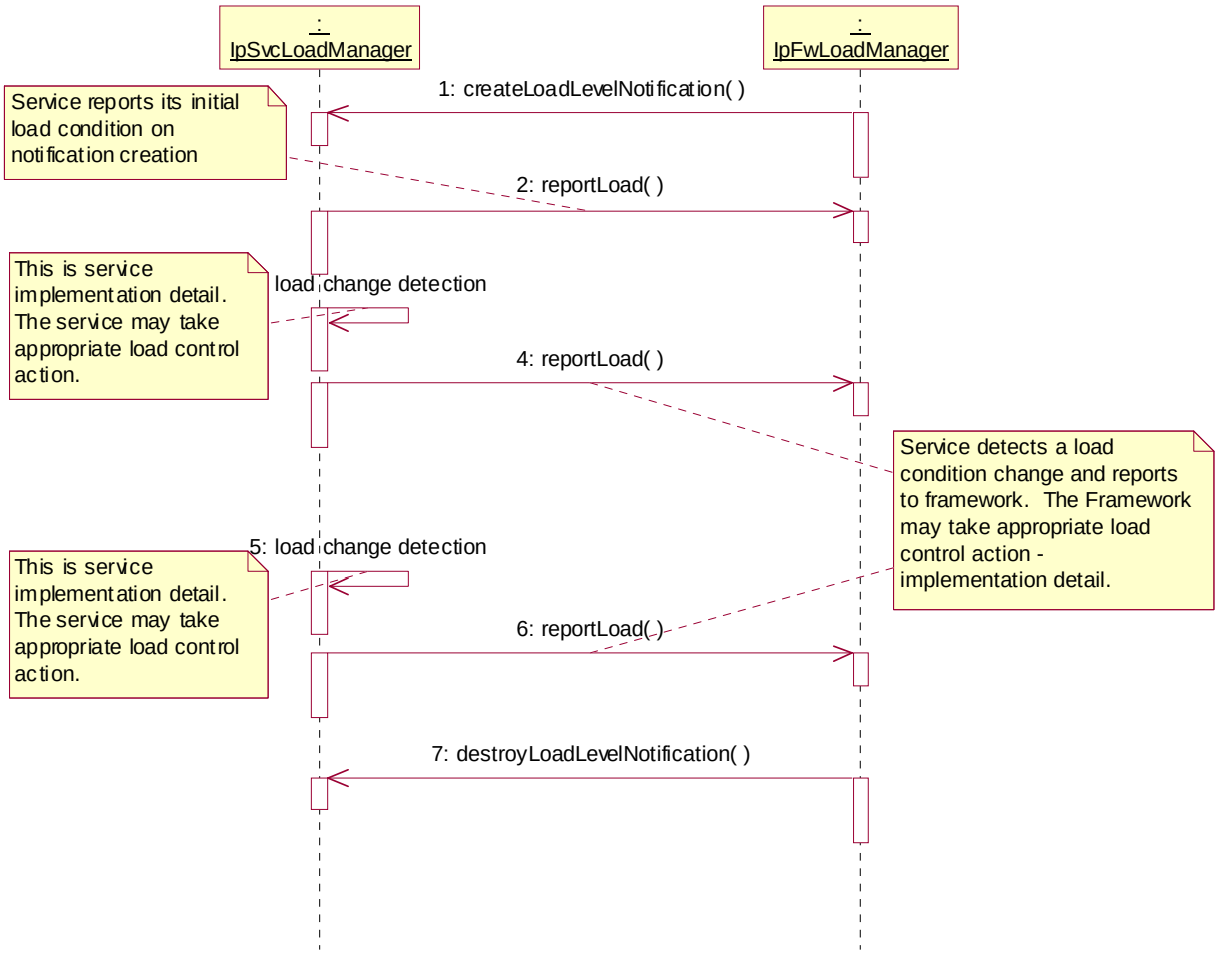
9.1.4.1 Load Management: Service callback registration and load control

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

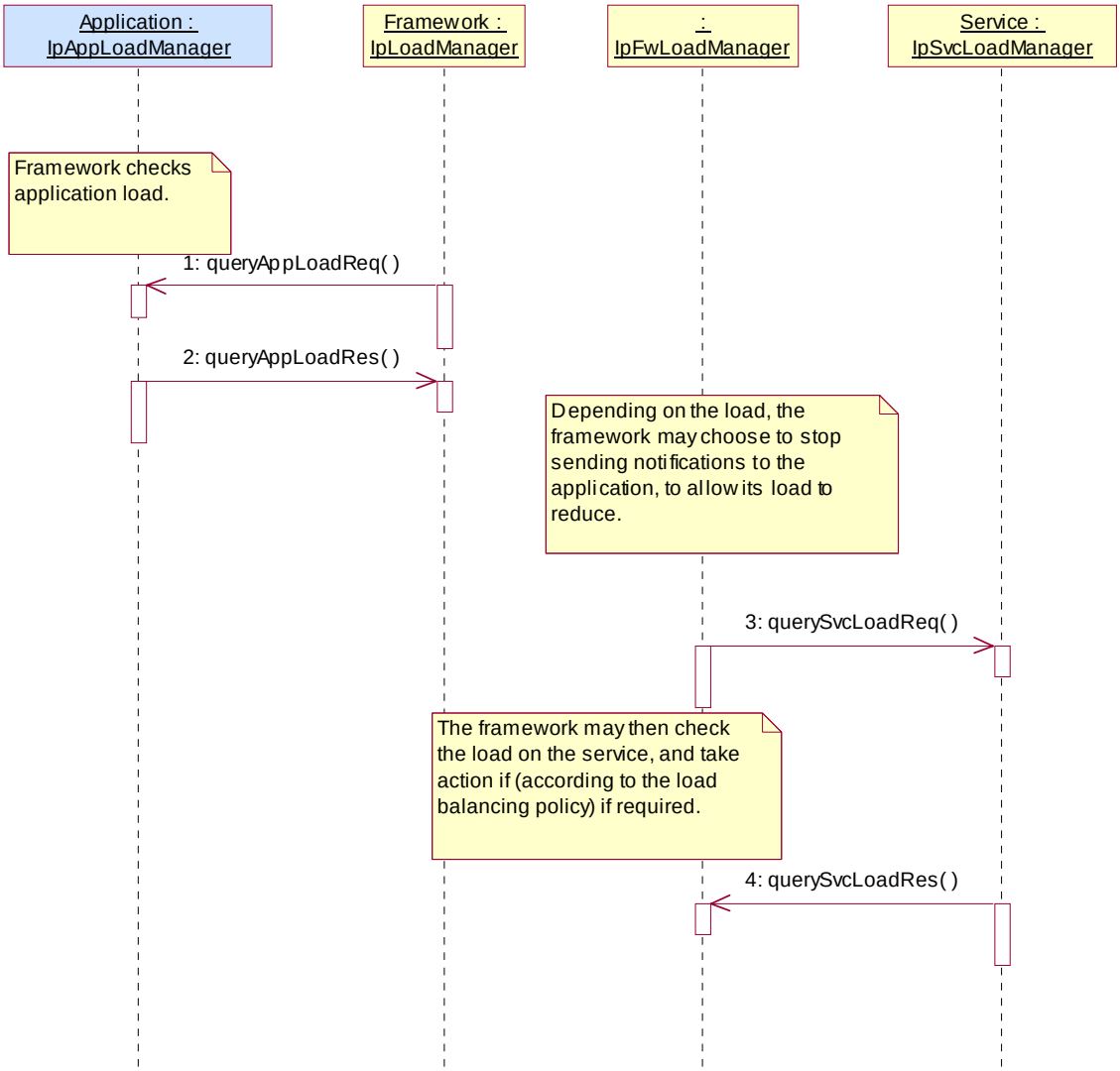


9.1.4.2 Load Management: Framework callback registration and service load control

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

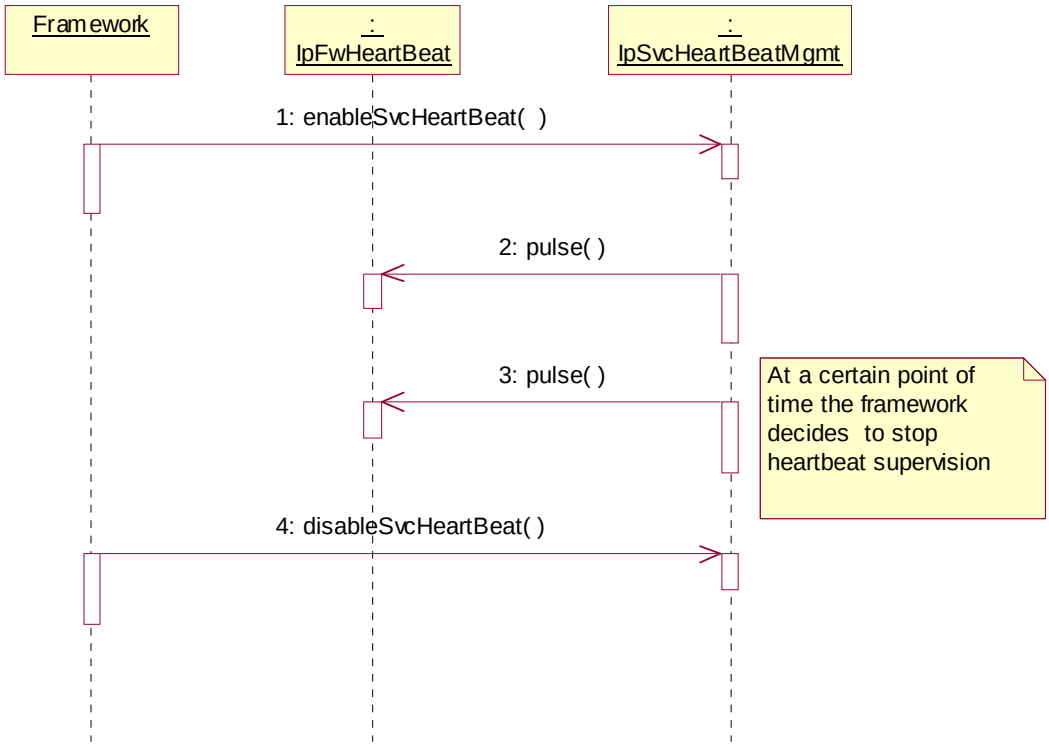


9.1.4.3 Load Management: Client and Service Load Balancing

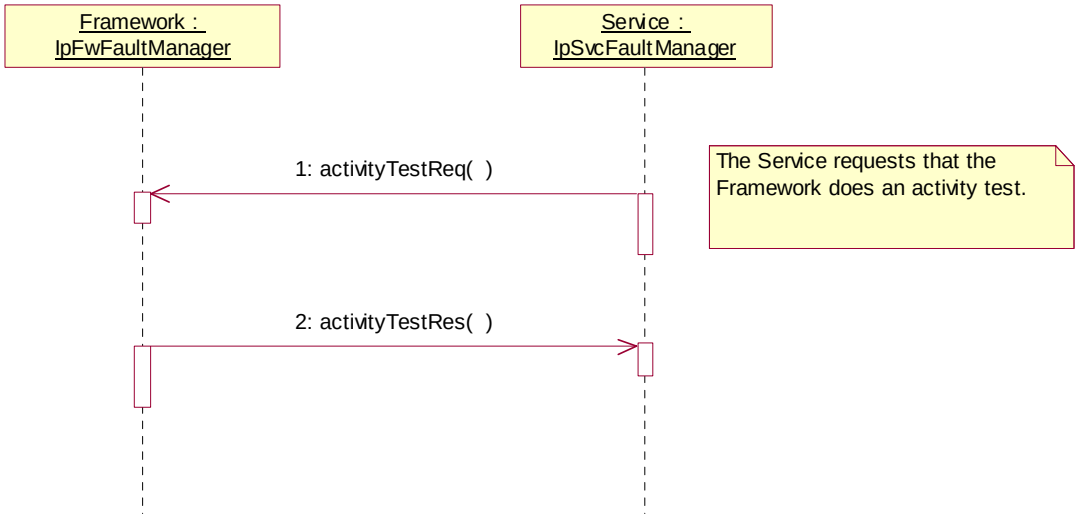


9.1.4.4 Heartbeat Management: Start/perform/end heartbeat supervision of the service

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



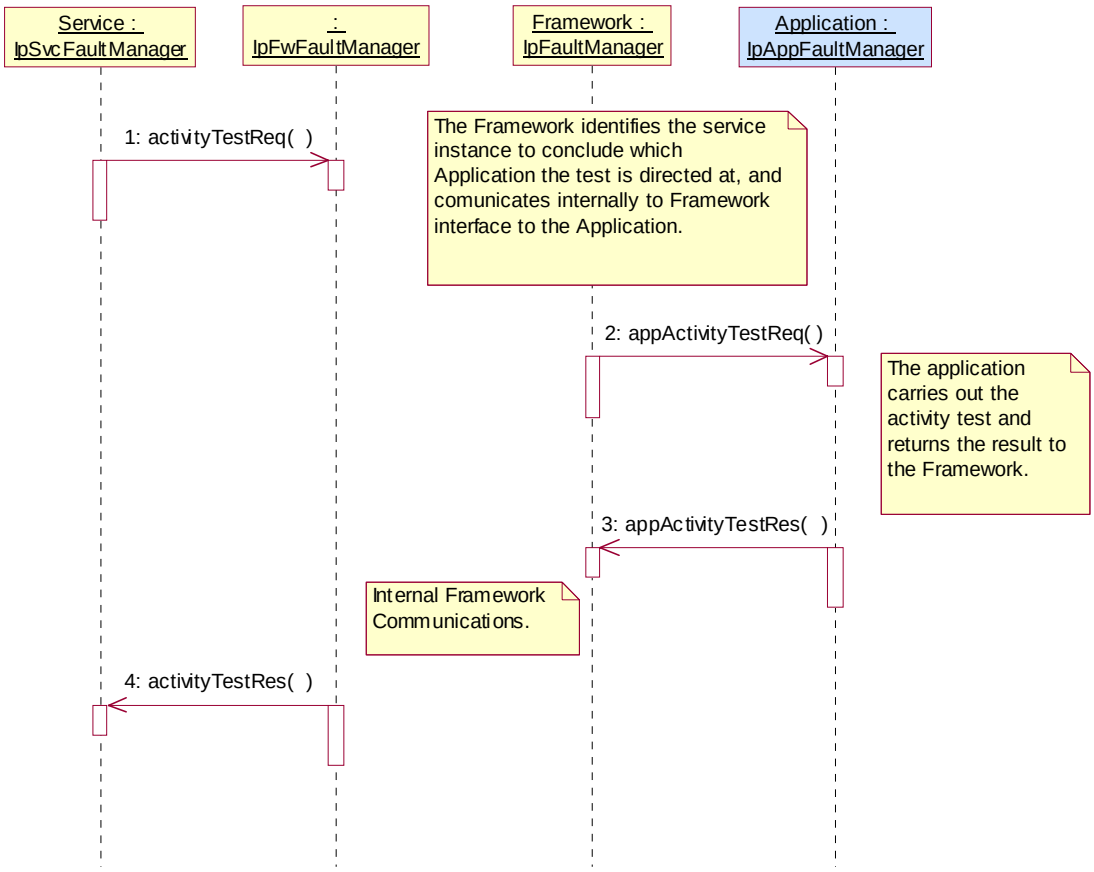
9.1.4.5 Fault Management: Service requests Framework activity test



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

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

9.1.4.6 Fault Management: Service requests Application activity test



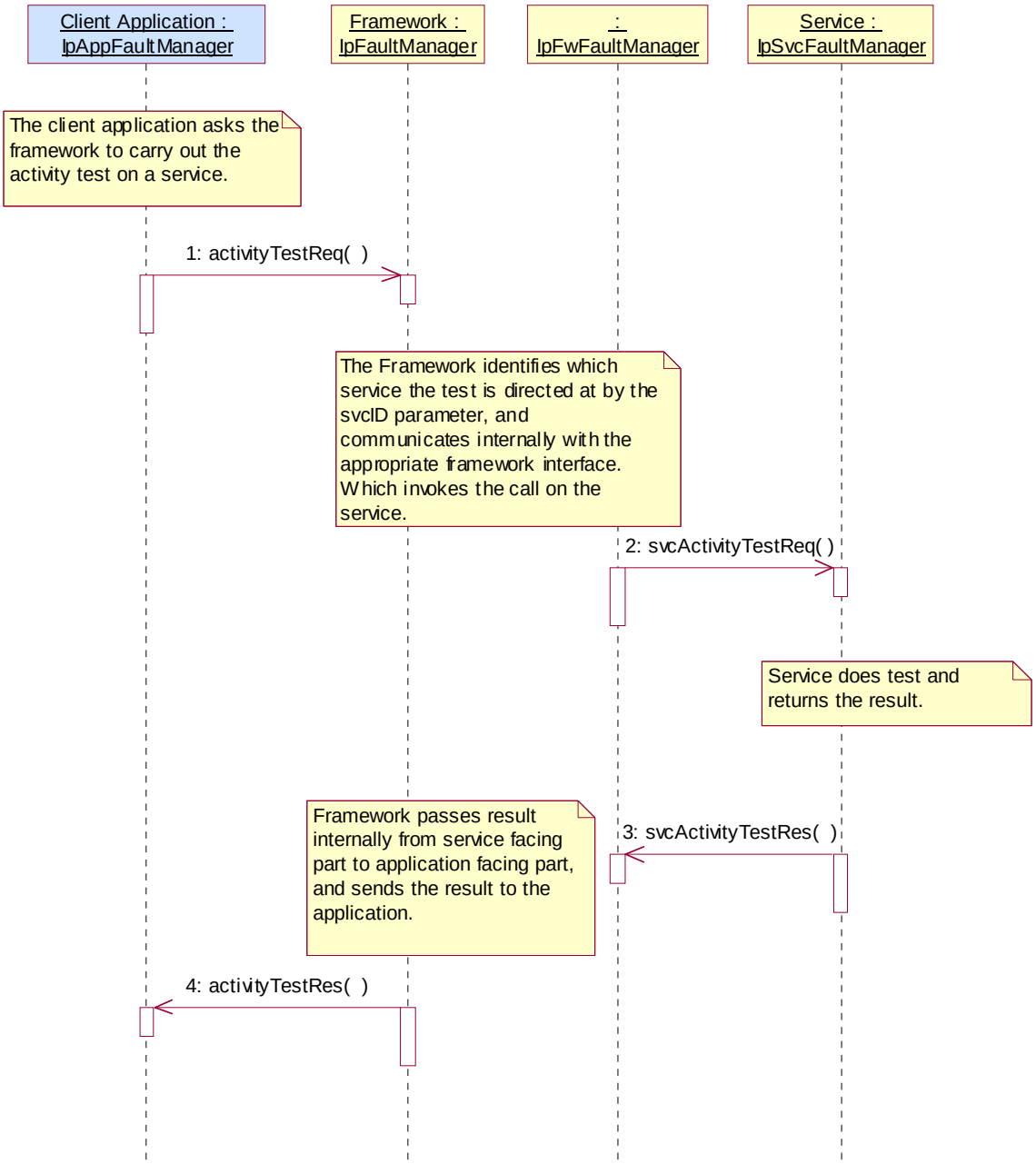
1: The service asks the framework to invoke an activity test on a client application, the application is identified by the appId parameter.

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

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

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

9.1.4.7 Fault Management: Application requests Service activity test



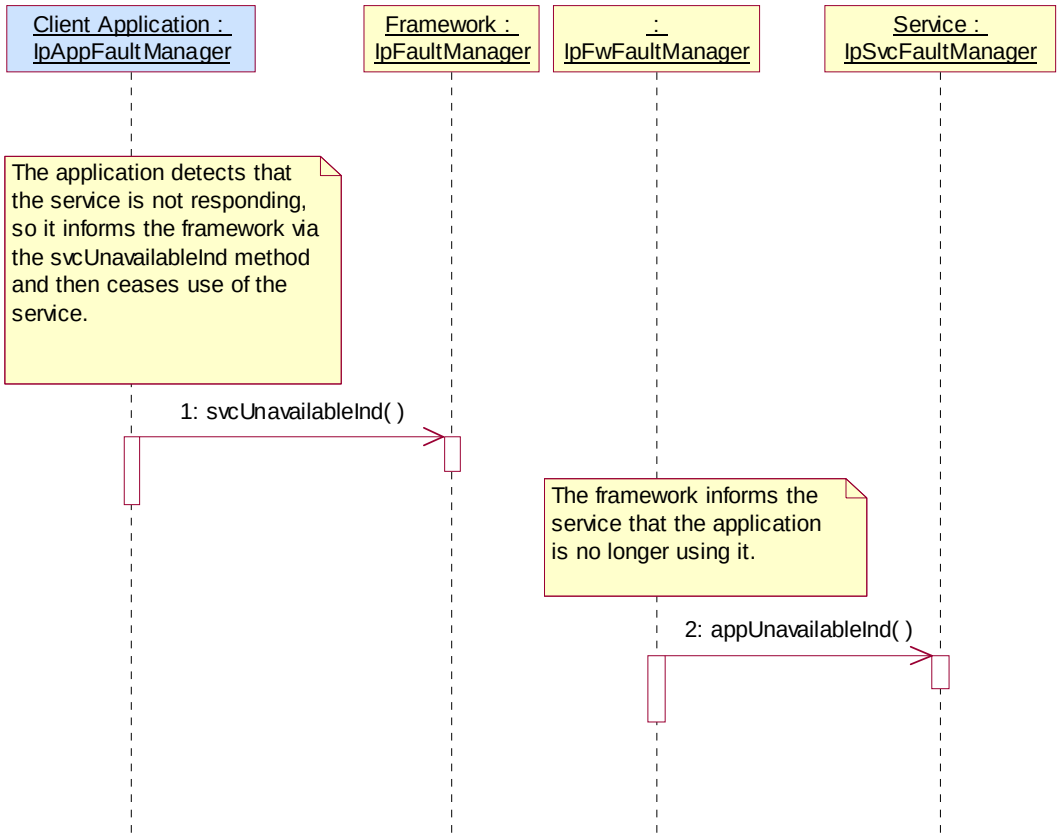
1: The client application asks the framework to invoke an activity test on a service, the service is identified by the svcId parameter.

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

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

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

9.1.4.8 Fault Management: Application detects service is unavailable



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

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

9.1.5 Event Notification Sequence Diagrams

No Sequence Diagrams exist for Event Notification.

9.2 Class Diagrams

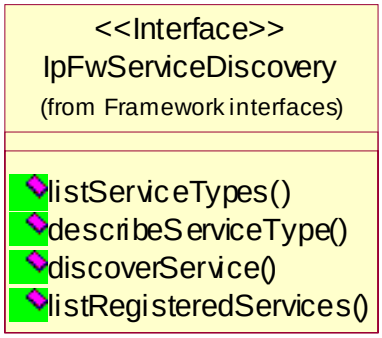


Figure 22: Service Discovery Package Overview

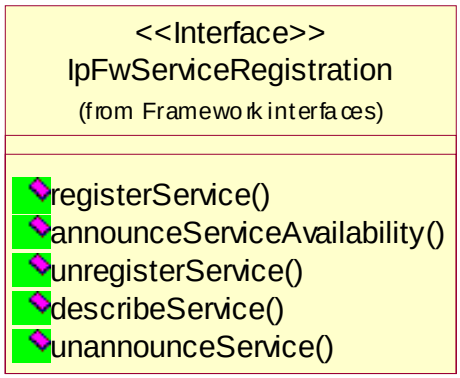


Figure 23: Service Registration Package Overview

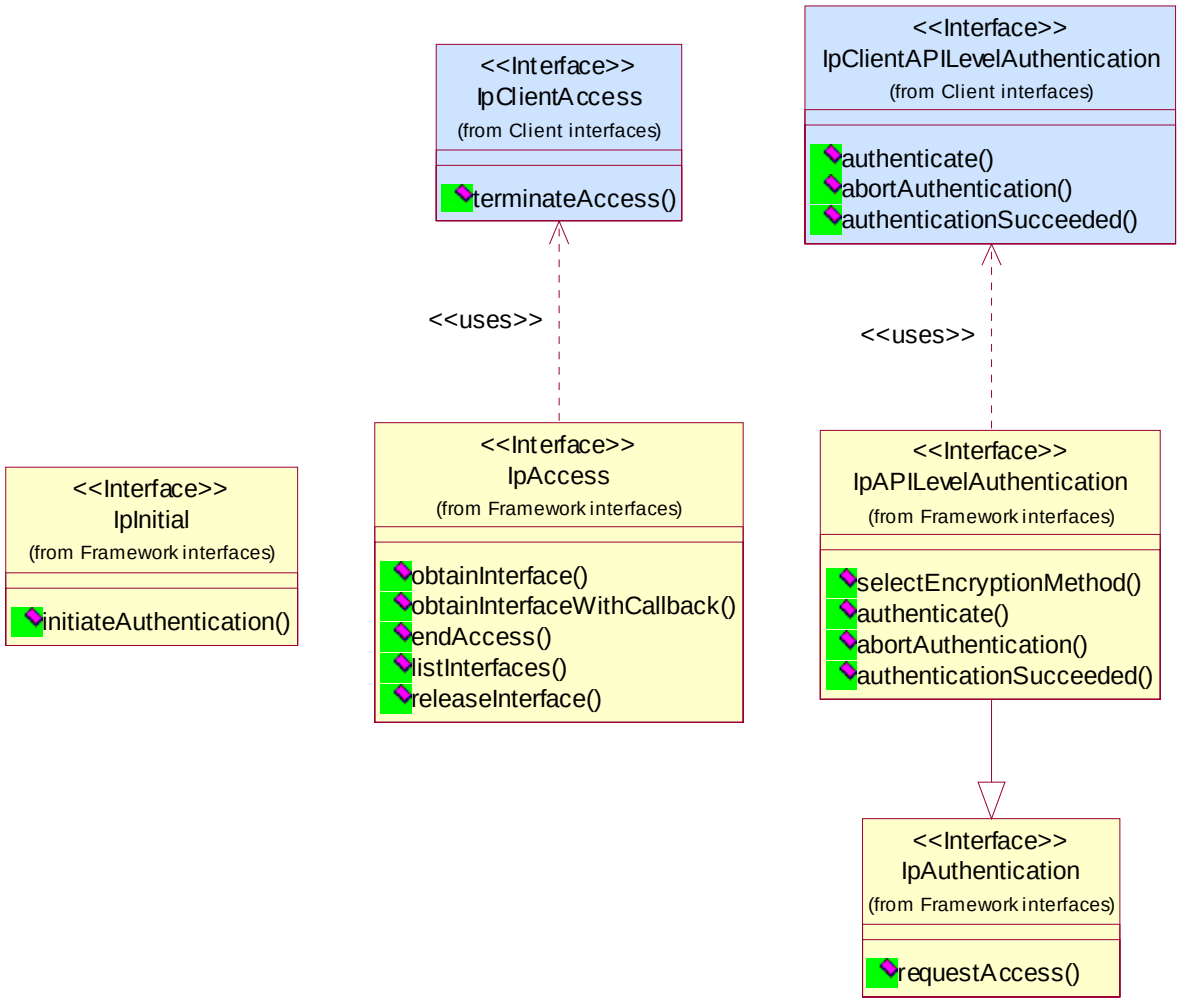


Figure 24: Trust and Security Management Package Overview

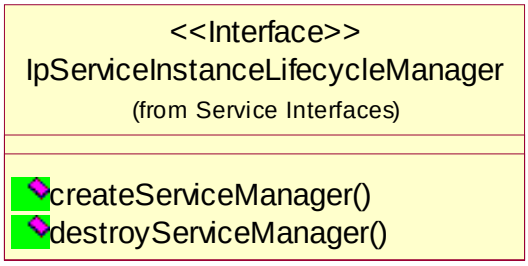


Figure 25: Service Instance Lifecycle Manager Package Overview

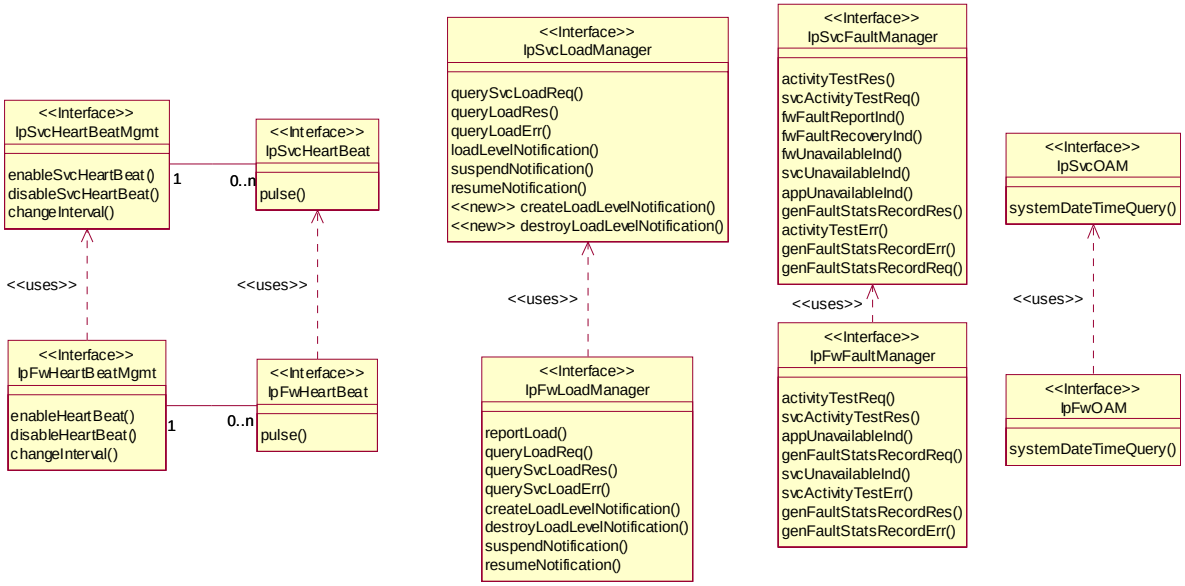


Figure 26: Integrity Management Package Overview

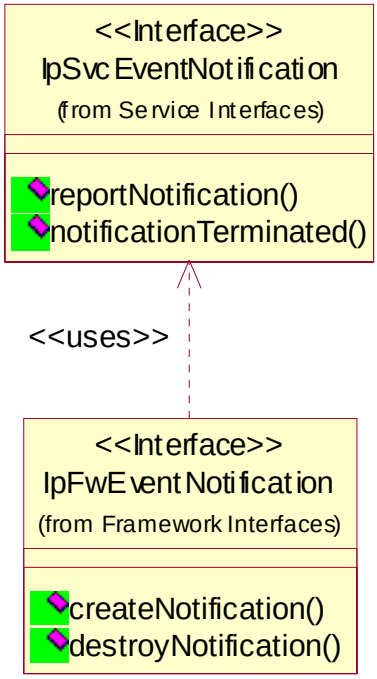


Figure 27: Event Notification Package Overview

9.3 Interface Classes

9.3.1 Service Registration Interface Classes

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

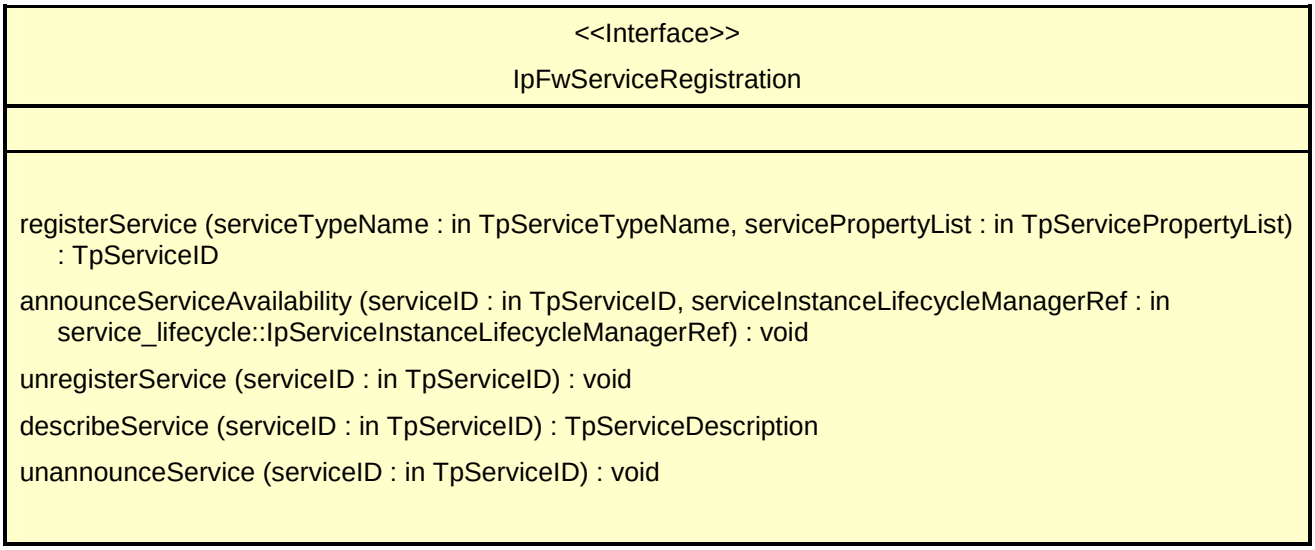
In order to register a new service in the framework, the service supplier must select a service type and the "property values" for the service. The service discovery functionality described in the previous clause enables the service supplier to obtain a list of all the service types supported by the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

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

9.3.1.1 Interface Class IpFwServiceRegistration

Inherits from: IpInterface.

The Service Registration interface provides the methods used for the registration of network SCFs at the framework. This interface and at least the methods registerService(), announceServiceAvailability(), unregisterService() and unannounceService() shall be implemented by a Framework.



Method

registerService()

The registerService() operation is the means by which a service is registered in the Framework, for subsequent discovery by the enterprise applications. Registration can only succeed when the Service type of the service is known to the Framework (ServiceType is "available"). A service-ID is returned to the service supplier when a service is registered in the Framework. When the service is not registered because the ServiceType is "unavailable", a P_SERVICE_TYPE_UNAVAILABLE is raised. The service-ID is the handle with which the service supplier can identify the registered service when needed (e.g. for withdrawing it). The service-ID is only meaningful in the context of the Framework that generated it.

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

Parameters

`serviceTypeName : in TpServiceTypeName`

The "serviceTypeName" parameter identifies the service type. If the string representation of the "type" does not obey the rules for identifiers, then an P_ILLEGAL_SERVICE_TYPE exception is raised. If the "type" is correct syntactically but the Framework is able to unambiguously determine that it is not a recognised service type, then a P_UNKNOWN_SERVICE_TYPE exception is raised.

servicePropertyList : in TpServicePropertyList

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

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

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

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

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

Returns

TpServiceID

Raises

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

Method

announceServiceAvailability()

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

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

Parameters

serviceID : in TpServiceID

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

serviceInstanceLifecycleManagerRef : in service_lifecycle::IpServiceInstanceLifecycleManagerRef

The interface reference at which the service instance lifecycle manager of the previously registered service is available.

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID, P_INVALID_INTERFACE_TYPE

Method

unregisterService()

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

Parameters

serviceID : in TpServiceID

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

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID

Method

describeService()

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

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

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

Parameters

serviceID : in TpServiceID

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

Returns

TpServiceDescription

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID

Method

unannounceService()

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

Parameters

serviceID : in TpServiceID

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

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_ID, P_UNKNOWN_SERVICE_ID

9.3.2 Service Instance Lifecycle Manager Interface Classes

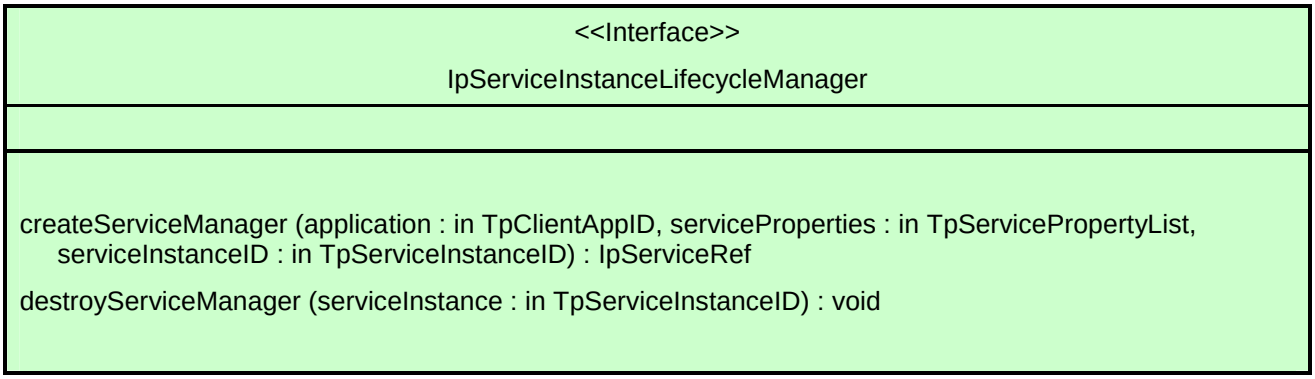
The IpServiceInstanceLifecycleManager interface allows the framework to get access to a service manager interface of a service. It is used during the signServiceAgreement, in order to return a service manager interface reference to the application. Each service has a service manager interface that is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

9.3.2.1 Interface Class IpServiceInstanceLifecycleManager

Inherits from: IpInterface.

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

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



Method

createServiceManager()

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

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

Parameters

application : in TpClientAppID

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

serviceProperties : in TpServicePropertyList

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

serviceInstanceID : in TpServiceInstanceID

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

Returns

IpServiceRef

Raises

TpCommonExceptions, P_INVALID_PROPERTY

Method

destroyServiceManager()

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

Parameters

serviceInstance : in TpServiceInstanceID

Identifies the Service Instance to be destroyed.

Raises

TpCommonExceptions

9.3.3 Service Discovery Interface Classes

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

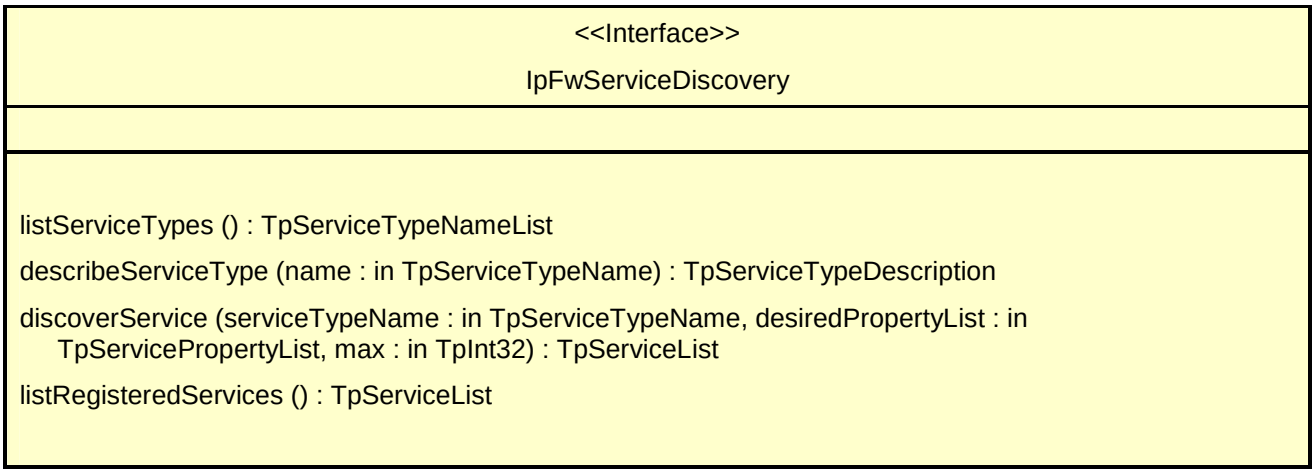
Before a service can be registered in the framework, the service supplier must know what "types" of services the Framework supports and what service "properties" are applicable to each service type. The "listServiceType()" method returns a list of all "service types" that are currently supported by the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

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

9.3.3.1 Interface Class IpFwServiceDiscovery

Inherits from: IpInterface.

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



Method
listServiceTypes()

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

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

Parameters
No Parameters were identified for this method.

Returns

TpServiceTypeNameList

Raises

TpCommonExceptions

Method
describeServiceType()

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

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

Parameters

name : in TpServiceTypeName

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

Returns

TpServiceTypeDescription

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE

Method

discoverService()

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

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

Parameters

serviceTypeName : in TpServiceTypeName

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

desiredPropertyList : in TpServicePropertyList

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

max : in TpInt32

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

Returns

TpServiceList

Raises

TpCommonExceptions, P_ILLEGAL_SERVICE_TYPE, P_UNKNOWN_SERVICE_TYPE, P_INVALID_PROPERTY

Method

listRegisteredServices()

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

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

Parameters

No Parameters were identified for this method.

Returns

TpServiceList

Raises

`TpCommonExceptions`

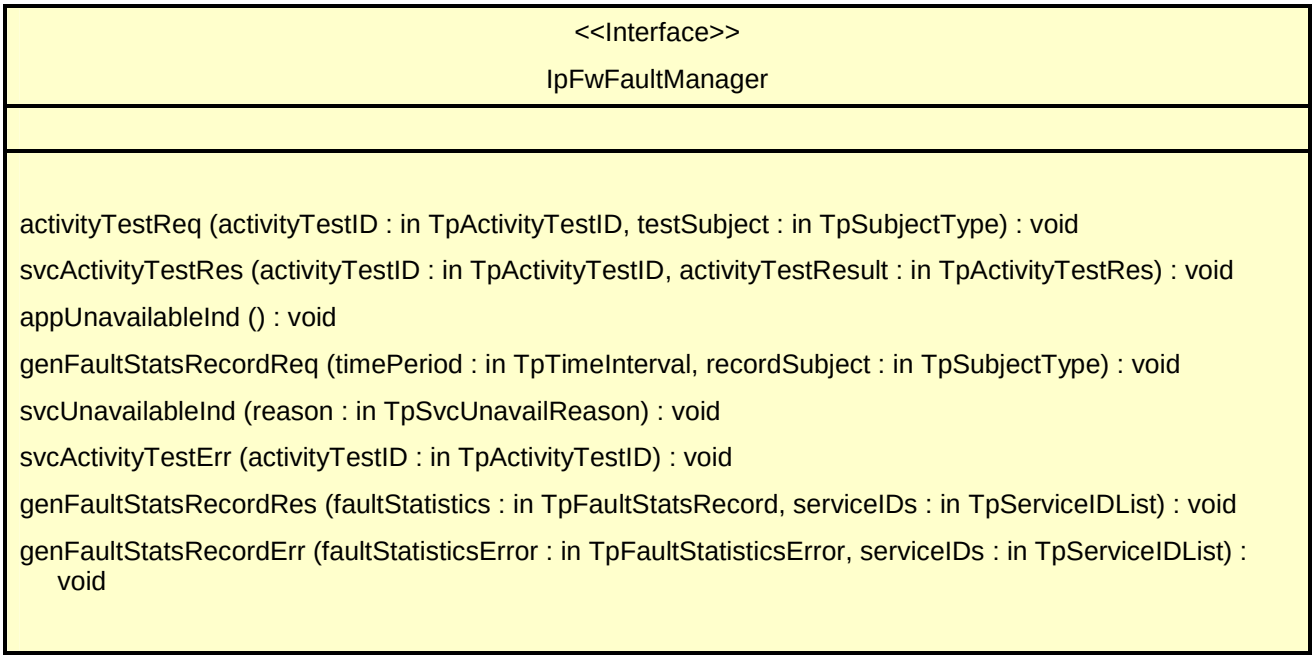
9.3.4 Integrity Management Interface Classes

9.3.4.1 Interface Class IpFwFaultManager

Inherits from: IpInterface.

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

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



Method

`activityTestReq()`

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

Parameters

`activityTestID : in TpActivityTestID`

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

`testSubject : in TpSubjectType`

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

Raises

TpCommonExceptions

Method

svcActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

appUnavailableInd()

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

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

Method

genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

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

recordSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

svcUnavailableInd()

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

Parameters

reason : in TpSvcUnavailReason

Identifies the reason for the service instance's unavailability.

Raises

TpCommonExceptions

Method

svcActivityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

genFaultStatsRecordRes()

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

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions

Method

genFaultStatsRecordErr()

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

Parameters

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

serviceIDs : in TpServiceIDList

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

Raises

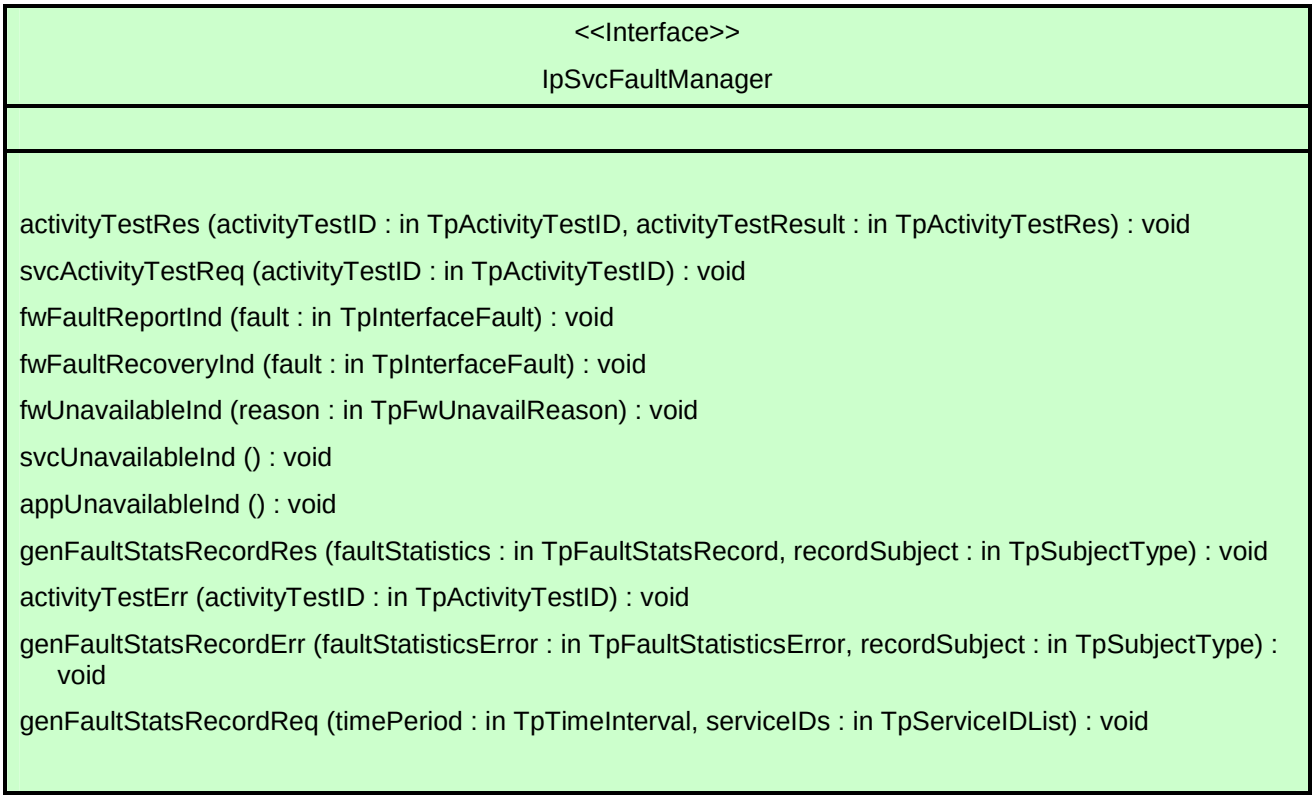
TpCommonExceptions

9.3.4.2 Interface Class IpSvcFaultManager

Inherits from: IpInterface.

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

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



Method

activityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

svcActivityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions

Method

fwFaultReportInd()

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

Parameters

fault : in TpInterfaceFault

Specifies the fault that has been detected by the framework.

Raises

TpCommonExceptions

Method

fwFaultRecoveryInd()

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

Parameters

fault : in TpInterfaceFault

Specifies the fault from which the framework has recovered.

Raises

TpCommonExceptions

Method

fwUnavailableInd()

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

Parameters

reason : in TpFwUnavailReason

Identifies the reason why the framework is no longer available.

Raises

TpCommonExceptions

Method

svcUnavailableInd()

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

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

Method

appUnavailableInd()

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

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

Method

genFaultStatsRecordRes()

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

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

recordSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

activityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

genFaultStatsRecordErr()

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

Parameters

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

recordSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

genFaultStatsRecordReq()

This method is used by the framework to solicit fault statistics from the service, for example when the framework was asked for these statistics by the client application using the genFaultStatsRecordReq operation on the IpFaultManager interface. On receipt of this request the service must produce a fault statistics record, for either the framework or for the client's instances of the specified services during the specified time interval, which is returned to the framework using the genFaultStatsRecordRes operation on the IpFwFaultManager interface. If the framework does not have access to a service instance with the specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

timePeriod : in TpTimeInterval

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

serviceIDs : in TpServiceIDList

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

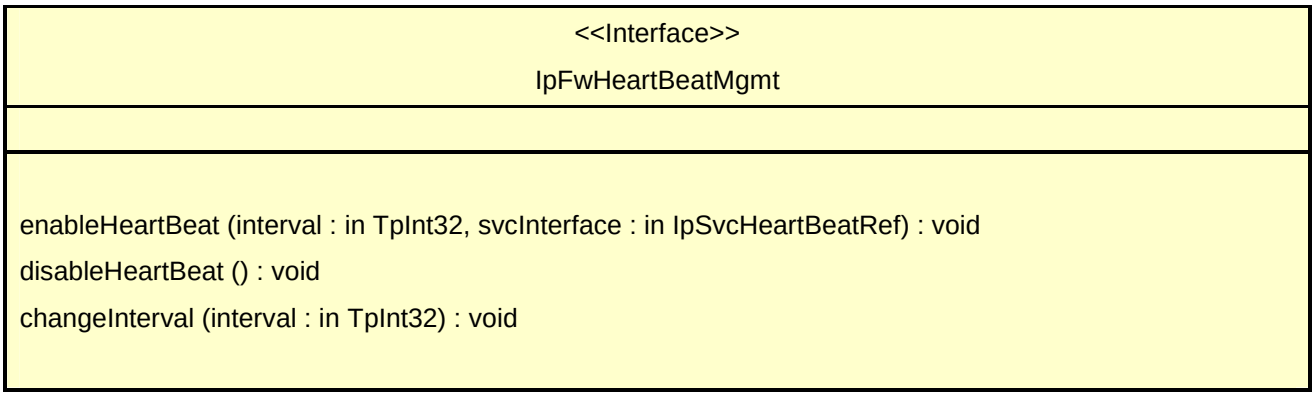
Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

9.3.4.3 Interface Class IpFwHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the framework by a service instance. If the IpFwHeartBeatMgmt interface is implemented by a Framework, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.



Method
enableHeartBeat()

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

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

svcInterface : in IpSvcHeartBeatRef

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

Raises

TpCommonExceptions, P_INVALID_INTERFACE_TYPE

Method
disableHeartBeat()

Instructs the framework to cease the sending of its heartbeat.

Parameters

No Parameters were identified for this method.

Raises

TpCommonExceptions

Method
changeInterval()

Allows the administrative change of the heartbeat interval.

Parameters

interval : in TpInt32

The time interval in milliseconds between the heartbeats.

Raises

TpCommonExceptions

9.3.4.4 Interface Class IpFwHeartBeat

Inherits from: IpInterface.

The service side framework heartbeat interface is used by the service instance to send the framework its heartbeat. If a Framework is capable of invoking IpSvcHeartBeatMgmt.enableHeartBeat(), it shall implement IpFwHeartBeat and the pulse() method.

<<Interface>> IpFwHeartBeat
pulse () : void

Method
`pulse()`

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

Parameters
No Parameters were identified for this method.

Raises

`TpCommonExceptions`

9.3.4.5 Interface Class IpSvcHeartBeatMgmt

Inherits from: IpInterface.

This interface allows the initialisation of a heartbeat supervision of the service instance by the framework. If the IpSvcHeartBeatMgmt interface is implemented by a Service, as a minimum enableHeartBeat() and disableHeartBeat() shall be implemented.

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

Method
`enableSvcHeartBeat()`

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

Parameters

`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

`fwInterface : in IpFwHeartBeatRef`

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

Raises

`TpCommonExceptions, P_INVALID_INTERFACE_TYPE`

Method

`disableSvcHeartBeat()`

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

Parameters

No Parameters were identified for this method.

Raises

`TpCommonExceptions`

Method

`changeInterval()`

Allows the administrative change of the heartbeat interval.

Parameters

`interval : in TpInt32`

The time interval in milliseconds between the heartbeats.

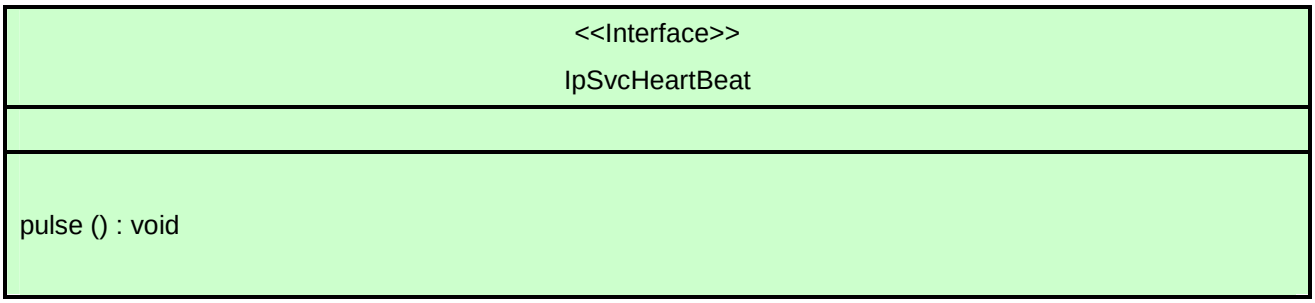
Raises

`TpCommonExceptions`

9.3.4.6 Interface Class IpSvcHeartBeat

Inherits from: IpInterface.

The service heartbeat interface is used by the framework to send the service instance its heartbeat. If a Service is capable of invoking IpFwHeartBeatMgmt.enableHeartBeat(), it shall implement IpSvcHeartBeat and the pulse() method.



Method
`pulse()`

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

Parameters
No Parameters were identified for this method.

Raises

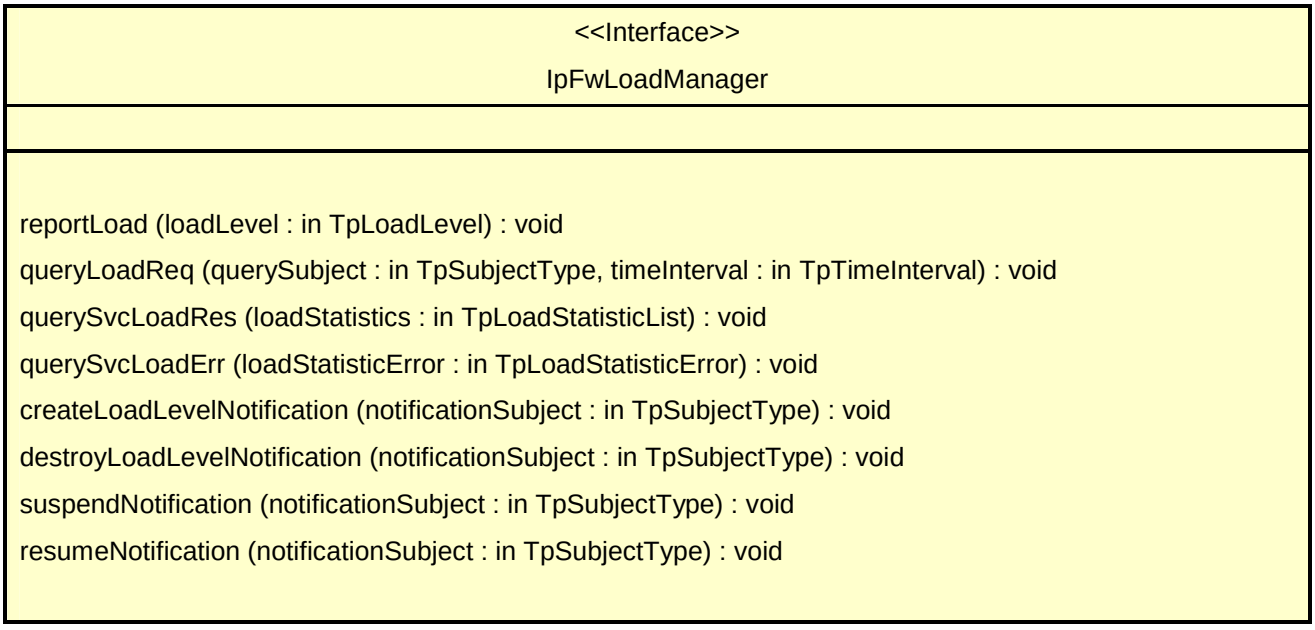
`TpCommonExceptions`

9.3.4.7 Interface Class IpFwLoadManager

Inherits from: IpInterface.

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

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



Method
reportLoad()

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

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

Parameters

loadLevel : in TpLoadLevel

Specifies the service instance's load level.

Raises

TpCommonExceptions

Method
queryLoadReq()

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

Parameters

querySubject : in TpSubjectType

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

timeInterval : in TpTimeInterval

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

Raises

TpCommonExceptions

Method
querySvcLoadRes()

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

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the service-supplied load statistics.

Raises

TpCommonExceptions

Method
querySvcLoadErr()

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

Parameters

loadStatisticError : in TpLoadStatisticError

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

Raises

TpCommonExceptions

Method

createLoadLevelNotification()

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

Parameters

notificationSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

destroyLoadLevelNotification()

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

Parameters

notificationSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

suspendNotification()

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

Parameters

notificationSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

resumeNotification()

The service instance uses this method to request the framework to resume sending it notifications associated with the framework or with the application that uses the service instance; e.g. after a period of suspension during which the service instance handled a temporary overload condition. Upon receipt of this method the framework shall inform the service instance of the current framework or application load using the loadLevelNotification method on the corresponding IpSvcLoadManager.

Parameters

notificationSubject : in TpSubjectType

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

Raises

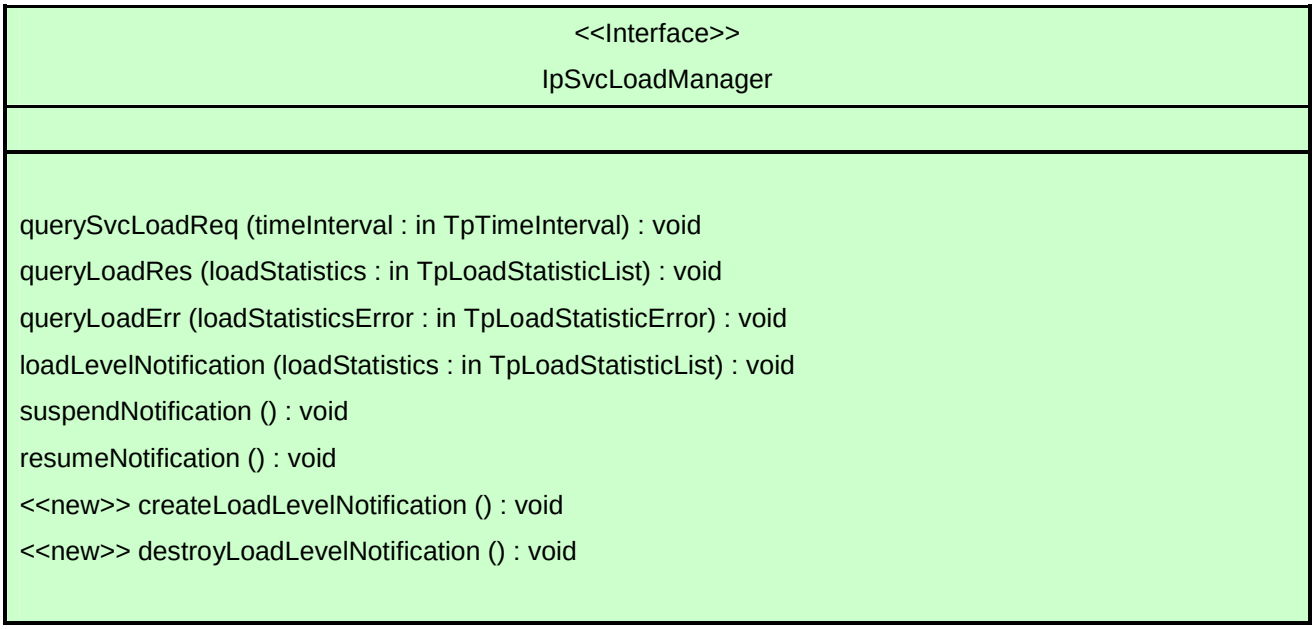
TpCommonExceptions

9.3.4.8 Interface Class IpSvcLoadManager

Inherits from: IpInterface.

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

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



Method

querySvcLoadReq()

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

Parameters

timeInterval : in TpTimeInterval

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

Raises

TpCommonExceptions

Method

queryLoadRes()

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

Parameters

loadStatistics : in TpLoadStatisticList

Specifies the framework-supplied load statistics.

Raises

TpCommonExceptions

Method

queryLoadErr()

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

Parameters

loadStatisticsError : in TpLoadStatisticError

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

Raises

TpCommonExceptions

Method

loadLevelNotification()

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

Parameters

loadStatistics : in TpLoadStatisticList

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

Raises

TpCommonExceptions

Method
suspendNotification()

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

Parameters
No Parameters were identified for this method.

Raises

TpCommonExceptions

Method
resumeNotification()

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

Parameters
No Parameters were identified for this method.

Raises

TpCommonExceptions

Method
<<new>> createLoadLevelNotification()

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

Parameters
No Parameters were identified for this method.

Raises

TpCommonExceptions

Method
<<new>> destroyLoadLevelNotification()

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

Parameters
No Parameters were identified for this method.

Raises

TpCommonExceptions

9.3.4.9 Interface Class IpFwOAM

Inherits from: IpInterface.

The OAM interface is used to query the system date and time. The service and the framework can synchronise the date and time to a certain extent. Accurate time synchronisation is outside the scope of this API. This interface and the systemDateTimeQuery() method are optional.

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

Method

`systemDateTimeQuery()`

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

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

Parameters

`clientDateAndTime : in TpDateAndTime`

This is the date and time of the client (service). The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

`TpDateAndTime`

Raises

`TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT`

9.3.4.10 Interface Class IpSvcOAM

Inherits from: IpInterface.

This interface and the systemDateTimeQuery() method are optional.

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

Method

systemDateTimeQuery()

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

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

Parameters

systemDateAndTime : in TpDateAndTime

This is the system date and time of the framework. The error code P_INVALID_DATE_TIME_FORMAT is returned if the format of the parameter is invalid.

Returns

TpDateAndTime

Raises

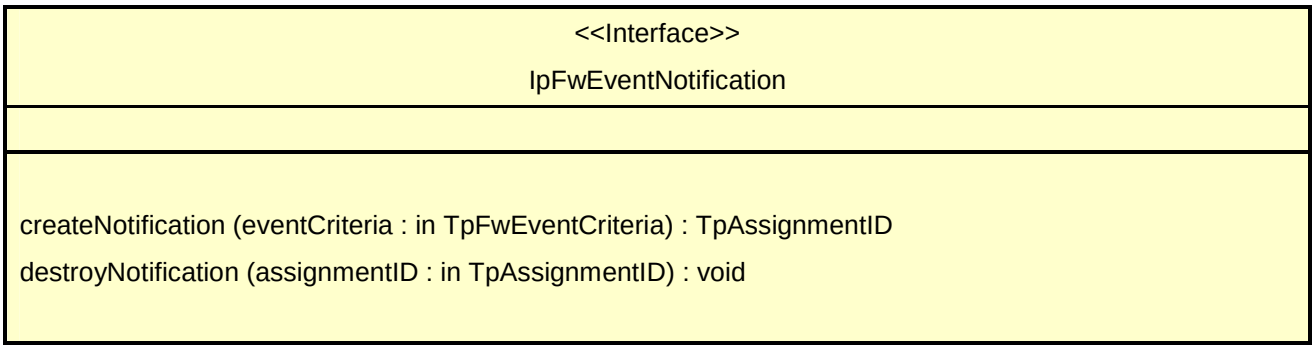
TpCommonExceptions, P_INVALID_TIME_AND_DATE_FORMAT

9.3.5 Event Notification Interface Classes

9.3.5.1 Interface Class IpFwEventNotification

Inherits from: IpInterface.

The event notification mechanism is used to notify the service of generic events that have occurred. If Event Notifications are supported by a Framework, this interface and the createNotification() and destroyNotification() methods shall be implemented.



Method

createNotification()

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

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

Parameters

eventCriteria : in TpFwEventCriteria

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

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_INVALID_EVENT_TYPE, P_INVALID_CRITERIA

Method

destroyNotification()

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

Parameters

assignmentID : in TpAssignmentID

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

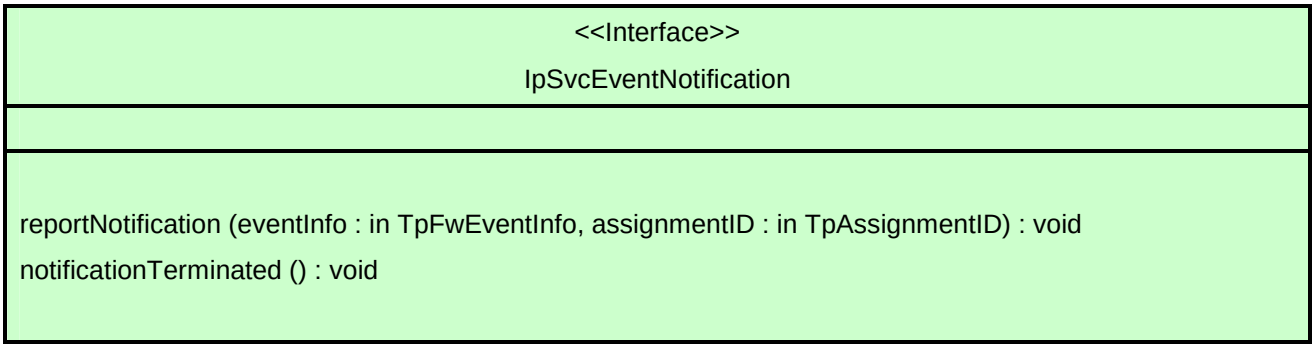
Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

9.3.5.2 Interface Class IpSvcEventNotification

Inherits from: IpInterface.

This interface is used by the framework to inform the service of a generic event. The Event Notification Framework will invoke methods on the Event Notification Service Interface that is specified when the Event Notification interface is obtained. If Event Notifications are supported by a Service, this interface and the reportNotification() and notificationTerminated() methods shall be implemented.



Method

reportNotification()

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

Parameters

eventInfo : in TpFwEventInfo

Specifies specific data associated with this event.

assignmentID : in TpAssignmentID

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

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

Method
`notificationTerminated()`

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

Parameters
No Parameters were identified for this method.

Raises

`TpCommonExceptions`

9.4 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 client are shown. Methods not shown for a specific state are not relevant for that state and will return an exception. Apart from the methods that can be invoked by the client also events internal to the gateway or related to network events are shown together with the resulting event or action performed by the gateway. These internal events are shown between quotation marks.

9.4.1 Service Registration State Transition Diagrams

9.4.1.1 State Transition Diagrams for IpFwServiceRegistration

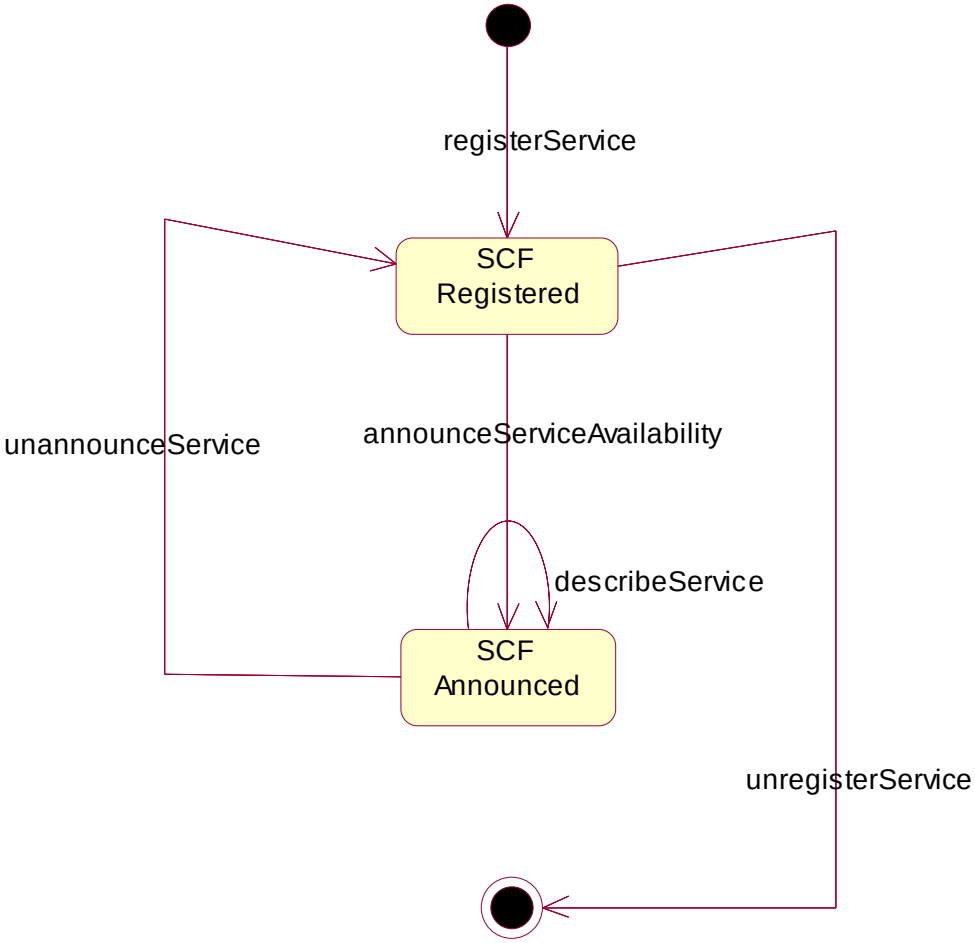


Figure 28: State Transition Diagram for IpFwServiceRegistration

9.4.1.1.1 SCF Registered State

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

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

9.4.1.1.2 SCF Announced State

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

9.4.2 Service Instance Lifecycle Manager State Transition Diagrams

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

9.4.3 Service Discovery State Transition Diagrams

There are no State Transition Diagrams defined for Service Discovery.

9.4.4 Integrity Management State Transition Diagrams

9.4.4.1 State Transition Diagrams for IpFwLoadManager

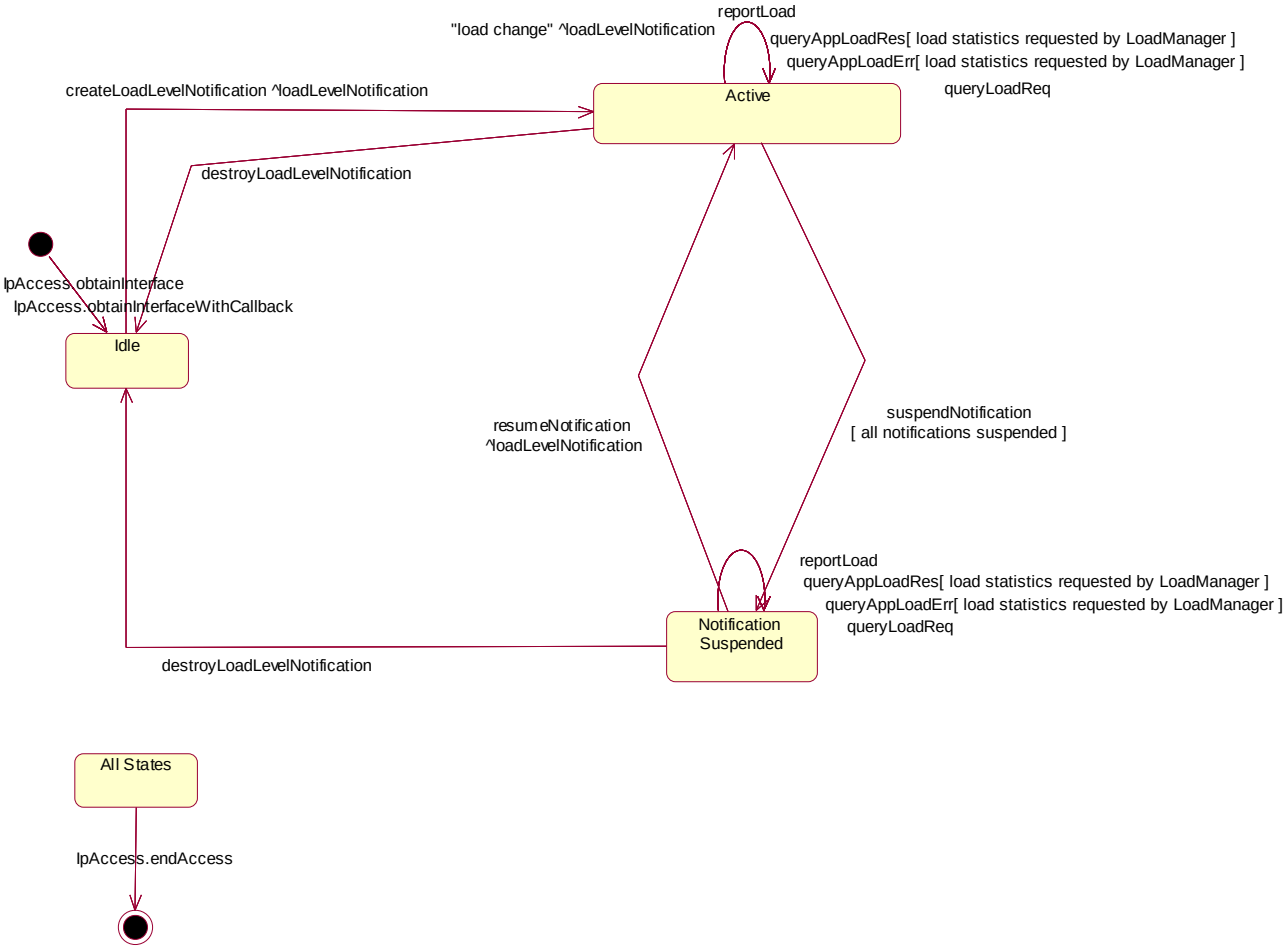


Figure 29: State Transition Diagram for IpFwLoadManager

9.4.4.1.1 Idle State

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

9.4.4.1.2 Notification Suspended State

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

9.4.4.1.3 Active State

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

9.4.5 Event Notification State Transition Diagrams

There are no State Transition Diagrams defined for Event Notification.

10 Service Properties

10.1 Service Property Types

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

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

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

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

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

Property type name	Description	Example value (array of strings)	Interpretation of example value
BOOLEAN_SET	set of Booleans	{"FALSE"}	The set of Booleans consisting of the Boolean "false".
INTEGER_SET	set of integers	{"1", "2", "5", "7"}	The set of integers consisting of the integers 1, 2, 5 and 7.
STRING_SET	set of strings	{"Sophia", "Rijen"}	The set of strings consisting of the string "Sophia" and the string "Rijen"
ADDRESSRANGE_SET (Deprecated)	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.
XML_ADDRESS_RANGE_SET	set of values of TpAddressRange. Values of TpAddressRange are described using XML. An XML schema is provided below for this purpose.	{"<AddressRangeSet> <AddressRange> <Plan>P_ADDRESS_PLAN_E164</Plan> <AddrString>123*</AddrString> </AddressRange> <AddressRange> <Plan>P_ADDRESS_PLAN_E164</Plan> <AddrString>234*</AddrString> </AddressRange> </AddressRangeSet>"}	Any addresses starting with 123 or starting with 456 in the E.164 Address Plan

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.

The value of XML_ADDRESS_RANGE_SET should comply with the following XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="AddressRangeSet">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="AddressRange" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Plan" type="xs:string" default="P_ADDRESS_PLAN_ANY"/>
                            <xs:element name="AddrString" type="xs:string"/>
                            <xs:element name="Name" type="xs:string" minOccurs="0"/>
                            <xs:element name="SubAddressString" type="xs:string" minOccurs="0"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

An example usage could be:

```
{"<?xml version="1.0" encoding="UTF-8"?>
<AddressRangeSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xml_address_range_set.xsd">
      <AddressRange>
            <Plan>P_ADDRESS_PLAN_E164</Plan>
            <AddrString>789*</AddrString>
      </AddressRange>
      <AddressRange>
            <Plan>P_ADDRESS_PLAN_ANY</Plan>
            <AddrString>123*</AddrString>
      </AddressRange>
      <AddressRange>
            <Plan>P_ADDRESS_PLAN_SIP</Plan>
            <AddrString>&lt;sip:*@parlay.org&gt;</AddrString>
            <Name/>
      </AddressRange>
</AddressRangeSet>"}
```

Note that the final address range corresponds to any sip address @parlay.org, i.e. <sip:*@parlay.org>.

10.2 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
- Operation Set.

10.2.1 Service Name

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

10.2.2 Service Version

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

10.2.3 Service Instance ID

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

10.2.4 Service Instance Description

This property contains a textual description of the service.

10.2.5 Product Name

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

10.2.6 Product Version

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

10.2.7 Supported Interfaces

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

10.2.8 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"}.

11 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.

All data types referenced but not defined in this clause are common data definitions which may be found in ES 201 915-2.

11.1 Common Framework Data Definitions

11.1.1 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.

11.1.2 TpClientAppIDList

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

11.1.3 TpDomainID

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

	Tag Element Type	
	TpDomainIDType	

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

11.1.4 TpDomainIDType

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

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

11.1.5 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).

11.1.6 TpPropertyName

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

11.1.7 TpPropertyValue

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

11.1.8 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

11.1.9 TpPropertyList

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

11.1.10 TpEntOpIDList

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

11.1.11 TpFwID

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

11.1.12 TpService

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

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

11.1.13 TpServiceList

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

11.1.14 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	

11.1.15 TpServiceID

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

11.1.16 TpServiceIDList

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

11.1.17 TpServiceInstanceID

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

11.1.18 TpServiceSpecString

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

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

11.1.19 TpServiceTypeProperty

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

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

11.1.20 TpServiceTypePropertyList

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

11.1.21 TpServiceTypePropertyMode

This type defines SCF property modes.

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

11.1.22 TpServicePropertyTypeName

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

11.1.23 TpServicePropertyName

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

11.1.24 TpServicePropertyNameList

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

11.1.25 TpServicePropertyValue

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

11.1.26 TpServicePropertyValueList

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

11.1.27 TpServiceProperty

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

Sequence Element Name	Sequence Element Type	Documentation
ServicePropertyName	TpServicePropertyName	
ServicePropertyValueList	TpServicePropertyValueList	

11.1.28 TpServicePropertyList

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

11.1.29 TpServiceSupplierID

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

11.1.30 TpServiceTypeDescription

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

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

11.1.31 TpServiceTypeName

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

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

Character String Value	Description
P_PAM_PRESENCE_AND_AVAILABILITY	The name of PAM presentity SCF
P_PAM_EVENT_MANAGEMENT	The name of PAM watcher SCF
P_PAM_PROVISIONING	The name of PAM provisioning SCF

11.1.32 TpServiceTypeNameList

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

11.1.33 TpSubjectType

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

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

11.2 Event Notification Data Definitions

11.2.1 TpFwEventName

Defines the name of event being notified.

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

11.2.2 TpFwEventCriteria

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

	Tag Element Type	
	`TpFwEventName`	

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	`TpString`	`EventNameUndefined`
P_EVENT_FW_ SERVICE_AVAILABLE	TpServiceTypeNameList	ServiceTypeNameList
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceTypeNameList	UnavailableServiceTypeNameList

11.2.3 TpFwEventInfo

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

	Tag Element Type	
	`TpFwEventName`	

Tag Element Value	Choice Element Type	Choice Element Name
P_EVENT_FW_NAME_UNDEFINED	TpString	EventNameUndefined
P_EVENT_FW_ SERVICE_AVAILABLE	TpServiceIDList	ServiceIDList
P_EVENT_FW_SERVICE_UNAVAILABLE	TpServiceIDList	UnavailableServiceIDList

11.3 Trust and Security Management Data Definitions

11.3.1 TpAccessType

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

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

11.3.2 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 clients with the opportunity to use an alternative to the OSA API Level Authentication interface. This can for example be an implementation specific authentication mechanism, e.g. CORBA Security, or a proprietary Authentication interface supported by the Network Operator. OSA API Level Authentication is the default authentication method. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_'. The following values are defined.

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

11.3.3 TpEncryptionCapability

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

String Value	Description
NULL	An empty (NULL) string indicates no client capabilities.
P_DES_56	A simple transfer of secret information that is shared between the client application and the Framework with protection against interception on the link provided by the DES algorithm with a 56-bit shared secret key. The ECB mode of DES is to be used.
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. Use of the P_DES_128 value of TpEncryptionCapability is deprecated, as DES cannot be used with a 128-bit 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 1 024-bit keys.
P_TDEA	The Triple-DES or TDEA algorithm with three 56-bit secret keys. The key exchange is handled separately, and may permit use of three, two or only one unique key. The TECB mode of Triple-DES is to be used.

11.3.4 TpEncryptionCapabilityList

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

11.3.5 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).

11.3.6 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 intends to access another.

11.3.7 TpInterfaceName

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

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

11.3.8 TpInterfaceNameList

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

11.3.9 TpServiceToken

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

11.3.10 TpSignatureAndServiceMgr

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

Sequence Element Name	Sequence Element Type
DigitalSignature	TpOctetSet
ServiceMgrInterface	IpServiceRef

The digitalSignature is the signed version of a hash of the service token and agreement text given by the client application. If no signing algorithm is used, the digitalSignature is the octet sequence 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.

11.3.11 TpSigningAlgorithm

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

String Value	Description
NULL	An empty (NULL) string indicates no signing algorithm is required
P_MD5_RSA_512	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit key.
P_MD5_RSA_1024	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public- key cryptography system using a 1 024-bit key

11.4 Integrity Management Data Definitions

11.4.1 TpActivityTestRes

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

11.4.2 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

11.4.3 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.

11.4.4 TpFaultStatisticsError

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

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

11.4.5 TpFaultStatsSet

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

11.4.6 TpActivityTestID

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

11.4.7 TpInterfaceFault

Defines the cause of the interface fault detected.

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

11.4.8 TpSvcUnavailReason

Defines the reason why a SCF is unavailable.

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

11.4.9 TpFwUnavailReason

Defines the reason why the Framework is unavailable.

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

11.4.10 TpLoadLevel

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

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

11.4.11 TpLoadThreshold

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

Sequence Element Name	Sequence Element Type
LoadThreshold	TpFloat

11.4.12 TpLoadInitVal

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

Sequence Element Name	Sequence Element Type
LoadLevel	TpLoadLevel
LoadThreshold	TpLoadThreshold

11.4.13 TpLoadPolicy

Defines the load balancing policy.

Sequence Element Name	Sequence Element Type
LoadPolicy	TpString

11.4.14 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

11.4.15 TpLoadStatisticList

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

11.4.16 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.	

11.4.17 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

11.4.18 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

11.4.19 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`

11.4.20 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

11.4.21 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`

11.5 Service Subscription Data Definitions

11.5.1 TpPropertyName

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

11.5.2 TpPropertyValue

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

11.5.3 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

11.5.4 TpPropertyList

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

11.5.5 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).

11.5.6 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

11.5.7 TpServiceContractID

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

11.5.8 TpServiceContractIDList

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

11.5.9 TpPersonName

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

11.5.10 TpPostalAddress

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

11.5.11 TpTelephoneNumber

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

11.5.12 TpEmail

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

11.5.13 TpHomePage

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

11.5.14 TpPersonProperties

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

11.5.15 TpPerson

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

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

11.5.16 TpServiceStartDate

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

11.5.17 TpServiceEndDate

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

11.5.18 TpServiceRequestor

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

11.5.19 TpBillingContact

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

11.5.20 TpServiceSubscriptionProperties

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

11.5.21 TpServiceContract

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

Sequence Element Name	Sequence Element Type
`ServiceContractID`	TpServiceContractID
`ServiceContractDescription`	TpServiceContractDescription

11.5.22 TpServiceContractDescription

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

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

11.5.23 TpClientAppProperties

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

11.5.24 TpClientAppDescription

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

Sequence Element Name	Sequence Element Type
`ClientAppID`	TpClientAppID
`ClientAppProperties`	TpClientAppProperties

11.5.25 TpSagID

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

11.5.26 TpSagIDList

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

11.5.27 TpSagDescription

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

11.5.28 TpSag

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

Sequence Element Name	Sequence Element Type
`SagID`	TpSagID
`SagDescription`	TpSagDescription

11.5.29 TpServiceProfileID

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

11.5.30 TpServiceProfileIDList

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

11.5.31 TpServiceProfile

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

Sequence Element Name	Sequence Element Type
`ServiceProfileID`	TpServiceProfileID
`ServiceProfileDescription`	TpServiceProfileDescription

11.5.32 TpServiceProfileDescription

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

Sequence Element Name	Sequence Element Type
`ServiceContractID`	TpServiceContractID
`ServiceStartDate`	TpServiceStartDate
`ServiceEndDate`	TpServiceEndDate
`ServiceTypeName`	TpServiceTypeName (see Note)
`ServiceSubscriptionProperties`	TpServiceSubscriptionProperties
NOTE: When the Framework returns a TpServiceProfileDescription to the enterprise operator, it should set this field to the same value as the corresponding field of the service contract; When the enterprise operator passes a TpServiceProfileDescription to the Framework, the Framework should ignore the value sent in this field to ensure interoperability; The enterprise operator should be required to set the field to the correct value when passing a TpServiceProfileDescription to the Framework.	

12 Exception Classes

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

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

Each exception class contains the following structure:

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

Annex A (normative): OMG IDL description of Framework

The OMG IDL representation of this interface specification is contained in text files (fw_data.idl, fw_if_access.idl, fw_if_app.idl, fw_if_entop.idl, fw_if_service.idl contained in archive es_20191503v010501m0.zip) which accompany the present document.

Annex B (informative): Contents of 3GPP OSA R4 Framework

All parts of the present document, except clause 8, Framework to Enterprise Operator API, are relevant for 3GPP TS 29.198-3 V4 (Release 4).

Annex C (informative): Record of changes

The following is a list of the changes made to this specification for each release. The list contains the names of all changed, deprecated, added or removed items in the specifications and not the actual changes. Any type of change information that is important to the reader is put in the *Others* part of this annex.

Changes are specified as changes to the prior major release, but every minor release will have its own part of the table allowing the reader to know when the actual change was made.

C.1 Interfaces

C.1.1 New

Identifier	Comments
Interfaces added in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Interfaces added in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Interfaces added in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.1.2 Deprecated

Identifier	Comments
Interfaces deprecated in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Interfaces deprecated in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Interfaces deprecated in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.1.3 Removed

Identifier	Comments
Interfaces removed in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Interfaces removed in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Interfaces removed in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.2 Methods

C.2.1 New

Identifier	Comments
Methods added in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Methods added in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
IpAppLoadManager.createLoadLevelNotification	
IpAppLoadManager.destroyLoadLevelNotification	
IpSvcLoadManager.createLoadLevelNotification	
IpSvcLoadManager.destroyLoadLevelNotification	
Methods added in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.2.2 Deprecated

Identifier	Comments
Methods deprecated in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Methods deprecated in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Methods deprecated in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.2.3 Modified

Identifier	Comments
Methods modified in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
IpAppFaultManager.appUnavailableInd	parameter serviceID added
Methods modified in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
IpLoadManager.createLoadLevelNotification	Behaviour changed to always send first load level notification
IpLoadManager.resumeNotification	Behaviour changed to always send first load level notification
IpAppLoadManager.resumeNotification	Behaviour changed to always send first load level notification
IpFwLoadManager.createLoadLevelNotification	Behaviour changed to always send first load level notification
IpFwLoadManager.resumeNotification	Behaviour changed to always send first load level notification
IpSvcLoadManager.resumeNotification	Behaviour changed to always send first load level notification
Methods modified in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.2.4 Removed

Identifier	Comments
Methods removed in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Methods removed in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Methods removed in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.3 Data Definitions

C.3.1 New

Identifier	Comments
Data Definitions added in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Data Definitions added in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Data Definitions added in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.3.2 Modified

Identifier	Comments
Data Definitions modified in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Data Definitions modified in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Data Definitions modified in ES 201 915-3 version 1.5.1 (Parlay 3.4)	
TpEncryptionCapability	P_DES_128 deprecated, P_TDEA added, mode of P_DES_56 clarified

C.3.3 Removed

Identifier	Comments
Data Definitions removed in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Data Definitions removed in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Data Definitions removed in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.4 Service Properties

C.4.1 New

Identifier	Comments
Service Properties added in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Service Properties added in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Service Properties added in ES 201 915-3 version 1.5.1 (Parlay 3.4)	
XML_ADDRESS_RANGE_SET	Service Property Type, to replace ADDRESSRANGE_SET

C.4.2 Deprecated

Identifier	Comments
Service Properties deprecated in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Service Properties deprecated in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Service Properties deprecated in ES 201 915-3 version 1.5.1 (Parlay 3.4)	
ADDRESSRANGE_SET	Service Property Type, replaced by XML_ADDRESS_RANGE_SET

C.4.3 Modified

Identifier	Comments
Service Properties modified in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Service Properties modified in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Service Properties modified in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.4.4 Removed

Identifier	Comments
Service Properties removed in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Service Properties removed in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Service Properties removed in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.5 Exceptions

C.5.1 New

Identifier	Comments
Exceptions added in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Exceptions added in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Exceptions added in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.5.2 Modified

Identifier	Comments
Exceptions modified in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Exceptions modified in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Exceptions modified in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.5.3 Removed

Identifier	Comments
Exceptions removed in ES 201 915-3 version 1.3.1 (Parlay 3.2)	
Exceptions removed in ES 201 915-3 version 1.4.1 (Parlay 3.3)	
Exceptions removed in ES 201 915-3 version 1.5.1 (Parlay 3.4)	

C.6 Others

None.

History

Document history		
V1.1.1	February 2002	Publication
V1.2.1	July 2002	Publication
V1.3.1	October 2002	Publication
V1.4.1	July 2003	Publication
V1.5.1	November 2004	Membership Approval Procedure MV 20050128: 2004-11-30 to 2005-01-28