3GPP TSG CN Plenary Meeting #17 **NP-020428**
4 - 6 September 2002, Biarritz, FRANCE

Source: **CN5 (OSA)**

Title: **Rel-5 CRs 29.198-03 OSA API Part 3: Framework**

Agenda item: **8.2**

Document for: **APPROVAL**

Doc-1st-Level	Spec	CR	Rev	Phase	Subject	Cat	Version-Current	Doc-2nd-Level	Workitem
NP-020428	29.198-03	046	-	Rel-5	Correction to description of TpServicePropertyTypeName	F	5.0.0	N5-020755	OSA2
NP-020428	29.198-03	047	-	Rel-5	Remove undefined exception in registerService	F	5.0.0	N5-020606	OSA2
NP-020428	29.198-03	048	-	Rel-5	Remove ServiceIDs from IpFwFaultManager.genFaultStatsRecordReq()	F	5.0.0	N5-020691	OSA2
NP-020428	29.198-03	049	-	Rel-5	Correct appUnavailableInd and related methods	F	5.0.0	N5-020692	OSA2
NP-020428	29.198-03	050	-	Rel-5	Remove unusable exception from IpFaultManager.appActivityTestRes()	F	5.0.0	N5-020693	OSA2
NP-020428	29.198-03	051	-	Rel-5	Clarify the sequence of events in signing the service agreement	F	5.0.0	N5-020695	OSA2
NP-020428	29.198-03	052	-	Rel-5	Correct use of electronic signatures	F	5.0.0	N5-020704	OSA2
NP-020428	29.198-03	053	-	Rel-5	Addition of Sequence Diagrams for terminateAccess	F	5.0.0	N5-020705	OSA2
NP-020428	29.198-03	054	-	Rel-5	Add indication what part of service agreement must be signed	F	5.0.0	N5-020710	OSA2
NP-020428	29.198-03	055	-	Rel-5	Add text to clarify requirements on support of methods	F	5.0.0	N5-020716	OSA2
NP-020428	29.198-03	056	-	Rel-5	Introduce types and modes for generic properties	F	5.0.0	N5-020741	OSA2
NP-020428	29.198-03	057	-	Rel-5	Correction on use of NULL in Framework API	A	5.0.0	N5-020751	OSA2
NP-020428	29.198-03	058	-	Rel-5	Add Negotiation of Authentication Mechanism for OSA level Authentication	F	5.0.0	N5-020760	OSA2

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020755**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **046** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Correction to description of TpServicePropertyTypeName
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	TpServicePropertyName and TpServicePropertyTypeName are currently defined to be the same thing. As a result it is not clear how to fully specify a service type. The service type definition is fundamental for all service registrations.
Summary of change:	⌘	Correct definition of TpServicePropertyTypeName
Consequences if not approved:	⌘	No clear mechanism for definition of service types exist resulting in possibly multiple vendor interpretations, and no support for multi-vendor service registration.

Clauses affected:	⌘	10.1;11.1.21
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

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

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

********** START OF CHANGE **********

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 service property types.

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

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

.

********** END OF FIRST CHANGE **********

********** START OF SECOND CHANGE **********

11.1.21 TpServicePropertyTypeName

This data type is identical to TpString and describes a valid SCF property type name. ~~The valid SCF property names are listed in the SCF data definition.~~ Valid service property type names are detailed in 10.1.

11.1.22 TpServicePropertyName

This data type is identical to TpString. It defines a valid SCF property name. The valid service property names are detailed in 10.2 and in the SCF data definition.

********** **END OF SECOND CHANGE** **********

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020606**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **047** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Remove undefined exception in registerService
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	In the text of selectService an exception is mentioned that is no longer valid-and in the raises two exceptions are mentioned regarding the serviceID, however, the serviceID is not an input paramter, but a result of the method, so these exceptions are never raised.
Summary of change:	⌘	The corresponding part of the text is removed. Both serviceID related exceptions are removed. This change is backward compatible (level0, but not level2).
Consequences if not approved:	⌘	Unclear specification. Confusion.

Clauses affected:	⌘	9.3.1.1
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

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

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

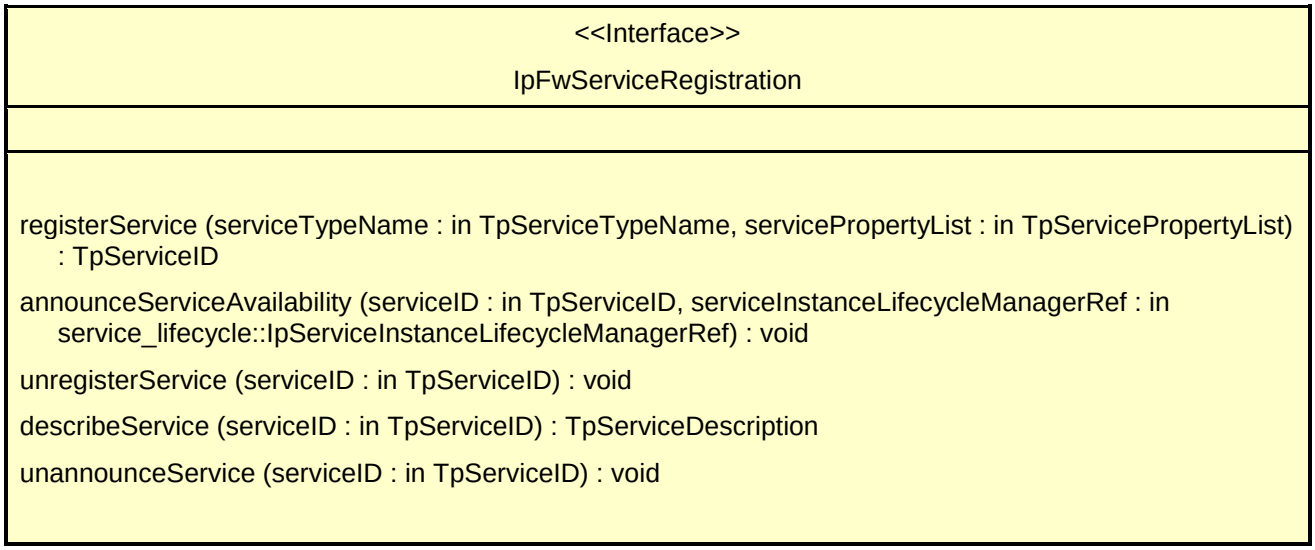
Proposed Changes

The following changes are proposed to 29.198-03:

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.



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

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **048** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Remove ServiceIDs from IpFwFaultManager.genFaultStatsRecordReq()
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	genFaultStatsRecordReq on IpSvcFaultManager and genFaultStatsRecordRes/Err on IpFwFaultManager contain parameter serviceIDs : TpServiceIDList. But these interfaces are between an instance of a service and the framework. It seems strange that the framework should request a service instance to record fault statistics for other service instances - this implies a dependance, not between service instances, but between different services (the parameter is not of type TpServiceInstanceID). Clearly, this parameter is a left-over from similar methods on the FW-Application interfaces. However, it is indicated that this parameter shall not be an empty list, and it is not described what might occur if the serviceID packed into this parameter, to prevent it being empty, did not correspond to the serviceID associated with the service instance which invokes these methods (genFaultStatsRecordRes/Err) or on which this method (genFaultStatsRecordReq) is invoked. These methods cannot ever operate as described, therefore they should be corrected. This requires deprecation of the existing methods and their replacement by generateFaultStatsRecordReq/Res/Err.
Summary of change:	⌘	Deprecate IpSvcFaultManager.genFaultStatsRecordReq() and add a similar new method generateFaultStatsRecordReq() without the serviceIDs parameter. Deprecate IpFwFaultManager.genFaultStatsRecordRes/Err() and add similar new methods generateFaultStatsRecordRes/Err() without the serviceIDs parameter.
Consequences if not approved:	⌘	The Framework may be unable to retreive fault statistics from the service, since these methods cannot operate as they are described. Their behaviour is undefined.

Clauses affected:	⌘	8.3.4
Other specs	⌘	☐ Other core specifications ⌘

affected:	☐ Test specifications ☐ O&M Specifications	
Other comments: ⌘		

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

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

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

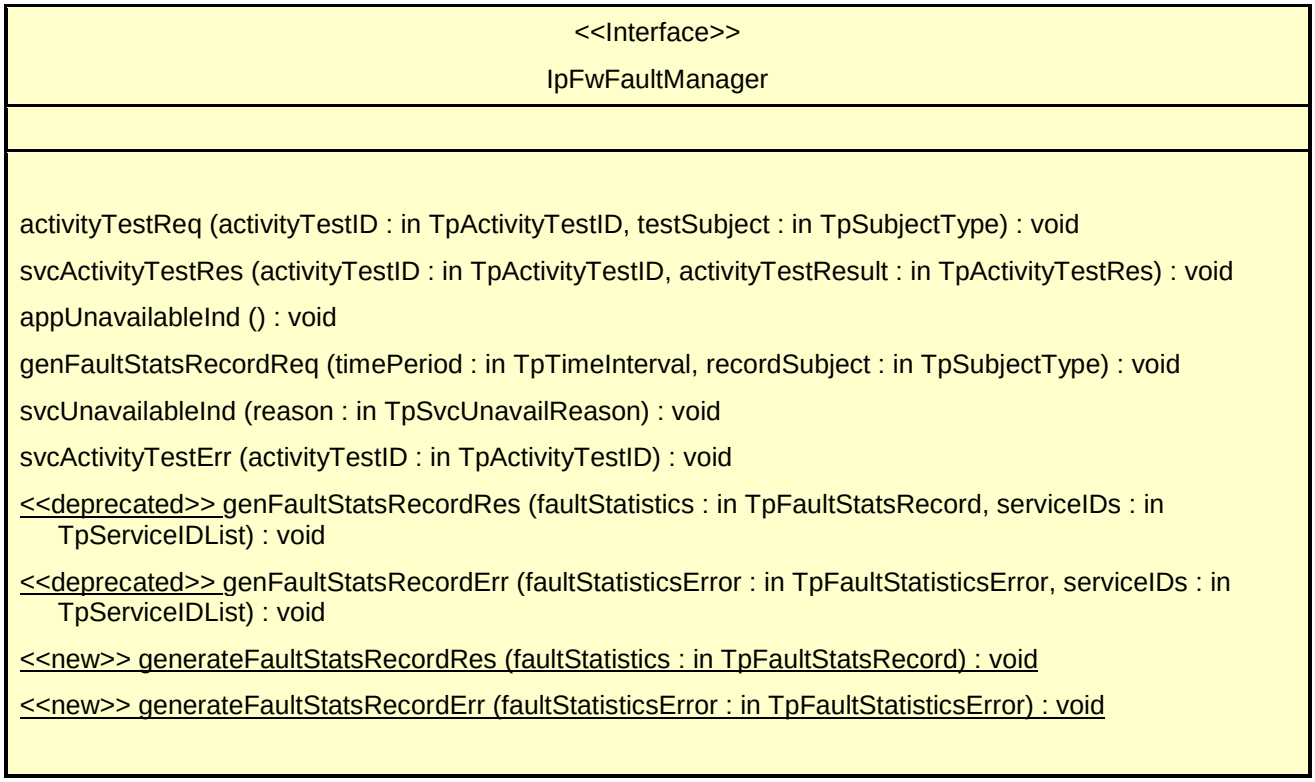
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

8.3.4 Integrity Management Interface Classes

8.3.4.1 Interface Class IpFwFaultManager

Inherits from: IpInterface.

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



Method

activityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

testSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

svcActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions,P_INVALID_ACTIVITY_TEST_ID

Method

appUnavailableInd()

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

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

Method

genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

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

recordSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

svcUnavailableInd()

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

Parameters

reason : in TpSvcUnavailReason

Identifies the reason for the service instance's unavailability.

Raises

TpCommonExceptions

Method

svcActivityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

<<deprecated>> genFaultStatsRecordRes()

This method is deprecated and will be removed in a later release. It cannot be used as described, since the serviceIDs parameter has no meaning. It is replaced with generateFaultStatsRecordRes().

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

<<deprecated>> genFaultStatsRecordErr()

This method is deprecated and will be removed in a later release. It cannot be used as described, since the serviceIDs parameter has no meaning. It is replaced with generateFaultStatsRecordErr().

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

Method

<<new>> generateFaultStatsRecordRes()

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.

Raises

TpCommonExceptions

Method

<<new>> generateFaultStatsRecordErr()

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.

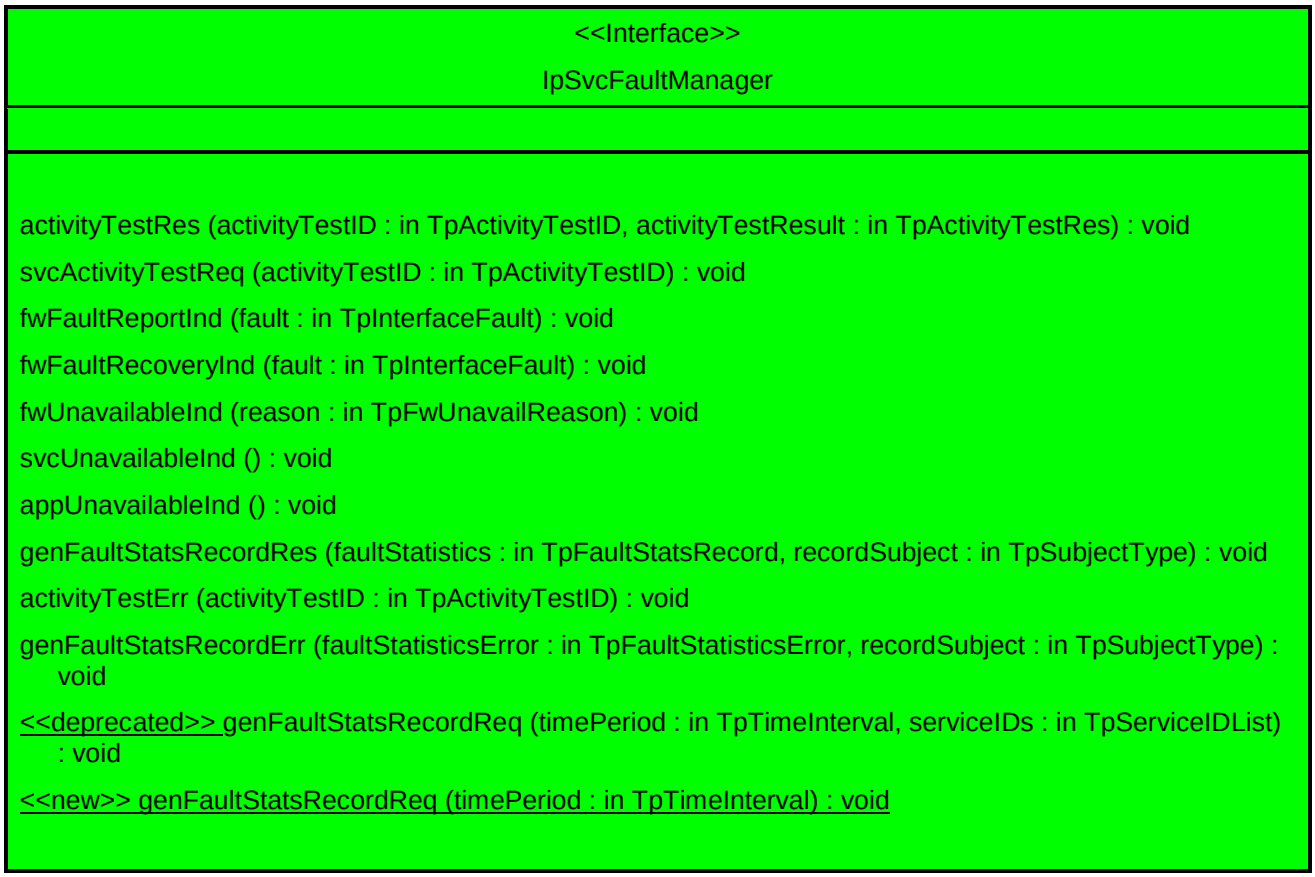
Raises

TpCommonExceptions

8.3.4.2 Interface Class IpSvcFaultManager

Inherits from: IpInterface.

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



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

<<deprecated>> genFaultStatsRecordReq()

This method is deprecated and will be removed in a later release. It cannot be used as described, since the serviceIDs parameter has no meaning. It is replaced with generateFaultStatsRecordReq().

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. A null value leaves this 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

Method

<<new>> generateFaultStatsRecordReq()

This method is used by the framework to solicit fault statistics from the service instance, 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 instance must produce a fault statistics record during the specified time interval, which is returned to the framework using the genFaultStatsRecordRes operation on the IpFwFaultManager interface.

Parameters

timePeriod : in TpTimeInterval

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

Raises

TpCommonExceptions

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **049** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Correct appUnavailableInd and related methods
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change: ⌘

Currently, the appUnavailableInd and svcUnavailableInd methods on IpFaultManager, IpAppFaultManager, IpFWFaultManager and IpSvcFaultManager all imply that the application or service instance is broken, can't be fixed, and is to be killed.
While this might be the reason why such methods are invoked, they should not automatically lead to deletion of service instances and termination of service agreements, using or not using the normal mechanisms (terminateServiceAgreement, destroyServiceManager etc.) until the application, framework or service instance has had the possibility to test and attempt recovery.
In particular, forcing termination of service agreements etc. by means of these methods is a perfect back-door for denial of service attacks, since these methods carry no identification or digital signature, unlike the terminateServiceAgreement method, yet currently have much the same effect.
Even requiring applications or the framework to invoke the protected terminateServiceAgreement themselves following this method call still amounts to a potential denial of service attack, with the interesting detail that applications or service instances or framework are forced to deny themselves service!

In particular, IpFaultManager.appUnavailableInd() is unnecessary, since terminateServiceAgreement invoked by the application is equally effective for one service instance, and IpFaultManager.appUnavailableInd() in its current form can only notify the application being unavailable for one service instance. If the application wishes to instantly notify that it is unavailable for all service instances, it could invoke the IpAccess.endAccess() method.
If IpFaultManager.appUnavailableInd() is to be kept but with different behaviour, its parameter should be removed, since the Framework will know which service instances the application is using.

Summary of change: ⌘

Remove all requirement or recommendation that applications, the framework or services cease their use of each other when methods appUnavableInd() or svcUnavailableInd() are called in IpAppFaultManager, IpFaultManager, IpFwFaultManager and IpSvcFaultManager.

		Deprecate IpFaultManager.appUnavailableInd()
Consequences if not approved:	⌘	Despite all efforts to clear up the authentication mechanism and tighten security, a back-door will remain for denial of service attacks on OSA applications, frameworks or SCFs.

Clauses affected:	⌘	7, 8		
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

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

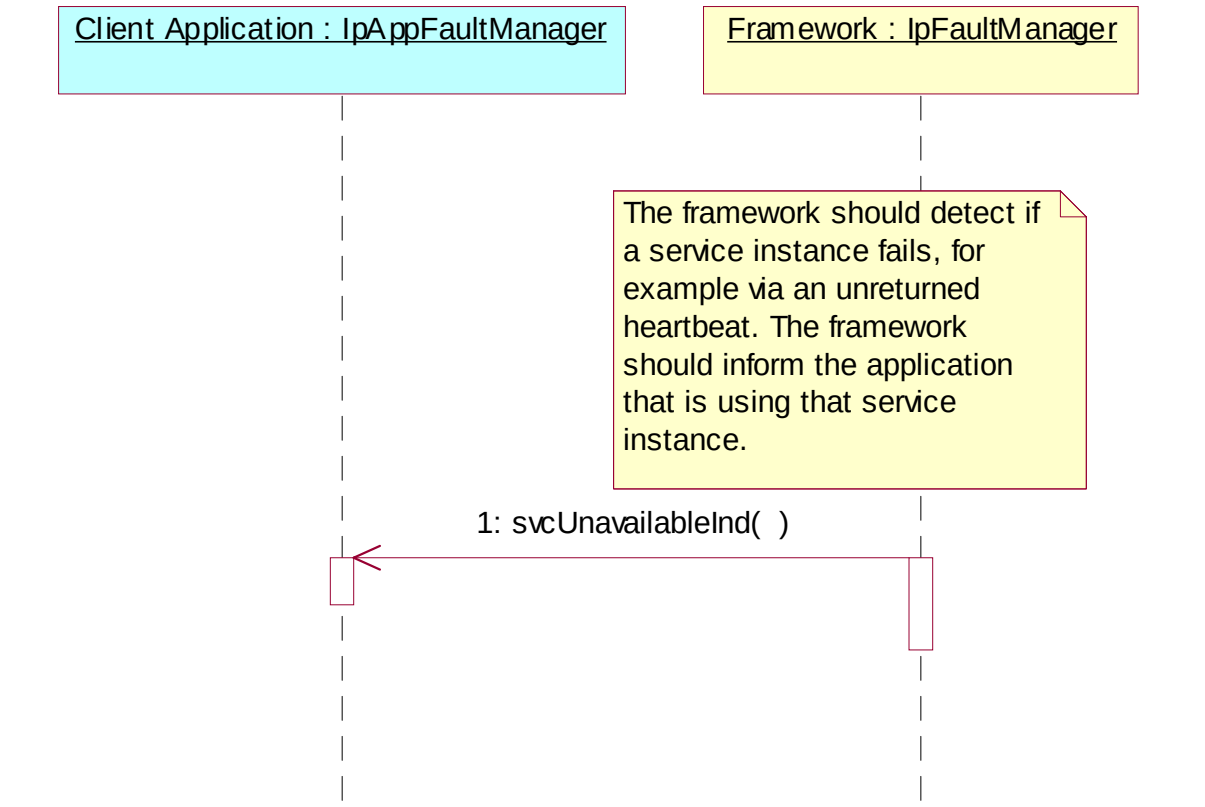
1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.

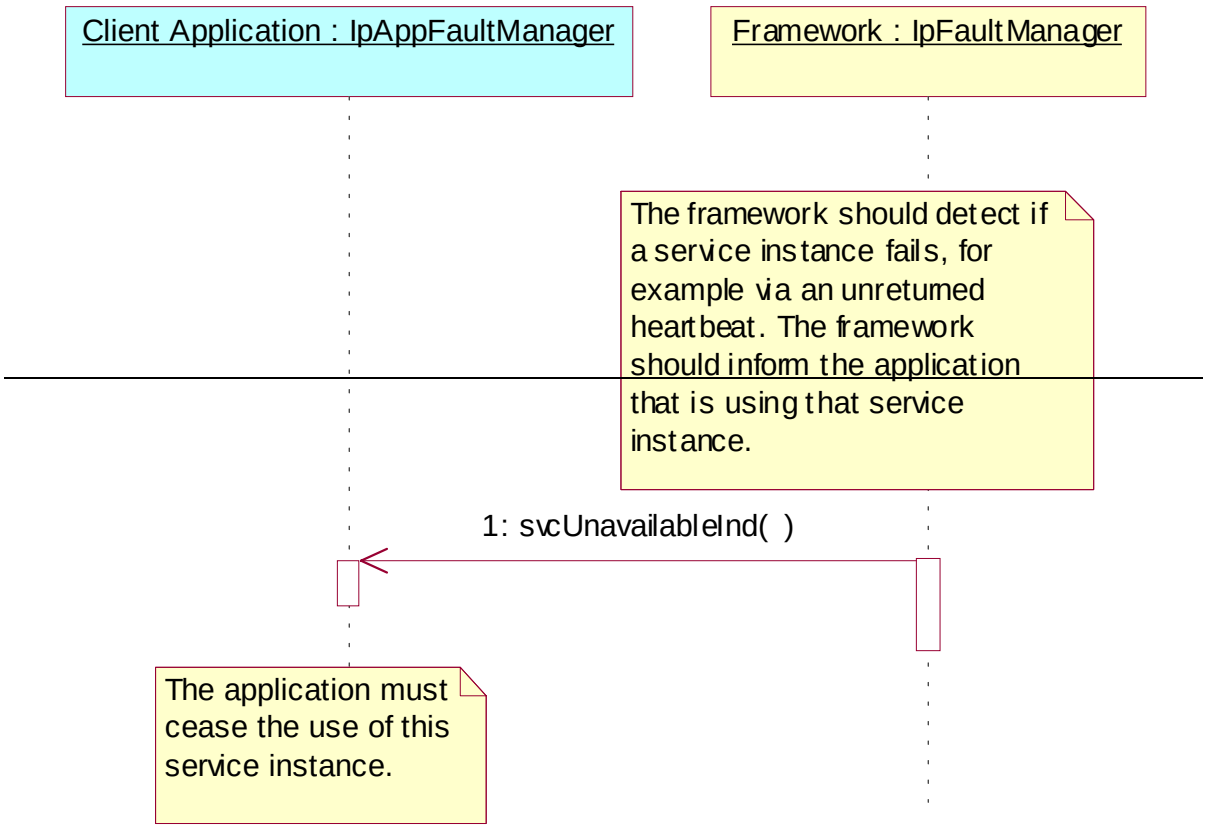
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest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

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

7.1.2.7 Fault Management: Framework detects a Service failure

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


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


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

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.~~

7.3.3 Integrity Management Interface Classes

7.3.3.1 Interface Class IpAppFaultManager

Inherits from: IpInterface.

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

<<Interface>> IpAppFaultManager
activityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void appActivityTestReq (activityTestID : in TpActivityTestID) : void fwFaultReportInd (fault : in TpInterfaceFault) : void fwFaultRecoveryInd (fault : in TpInterfaceFault) : void svcUnavailableInd (serviceID : in TpServiceID, reason : in TpSvcUnavailReason) : void genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord, serviceIDs : in TpServiceIDList) : void fwUnavailableInd (reason : in TpFwUnavailReason) : void activityTestErr (activityTestID : in TpActivityTestID) : void genFaultStatsRecordErr (faultStatisticsError : in TpFaultStatisticsError, serviceIDs : in TpServiceIDList) : void appUnavailableInd (serviceID : in TpServiceID) : void genFaultStatsRecordReq (timePeriod : in TpTimeInterval) : void

Method

`activityTestRes()`

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

Parameters

`activityTestID : in TpActivityTestID`

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

`activityTestResult : in TpActivityTestRes`

The result of the activity test.

Method
appActivityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

Method
fwFaultReportInd()

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

Parameters

fault : in TpInterfaceFault

Specifies the fault that has been detected by the framework.

Method
fwFaultRecoveryInd()

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

Parameters

fault : in TpInterfaceFault

Specifies the fault from which the framework has recovered.

Method
svcUnavailableInd()

The framework invokes this method to inform the client application that ~~it can no longer use~~ may experience difficulties using 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

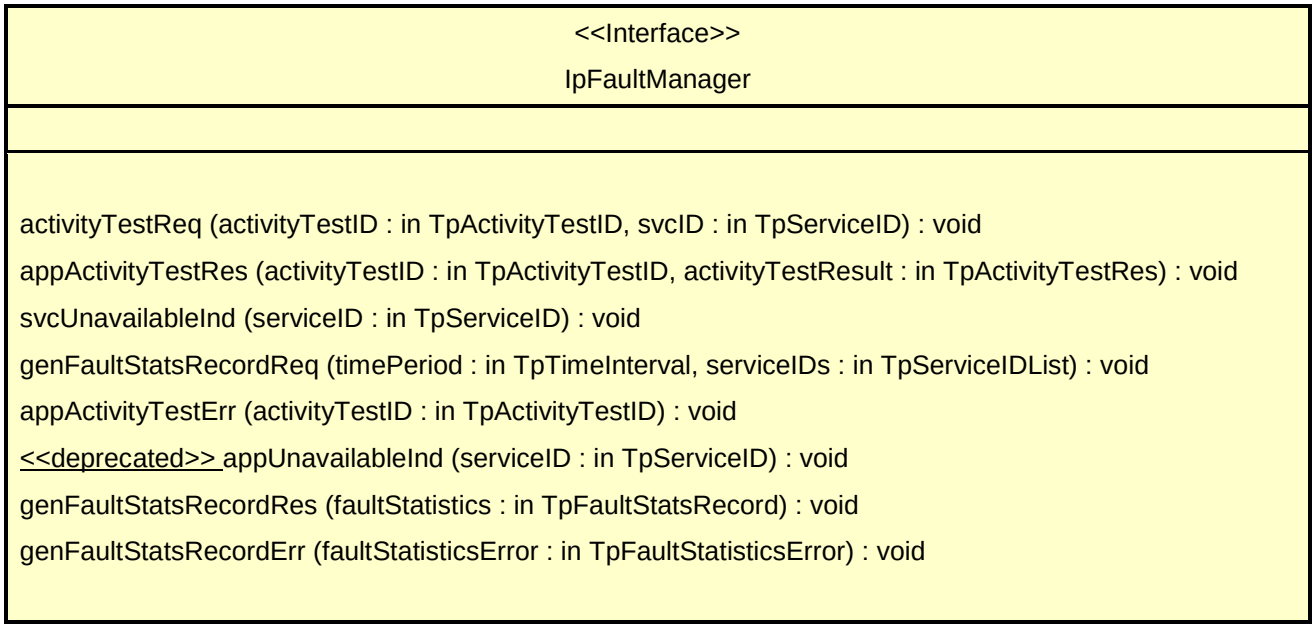
`timePeriod : in TpTimeInterval`

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



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 a null value.

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

appActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_INVALID_ACTIVITY_TEST_ID

Method

svcUnavailableInd()

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

Parameters

serviceID : in TpServiceID

Identifies the service that the application can no longer use.

Raises

TpCommonExceptions ,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

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

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

appActivityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

<<deprecated>> appUnavailableInd()

This method is deprecated and will be removed in a later release. It is strongly recommended not to implement this method. Applications can indicate they no longer use a particular service instance using

IpServiceAgreementManagement.terminateServiceAgreement(). Applications can indicate a fault with a particular service instance using IpFaultManager.svcUnavailableInd().

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

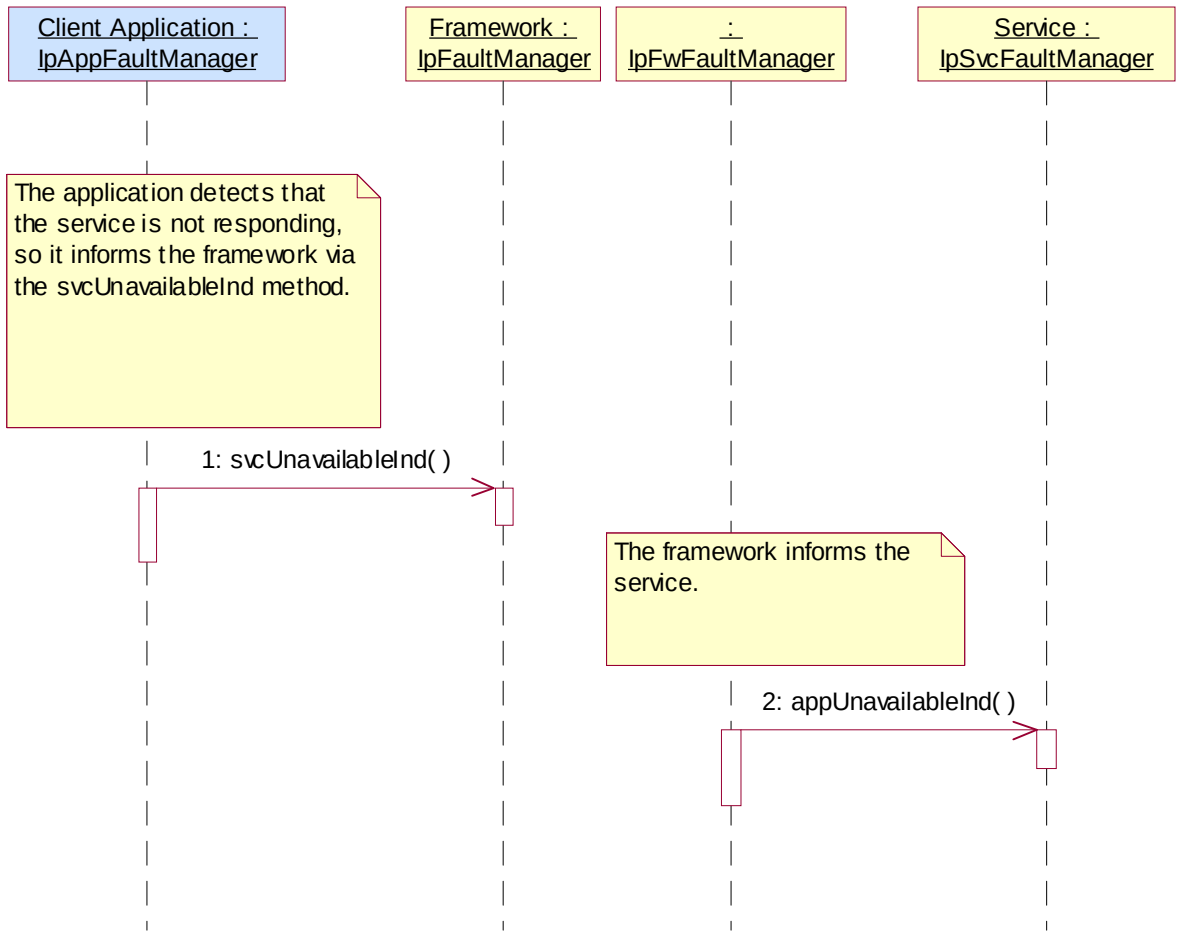
faultStatisticsError : in TpFaultStatisticsError

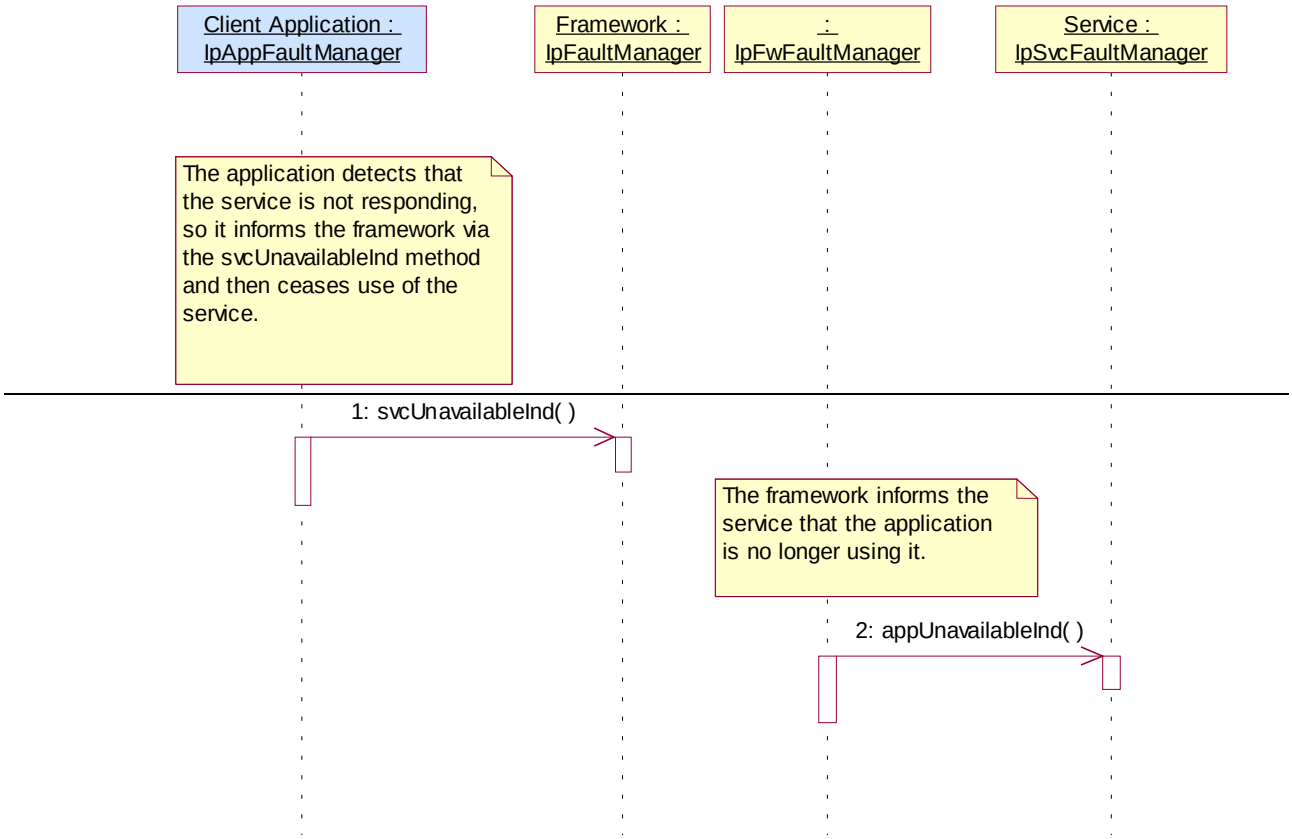
The fault statistics error.

Raises

TpCommonExceptions

8.1.4.7 Fault Management: Application detects service is unavailable


Client Application : IpAppFaultManager
Framework : IpFaultManager
: IpFwFaultManager
Service : IpSvcFaultManager
The application detects that the service is not responding, so it informs the framework via the svcUnavailableInd method.
1: svcUnavailableInd( )
The framework informs the service.
2: appUnavailableInd( )



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

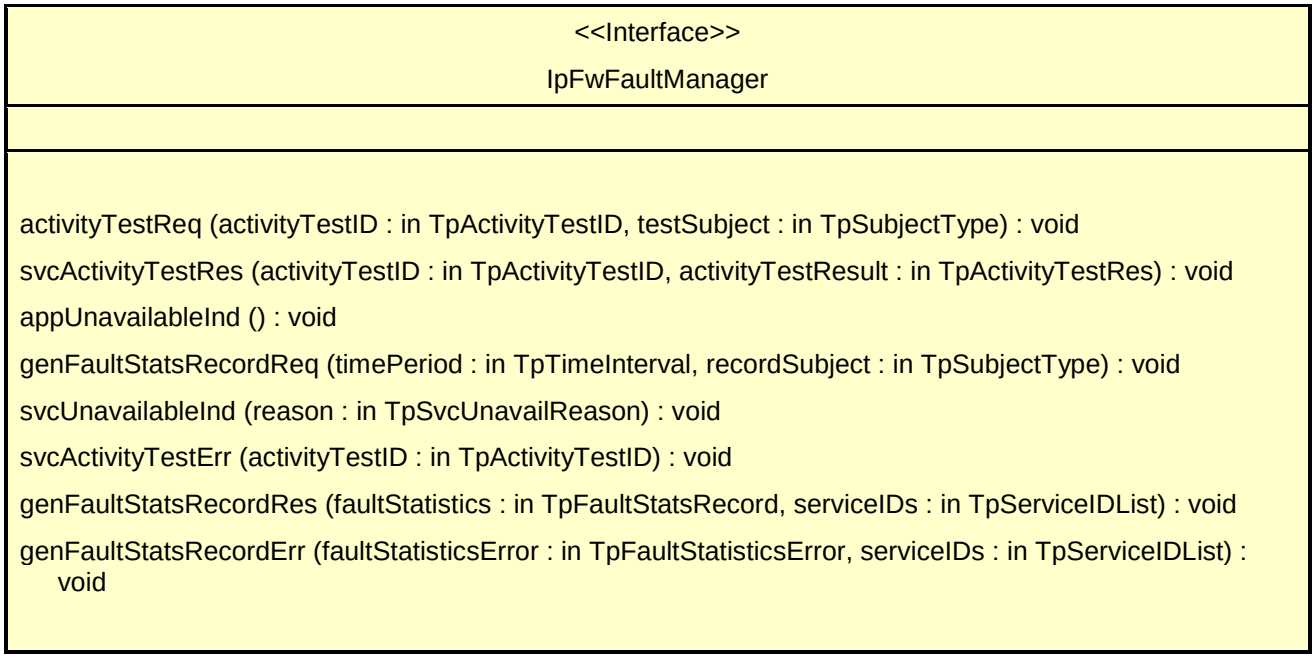
2: The framework informs the service instance that the client application was unable to get a response from it ~~and has ceased to be one of its users. The framework and service instance must carry out the appropriate updates to remove the client application as one of the users of this service instance.~~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

8.3.4 Integrity Management Interface Classes

8.3.4.1 Interface Class IpFwFaultManager

Inherits from: IpInterface.

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



Method

activityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

testSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method
svcActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions,P_INVALID_ACTIVITY_TEST_ID

Method
appUnavailableInd()

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

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

Method
genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

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

recordSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method

svcUnavailableInd()

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

Parameters

reason : in TpSvcUnavailReason

Identifies the reason for the service instance's unavailability.

Raises

TpCommonExceptions

Method

svcActivityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

genFaultStatsRecordRes()

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

Parameters

faultStatistics : in TpFaultStatsRecord

The fault statistics record.

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions

Method

genFaultStatsRecordErr()

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

Parameters

faultStatisticsError : in TpFaultStatisticsError

The fault statistics error.

serviceIDs : in TpServiceIDList

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

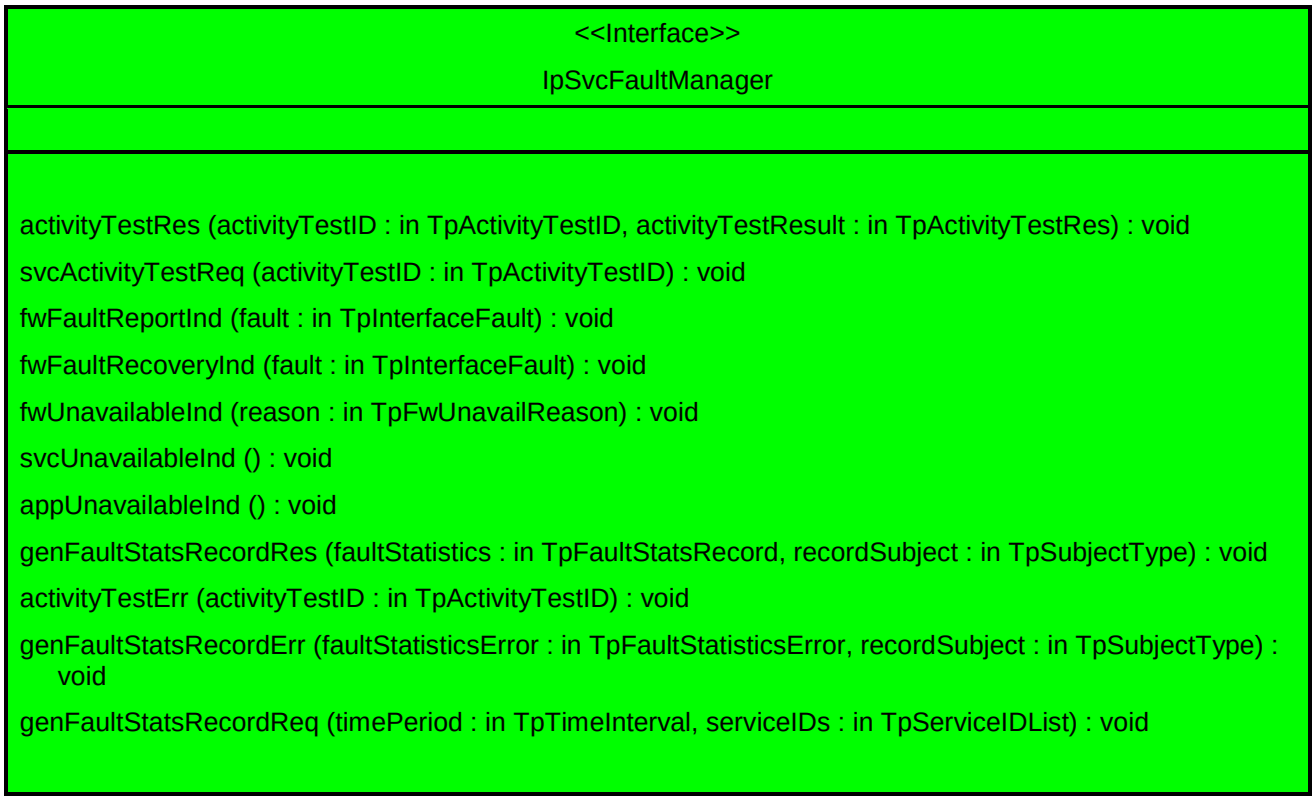
Raises

TpCommonExceptions

8.3.4.2 Interface Class IpSvcFaultManager

Inherits from: IpInterface.

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



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. A null value leaves this 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

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **050** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Remove unusable exception from IpFaultManager.appActivityTestRes()
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	Method IpFaultManager.appActivityTestRes() has exception P_INVALID_SERVICE_ID, yet there is no parameter containing a Service ID, and no reason to raise this exception. Removing this exception simplifies life slightly for application developers, since they don't have to include code to trap an exception which will never occur.
Summary of change:	⌘	Remove exception P_INVALID_SERVICE_ID from the exceptions list of IpFaultManager.appActivityTestRes() This change is entirely backwards compatible according to the backwards compatibility rules agreed at previous meetings: it is permitted to remove exceptions from the Framework or SCF side, since the application does not require modification if it already has code to trap these exceptions.
Consequences if not approved:	⌘	Application developers will be forced to include code in their applications to trap this exception, code which can never be used since the exception can never be raised.

Clauses affected:	⌘	7.3.3.2
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

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

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

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

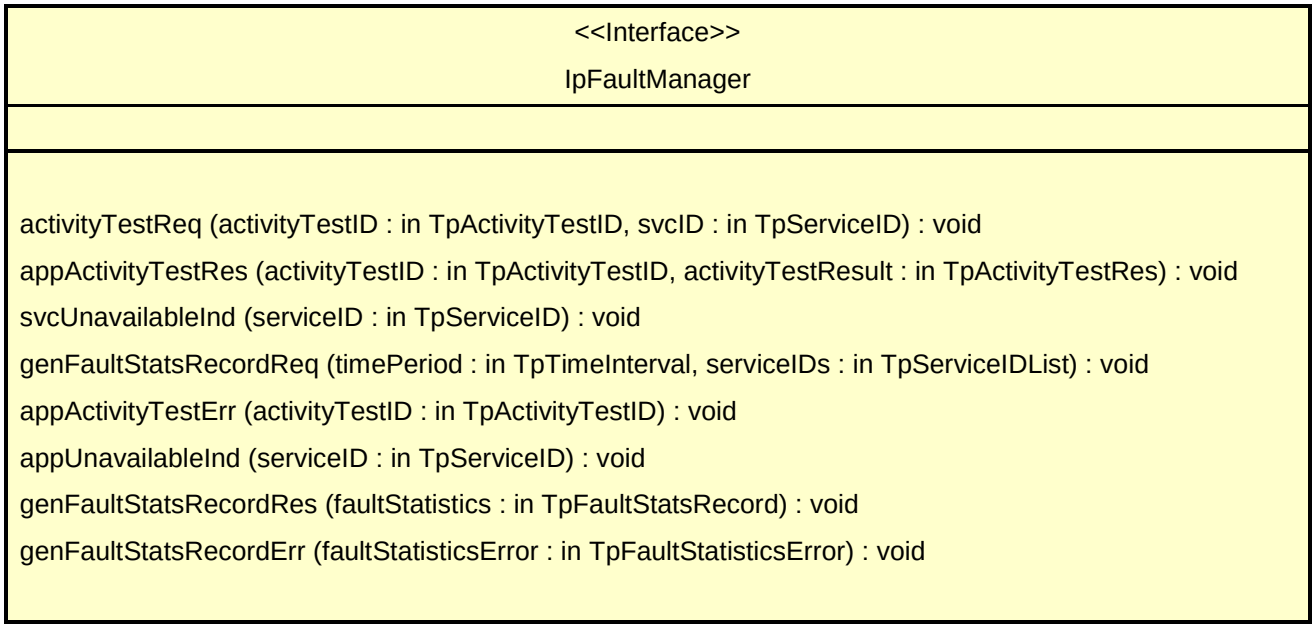
downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.

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

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.



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 a null value.

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

appActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions,~~P_INVALID_SERVICE_ID,~~P_INVALID_ACTIVITY_TEST_ID

Method

svcUnavailableInd()

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

Parameters

serviceID : in TpServiceID

Identifies the service that the application can no longer use.

Raises

TpCommonExceptions ,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

genFaultStatsRecordReq()

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

specified serviceID, the P_UNAUTHORISED_PARAMETER_VALUE exception shall be thrown. The extraInformation field of the exception shall contain the corresponding serviceID.

Parameters

timePeriod : in TpTimeInterval

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

serviceIDs : in TpServiceIDList

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

Raises

TpCommonExceptions, P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

appActivityTestErr()

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

Parameters

activityTestID : in TpActivityTestID

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

Raises

TpCommonExceptions, P_INVALID_ACTIVITY_TEST_ID

Method

appUnavailableInd()

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

Parameters

serviceID : in TpServiceID

Identifies the affected application.

Raises

TpCommonExceptions

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

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020695**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **051** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Clarify the sequence of events in signing the service agreement
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	In developing the test specifications for OSA, we have discovered that the sequence of events in signing the service agreement can be misunderstood. The only place the sequence of events is defined is in a sequence diagram. However, most sequence diagrams in the OSA specification show suggested behaviour, not the only permitted behaviour, so this is not sufficient.
Summary of change:	⌘	Add text to the signServiceAgreement() methods, and to initiateSignServiceAgreement(), to clarify the following sequence as the only permitted sequence: Application calls initiateSignServiceAgreement() on Framework; Framework calls signServiceAgreement() on Application; Application calls signServiceAgreement() on Framework;
Consequences if not approved:	⌘	Application developers may interpret the specifications wrongly and introduce behaviour to their applications which will never interwork, therefore their applications will never be able to get access to any service instances via the Framework.

Clauses affected:	⌘	7.3.2
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

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

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

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

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

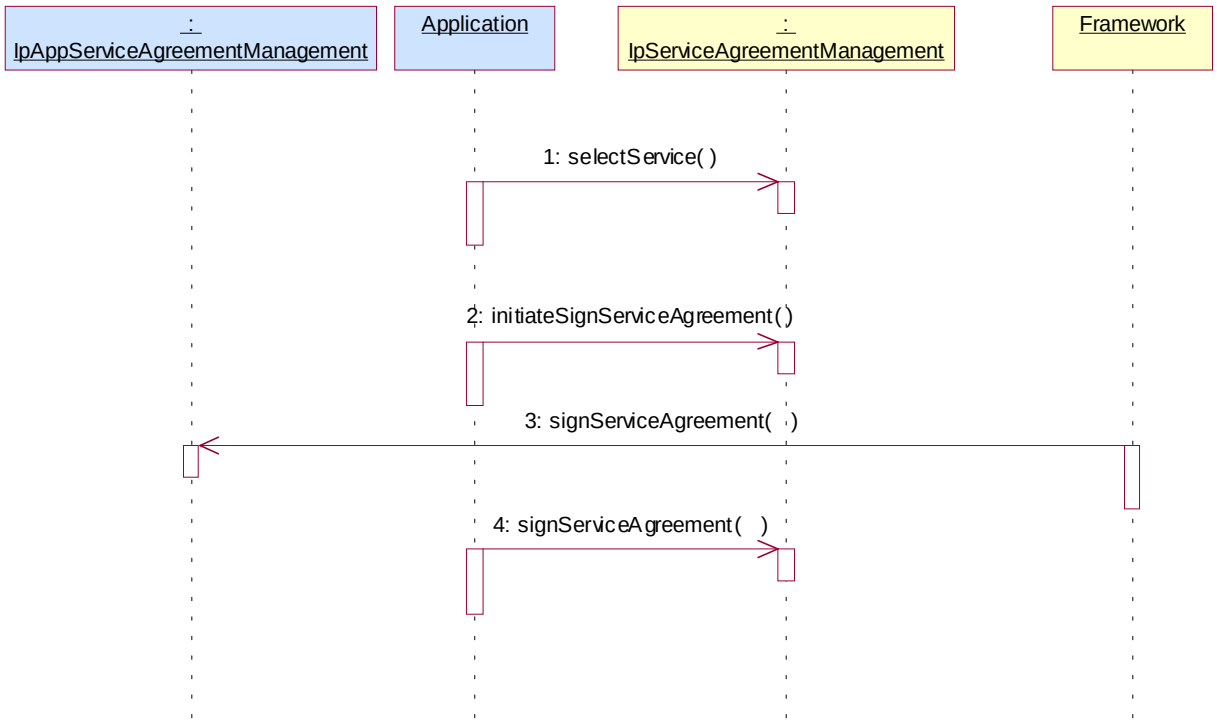
7.1.4 Service Agreement Management Sequence Diagrams

7.1.4.1 Service Selection

The following figure shows the process of selecting an SCF.

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

This is achieved by the following two steps:



1: Service Selection: first step - selectService

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

Input is:

· in serviceID

This identifies the SCF required.

And output:

· out serviceToken

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

2: Service Selection: second step - signServiceAgreement

In this second step an agreement is signed that allows the Application to use the chosen SCF version. And once this contractual details have been agreed, then the Application can be given the means to actually use it. The means are a reference to the manager interface of the SCF version (remember that a manager is an entry point to any SCF). By

calling the createServiceManager operation on the lifecycle manager the Framework retrieves this interface and returns it to the Application. The service properties suitable for this application are also fed to the SCF (via the lifecycle manager interface) in order for the SCS to instantiate an SCF version that is suitable for this application.

The sequence of events indicated above, where the application initiates the signature process by calling initiateSignServiceAgreement, and where the framework calls signServiceAgreement on the application's IpAppServiceAgreementManagement interface before the application calls signServiceAgreement on the frameworks's IpServiceAgreementManagement, is the only sequence permitted.

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:

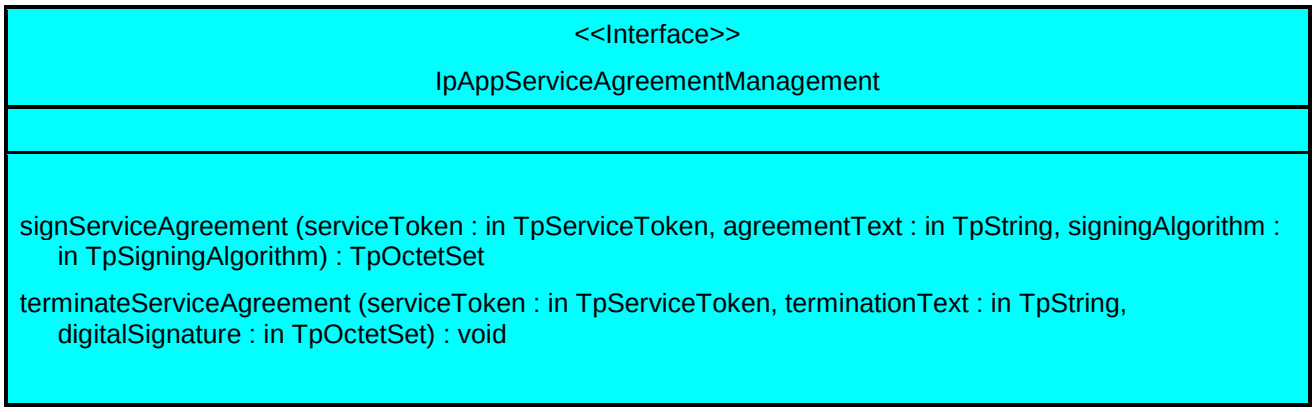
· out signatureAndServiceMgr

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.

7.3.2 Service Agreement Management Interface Classes

7.3.2.1 Interface Class IpAppServiceAgreementManagement

Inherits from: IpInterface.



Method

signServiceAgreement()

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

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

Parameters

serviceToken : in TpServiceToken

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

agreementText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

Returns

TpOctetSet

Raises

TpCommonExceptions, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM

Method

terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpOctetSet

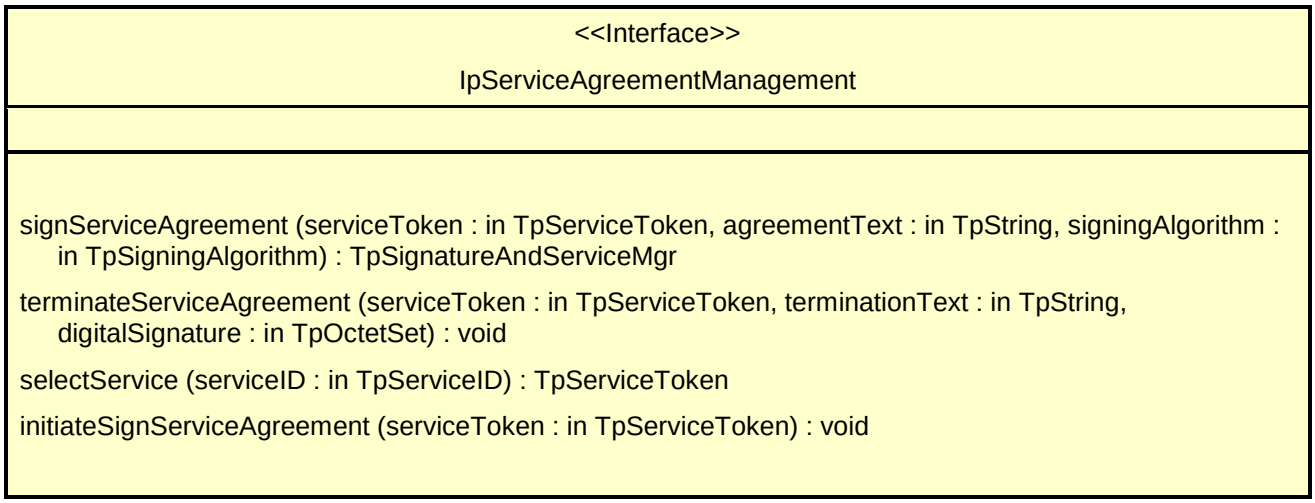
This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client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

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

7.3.2.2 Interface Class IpServiceAgreementManagement

Inherits from: IpInterface.



Method

signServiceAgreement()

After the framework has called signServiceAgreement() on the application's IpAppServiceAgreementManagement interface, ~~T~~this method is used by the client application to request that the framework sign ~~an~~ the service 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~~A reference to the service manager interface of the service is returned to the client application. The service manager returned will be configured as per the service level agreement. If the framework uses service subscription, the service level agreement will be encapsulated in the subscription properties contained in the contract/profile for the client application, which will be a restriction of the registered properties. If the client application is not allowed to access the service, then an error code (P_SERVICE_ACCESS_DENIED) is returned.

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

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

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

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

Parameters

serviceToken : in TpServiceToken

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

agreementText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

Returns

TpSignatureAndServiceMgr

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AGREEMENT_TEXT,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED

Method

terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpOctetSet

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

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

Method

selectService()

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

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

Parameters

serviceID : in TpServiceID

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

Returns

TpServiceToken

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_SERVICE_ACCESS_DENIED

Method

initiateSignServiceAgreement()

This method is used by the client application to initiate the sign service agreement process. This method shall be invoked following the application's call to selectService(), and before the signing of the service agreement can take place. 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

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **052** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Correct use of electronic signatures
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change: ⌘

This CR proposes an overhaul of the digital signature usage in OSA. The changes in this CR are based on those proposed in contributions N5-020283.

Digital signatures are used in OSA for the signing of service agreements. They are also used for the termination of service agreements, and for the Framework's termination of the client's access session. But they are not used for other methods which result in termination of service agreements: those invoked by a client which terminate a client's access session with the Framework. This is a potential security hole, offering a means to perform denial of service attacks.

There is no negotiation mechanism in the API to enable negotiation of the signing algorithms, yet negotiation is used for such things as encryption capabiltities in Authentication.

The choice of signing algorithms is resticted and should be extended with newer choices.

IpClientAccess.terminateAccess() has had correct digital Signature added, including replay protection. Also, functionality extended to close also all service instances associated with access session.

TpSigningAlgorithm extended with state of the art signing algorithms.

IpAccess.endAccess replaced with terminateAccess: to add digital signature for security, to prevent denial of service attacks on this unprotected method. Also to remove the endAccessProperties which were undefined, but without which the method would throw an exception. This removes possiblity to leave service instances open following close of Framework access session, which was a further security hole.

IpAccess.releaseInterface() replaced with relinquishInterface, to add digital signature parameters for security, to prevent denial of service attacks on this unprotected method.

Summary of change:	⌘	Add a negotiation mechanism for Signing Algorighms: selectSigningAlgorithm added to provide negotiation for algorithm to be used for ALL digital signatures (even those in signServiceAgreement). TpSigningAlgorithm extended with state of the art signing algorithms. IpClientAccess.terminateAccess() has had correct digital Signature added, including replay protection (timestamp). Also, functionality extended to close also all service instances associated with access session. IpAccess.endAccess replaced with terminateAccess: to add digital signature for security, to prevent denial of service attacks on this unprotected method. Also to remove the endAccessProperties which were undefined, but without which the method would throw an exception. This removes possiblity to leave service instances open following close of Framework access session, which was a further security hole. IpAccess.releaseInterface() replaced with relinquishInterface, to add digital signature parameters for security, to prevent denial of service attacks on this unprotected method. Methods signServiceAgreement() and terminateServiceAgreement clarified to use the signing algorithm negotiated earlier, and to include replay protection using timestamping.
Consequences if not approved:	⌘	Security weaknesses will remain in the OSA specifications, which will limit their adoption and use. Lack of clear instructions for implementors will lead to interoperability difficulties.

Clauses affected:	⌘	6.3, 7.3, 10.3, 11		
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

How to create CRs using this form:

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

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

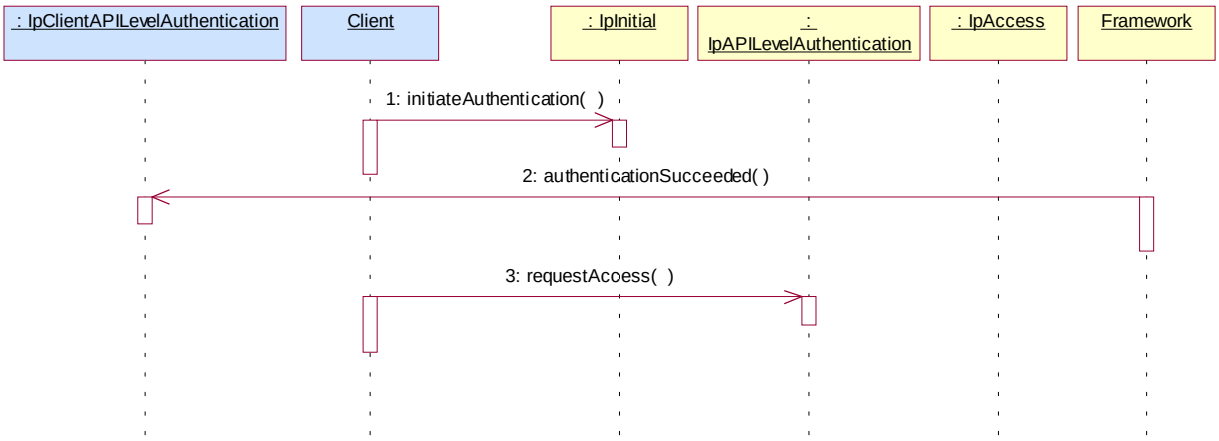
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest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

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

6.1.1 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.

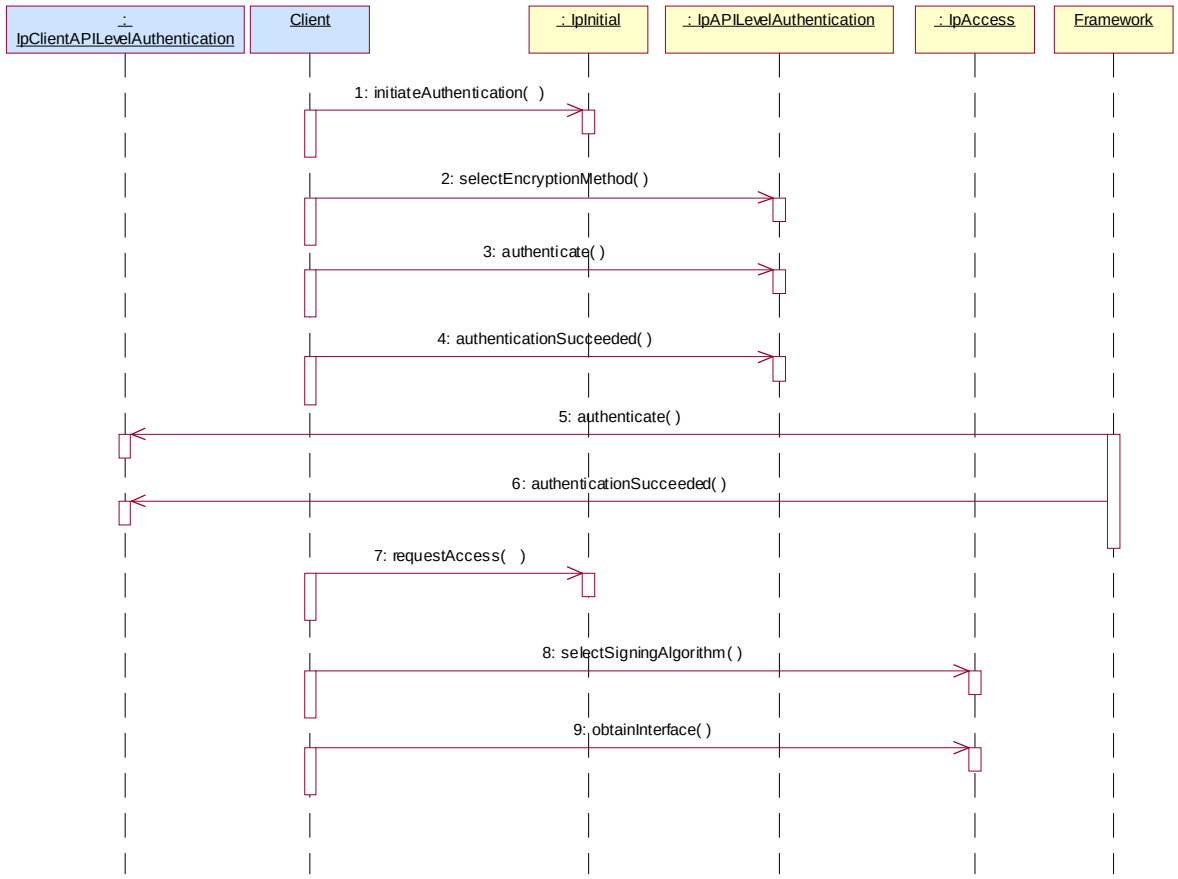
3: The Client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

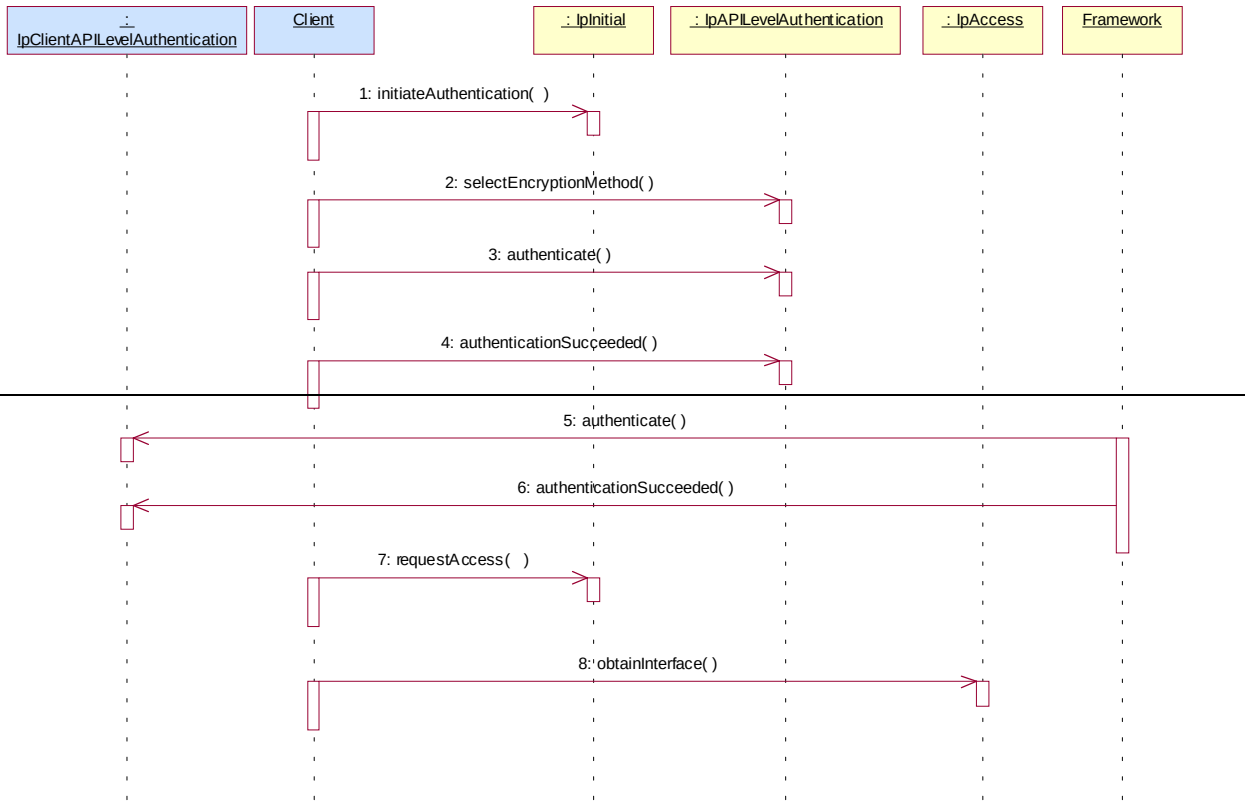
6.1.1.2 Initial Access

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.


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


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

1: Initiate Authentication

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

2: Select Encryption Method

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

3: Authenticate

4: The client provides an indication if authentication succeeded.

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

6: The Framework provides an indication if authentication succeeded.

7: Request Access

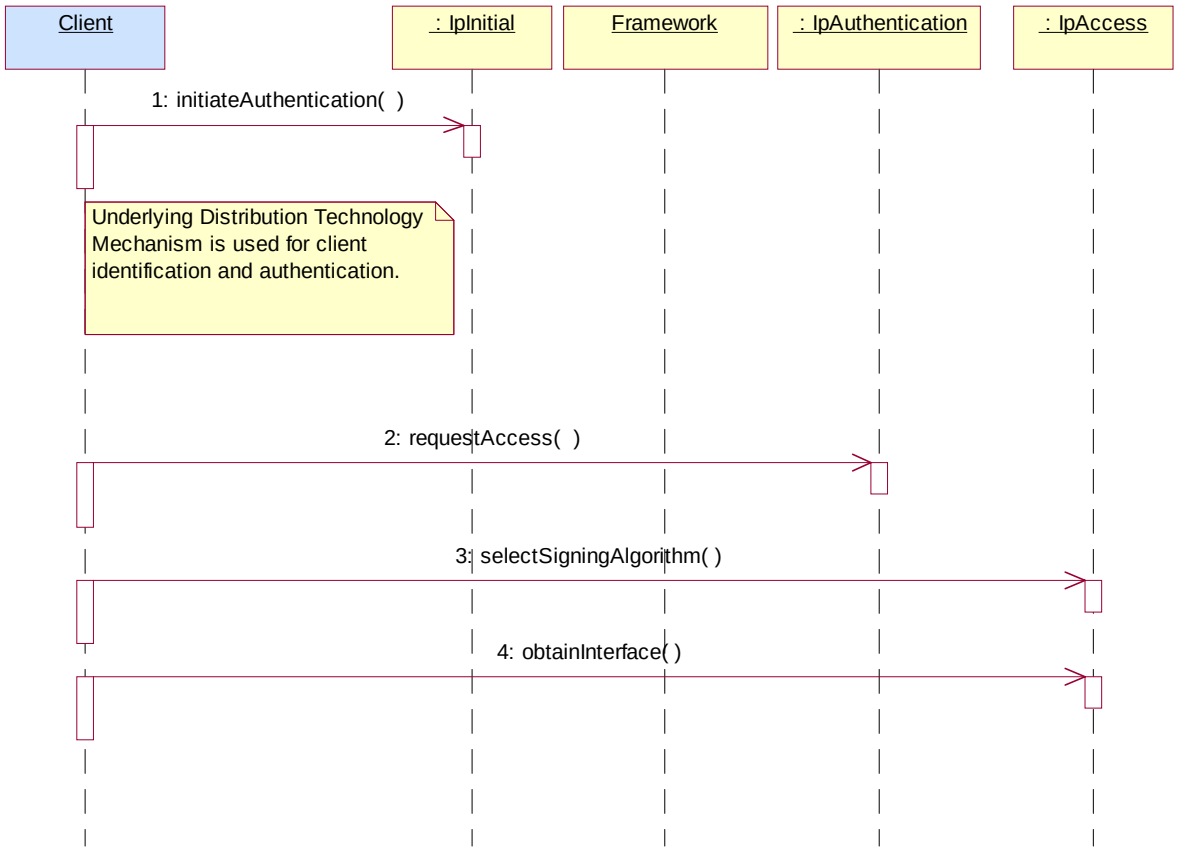
Upon successful (mutual) authentication, the client invokes requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface.

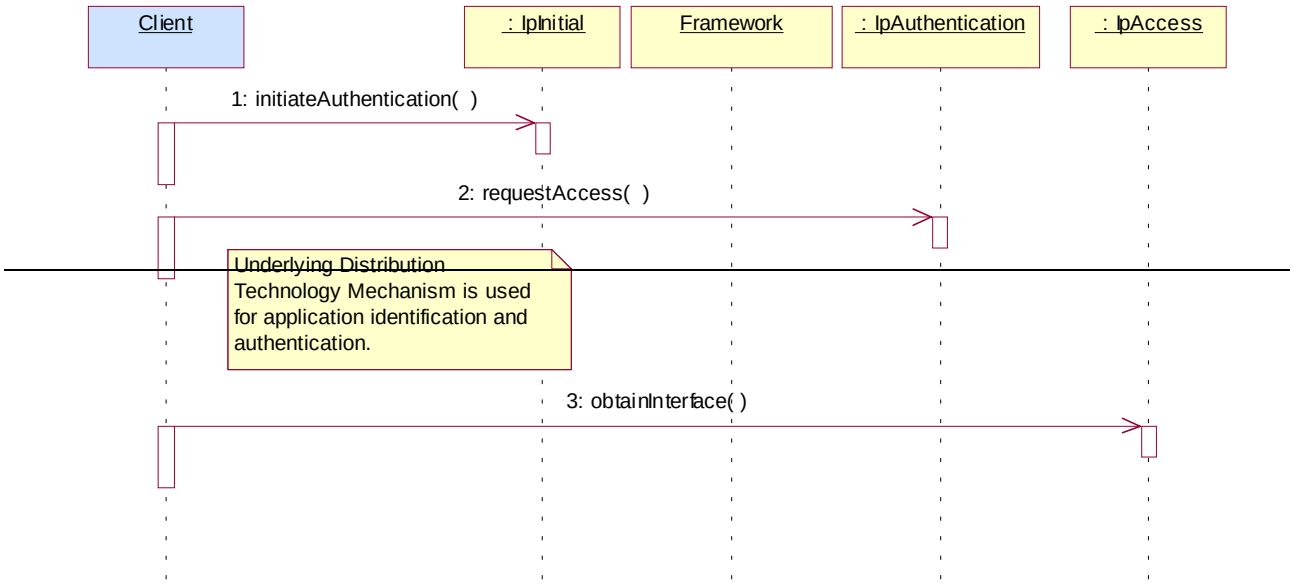
8: The client and framework negotiate the signing algorithm to be used for any signed exchanges.

~~8~~9: 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 and the framework can negotiate, on the framework's Access interface, the signing algorithm to be used for any signed exchanges.

34: ~~If the authentication was successful, t~~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.

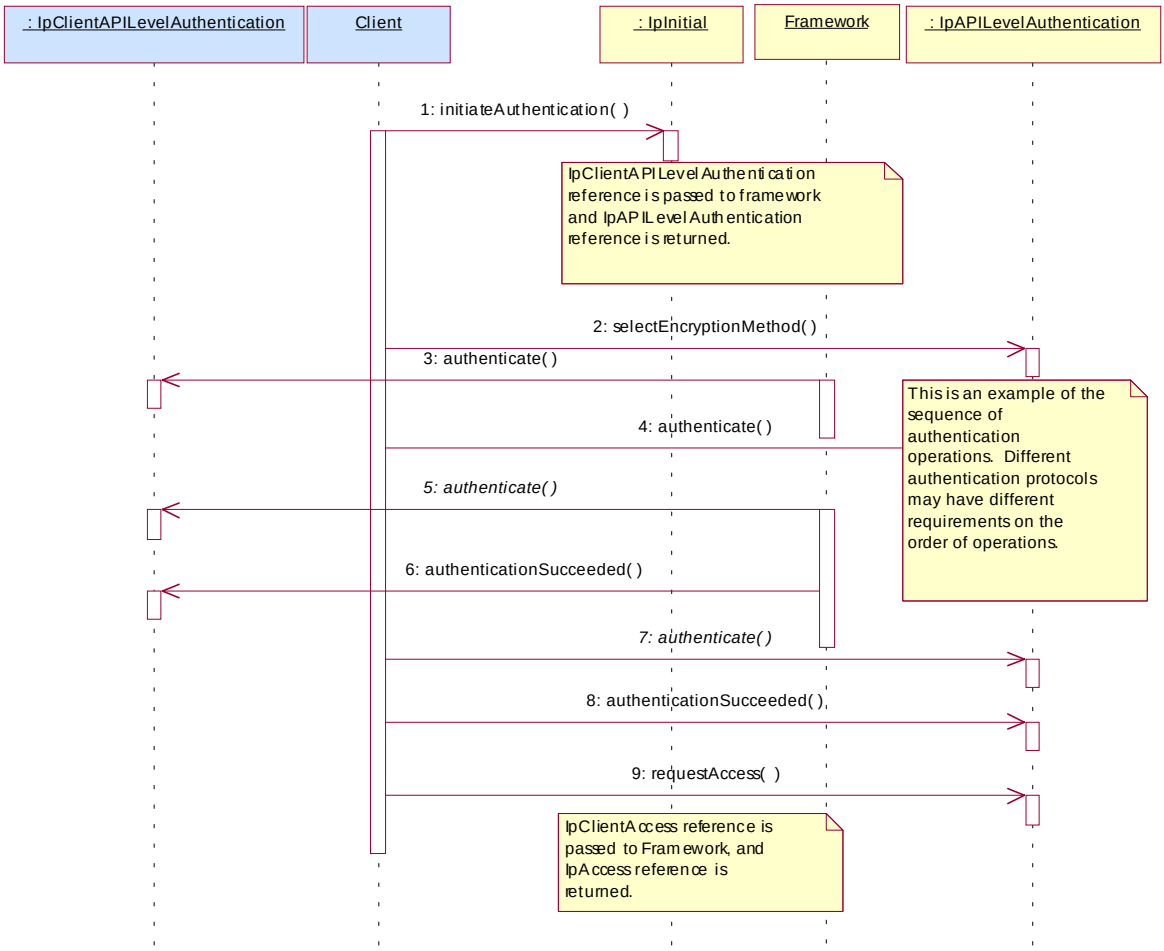
The client must authenticate with the Framework before it is able to use any of the other interfaces supported by the Framework. Invocations on other interfaces will fail until authentication has been successfully completed.

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.

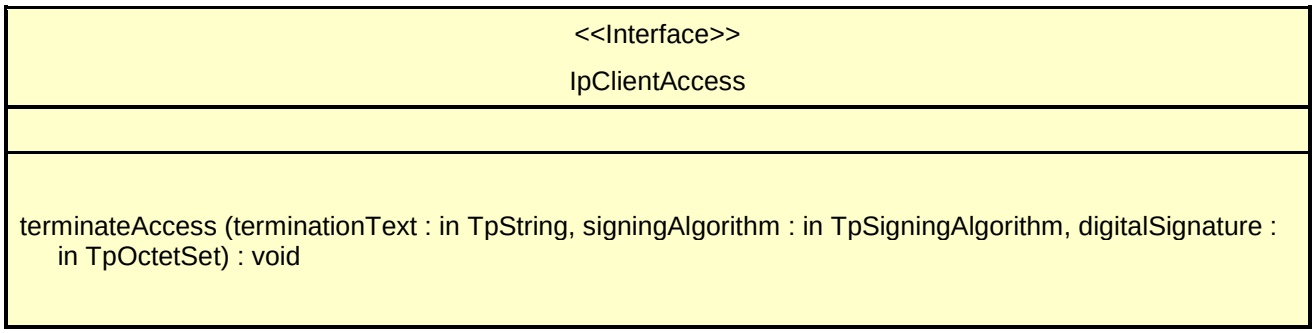
Note that at any point during the access session, either side can request re-authentication. Re-authentication does not have to be mutual.


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
1: initiateAuthentication( )
IpClientAPILevelAuthentication reference is passed to framework and IpAPILevelAuthentication reference is returned.
2: selectEncryptionMethod( )
3: authenticate( )
4: authenticate( )
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
5: authenticate( )
6: authenticationSucceeded( )
7: authenticate( )
8: authenticationSucceeded( )
9: requestAccess( )
IpClientAccess reference is passed to Framework, and IpAccess reference is returned.

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.



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. Also, all remaining service instances created by the framework either directly in this access session or on behalf of the client during this access session shall be terminated. 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. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the client, the P_INVALID_SIGNING_ALGORITHM exception will be thrown. The list of possible algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

digitalSignature : in TpOctetSet

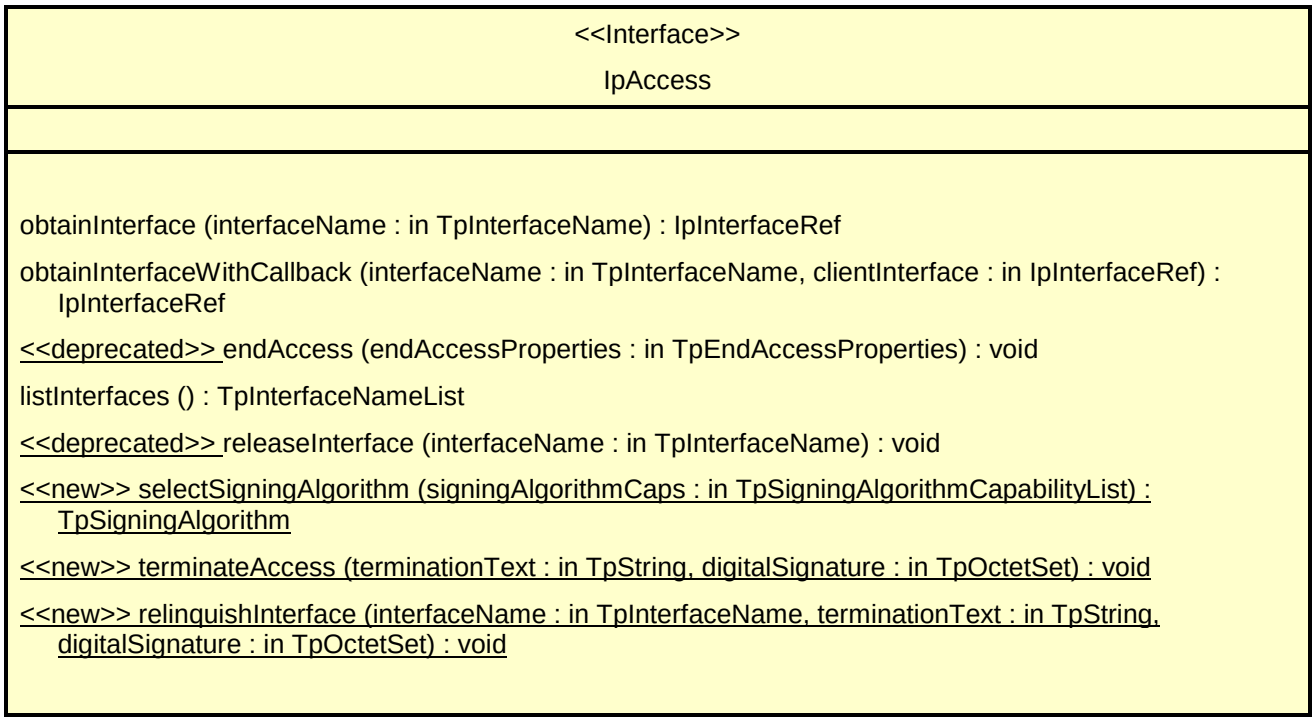
This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~This is a signed version of a hash of the termination text.~~ The framework uses this to confirm its identity to the client. The client can check that the terminationText has been signed by the framework. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNING_ALGORITHM, P_INVALID_SIGNATURE

6.3.1.6 Interface Class IpAccess

Inherits from: IpInterface.



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

<<deprecated>> 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
<<deprecated>> 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

Method
<<new>> selectSigningAlgorithm()

The client uses this method to inform the Framework of the different signing algorithms it supports for use in all cases where digital signatures are required. The Framework will select one of the suggested algorithms. This method shall be the first method invoked by the client on IpAccess. The algorithm chosen as a result of the response to this method remains valid for an instance of IpAccess and until this method is re-invoked by the client. If an algorithm that is acceptable to the framework within the capability of the client cannot be found, the framework throws the P_NO_ACCEPTABLE_SIGNING_ALGORITHM exception.

Returns: selectedAlgorithm. This is the signing algorithm chosen by the Framework. The chosen algorithm shall be taken from the list proposed by the Client.

Parameters

signingAlgorithmCaps : in TpSigningAlgorithmCapabilityList

The list of signing algorithms supported by the client.

Returns

TpSigningAlgorithm

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_NO_ACCEPTABLE_SIGNING_ALGORITHM

Method

<<new>> terminateAccess()

The terminateAccess method 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. Also, all remaining service instances created by the framework either directly in this access session or on behalf of the client during this access session shall be terminated.

Parameters

terminationText : in TpString

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

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The client uses this to confirm its identity to the framework. The framework can check that the terminationText has been signed by the client. If a match is made, the access session is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SIGNATURE

Method

<<new>> relinquishInterface()

The client uses this method to release an instance of 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.

terminationText : in TpString

This is the termination text describes the reason for the release of the interface. This text is required simply because the digitalSignature parameter requires a terminationText to sign.

digitalSignature : in TpOctetSet

This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. The client uses this to confirm its identity to the framework. The framework can check that the terminationText has been signed by the client. If a match is made, the interface is released, otherwise the P_INVALID_SIGNATURE exception will be thrown.

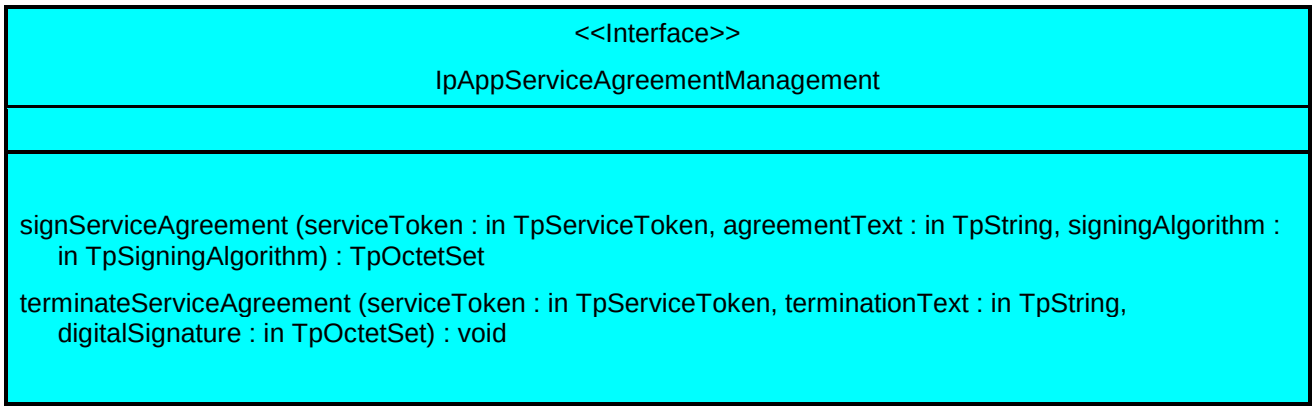
Raises

TpCommonExceptions, P_INVALID_SIGNATURE, P_INVALID_INTERFACE_NAME

7.3.3 Service Agreement Management Interface Classes

7.3.3.1 Interface Class IpAppServiceAgreementManagement

Inherits from: IpInterface.



7.3.3.1.1 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> : This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and agreement text given by the framework. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and agreement text). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~The digitalSignature is the signed version of a hash of the service token and agreement text given by the framework.~~ If the signature is incorrect the serviceToken will be expired immediately.

Parameters

`serviceToken : in TpServiceToken`

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

`agreementText : in TpString`

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

`signingAlgorithm : in TpSigningAlgorithm`

This is the algorithm used to compute the digital signature. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the client application, the P_INVALID_SIGNING_ALGORITHM exception is thrown. The list of possible

algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

Returns

TpOctetSet

Raises

TpCommonExceptions, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM

7.3.3.1.2 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

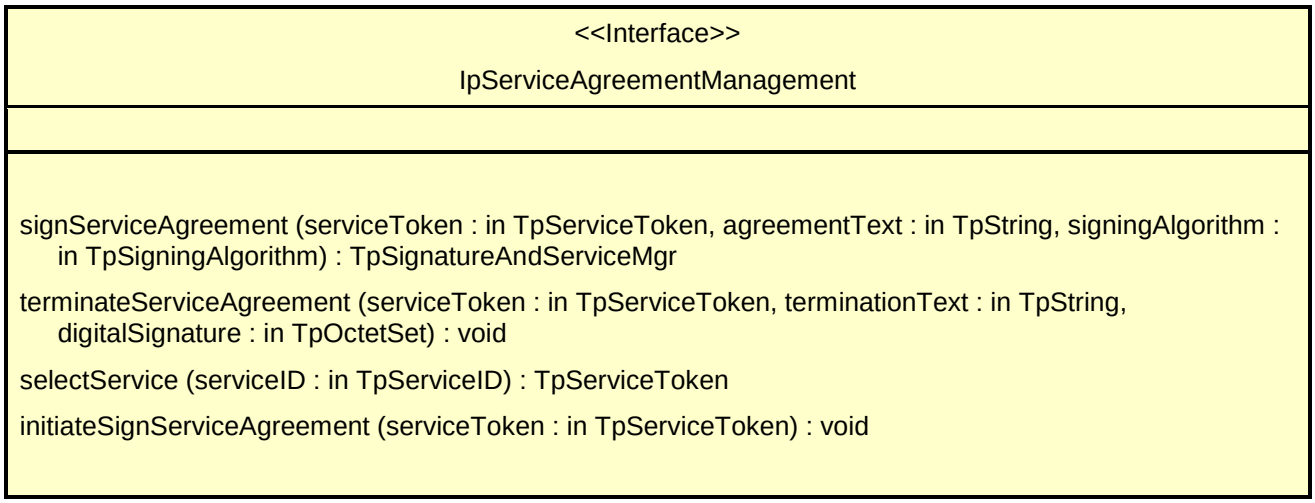
This contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~This is a signed version of a hash of the service token and the termination text.~~ The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The client application can check that the terminationText has been signed by the framework. If a match is made, the service agreement is terminated, otherwise the P_INVALID_SIGNATURE exception will be thrown.

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

7.3.3.2 Interface Class IpServiceAgreementManagement

Inherits from: IpInterface.



7.3.3.2.1 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 contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and agreement text given by the client application. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and agreement text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~is the signed version of a hash of the service token and agreement text given by the client application.~~

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

Parameters

`serviceToken : in TpServiceToken`

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

`agreementText : in TpString`

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

`signingAlgorithm : in TpSigningAlgorithm`

This is the algorithm used to compute the digital signature. It shall be identical to the one chosen by the framework in response to IpAccess.selectSigningAlgorithm(). If the signingAlgorithm is not the chosen one, is invalid, or unknown to the framework, an error code (P_INVALID_SIGNING_ALGORITHM) is returned. The list of possible algorithms is as specified in the TpSigningAlgorithm table. The identifier used in this parameter must correspond to the digestAlgorithm and signatureAlgorithm fields in the SignerInfo field in the digitalSignature (see below).

Returns

TpSignatureAndServiceMgr

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AGREEMENT_TEXT,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED

7.3.3.2.2 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 contains a CMS (Cryptographic Message Syntax) object (as defined in [RFC 2630]) with content type Signed-data. The signature is calculated and created as per section 5 of RFC 2630. The content is made of the service token and the termination text. The "external signature" construct shall not be used (ie the eContent field in the EncapsulatedContentInfo field shall be present and contain the service token and the termination text string). The signing-time attribute, as defined in section 11.3 of RFC 2630, shall also be used to provide replay prevention. ~~This is a signed version of a hash of the service token and the termination text.~~ The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement().The framework uses this to check that the terminationText has been signed by the client application. If a match is made, the service agreement is terminated, otherwise an error code (P_INVALID_SIGNATURE) is returned.

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

7.3.3.2.3 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

7.3.3.2.4 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

10.3.11 TpSigningAlgorithm

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

String Value	Description
NULL	An empty (NULL) string indicates no signing algorithm is required
P_MD5_RSA_512	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit modulus. The signature generation follows the process and format defined in RFC 2313 (PKCS#1 v1.5). The use of this signing method is deprecated.~~MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public-key cryptography system using a 512-bit key.~~
P_MD5_RSA_1024	MD5 takes an input message of arbitrary length and produces as output a 128-bit message digest of the input. This is then encrypted with the private key under the RSA public- key cryptography system using a 1024-bit modulus. .The signature generation follows the process and format defined in RFC 2313 (PKCS#1 v1.5). The use of this signing method is deprecated.~~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~~
~~P_RSASSA-PKCS1-v1_5_SHA1_1024~~P_RSASSA_PKCS1_v1_5_SHA1_1024	SHA-1 is used to produce a 160-bit message digest based on the input message to be signed. RSA is then used to generate the signature value, following the process defined in section 8 of RFC 2437 and format defined in section 9.2.1 of RFC 2437. The RSA private/public key pair is using a 1024-bit modulus.
P_SHA1_DSA	SHA-1 is used to produce a 160-bit message digest based on the input message to be signed. DSA is then used to generate the signature value. The signature generation follows the process and format defined in section 7.2.2 of RFC 2459.

10.3.12 TpSigningAlgorithmList

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

11 Exception Classes

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

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_DUPLICATE_PROPERTY_NAME	A 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	No encryption mechanism, which is acceptable to the framework, is supported by the client
P_NO_ACCEPTABLE_SIGNING_ALGORITHM	No signing algorithm, which is acceptable to the framework, is 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

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020705**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **053** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Addition of Sequence Diagrams for terminateAccess
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	There are no sequence diatrams which illustrate the use of and consequences of using the terminateAccess methods.
Summary of change:	⌘	Add two new sequence diagrams to the Framework Access section to illustrate the use of terminateAccess.
Consequences if not approved:	⌘	Application developers may not realise the consequences of using these methods.

Clauses affected:	⌘			
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘	6.1.1		

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

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

6.1.1.5 Framework Terminates Access

This sequence shows how a Framework could terminate an application's use of the Framework and of all service instances. This type of termination is unusual, but possible with the terminateAccess method. Note that 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.

1: Following successful authentication and service discovery, the client initiates the service agreement signing process (not shown). This is completed when the client invokes signServiceAgreement on the Framework's IpServiceAgreementManagement interface, and a reference to an instance of a service manager interface is returned.

2: The client (application) had initiated service agreement signing process for a second service agreement (not shown), and when the client signs this second service agreement, a reference to an instance of another service manager, for another service type, is returned.

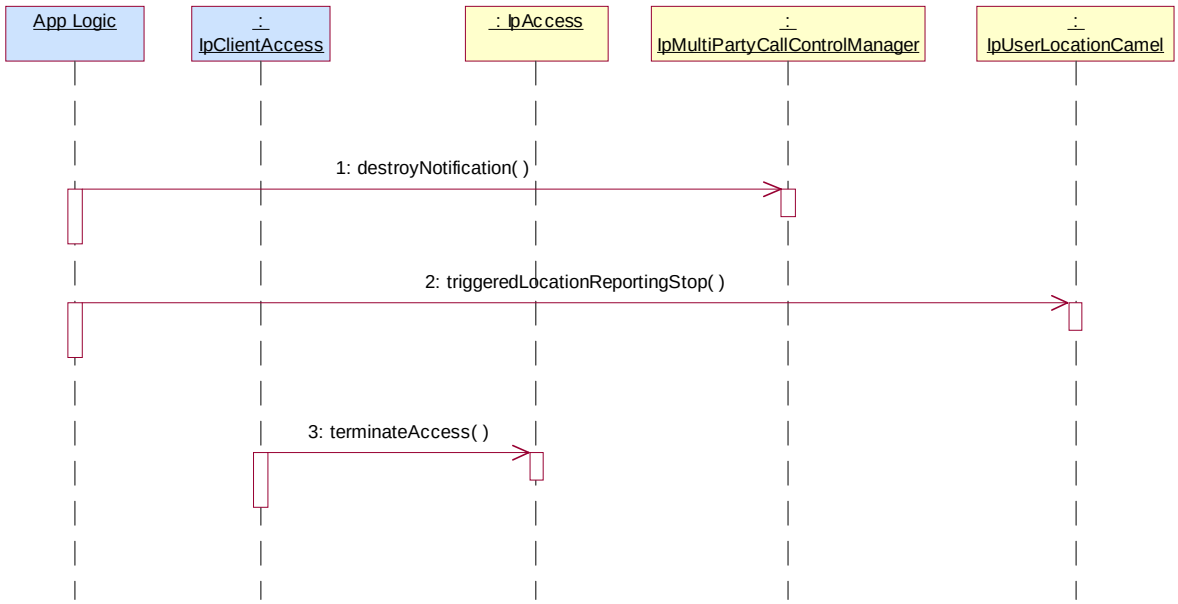
3: The application starts to use the new service manager interface.

4: The application starts to use the other new service manager interface.

5: The framework decides to terminate the application's access session, and to terminate all its service agreements. This is an unusual and drastic step, but could be e.g. due to violation or expiry of the application's service agreements, or some problem within the framework itself. The framework will also destroy each of the service managers the application was using (not shown). The application is now no longer authenticated with the framework, and all Framework and service interfaces it was using are destroyed.

6.1.1.6 Application Terminates Access

This sequence shows how an application could terminate its use of the Framework and of all service instances. This type of termination is unusual, but possible with the terminateAccess method.



1: The application terminates its use of the service manager instances in a controlled manner.

3: The application decides to terminate its access session and all its service agreements in one go. The framework will also destroy each of the service managers the application was using (not shown). The application is now no longer authenticated with the framework, and all Framework and service interfaces it was using are destroyed. The application could have terminated its service agreeements one by one, by invoking terminateServiceAgreement on the Framework's IpServiceAgreementManager interface, and then invoked terminateAccess on the Framework's IpAccess interface, which would have been a more controlled shutdown.

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **054** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Add indication what part of service agreement must be signed
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change: ⌘ The IpAppServiceAgreementManagement and IpServiceAgreementManagement interfaces contain the method signServiceAgreement(). This method is used for signing a service level agreement. One of the return values is the digital signature.

The current description states:

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

The specification does not clearly specify how the hash shall be calculated based on service token and agreement text. Several ways are possible to clarify this in the specification.

Summary of change: ⌘ It is indicated that only the agreement text shall be signed.

Consequences if not approved: ⌘ Interoperability problems.

Clauses affected: ⌘

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

Other comments: ⌘

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

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

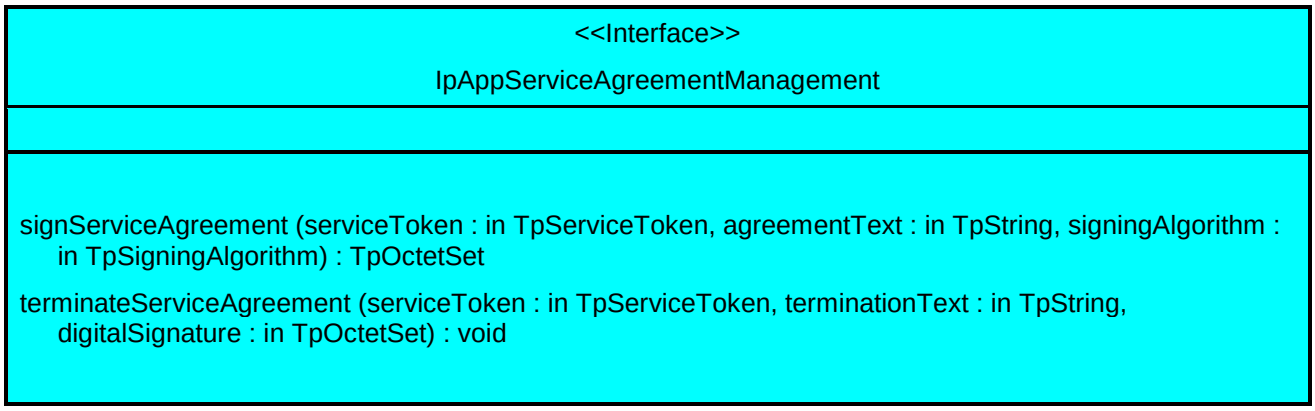
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest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

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

Proposed Changes

7.3.2.1 Interface Class IpAppServiceAgreementManagement

Inherits from: IpInterface.



Method

signServiceAgreement()

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

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

Parameters

serviceToken : in TpServiceToken

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

agreementText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

Returns

TpOctetSet

Raises

TpCommonExceptions, P_INVALID_AGREEMENT_TEXT, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNING_ALGORITHM

Method

terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpOctetSet

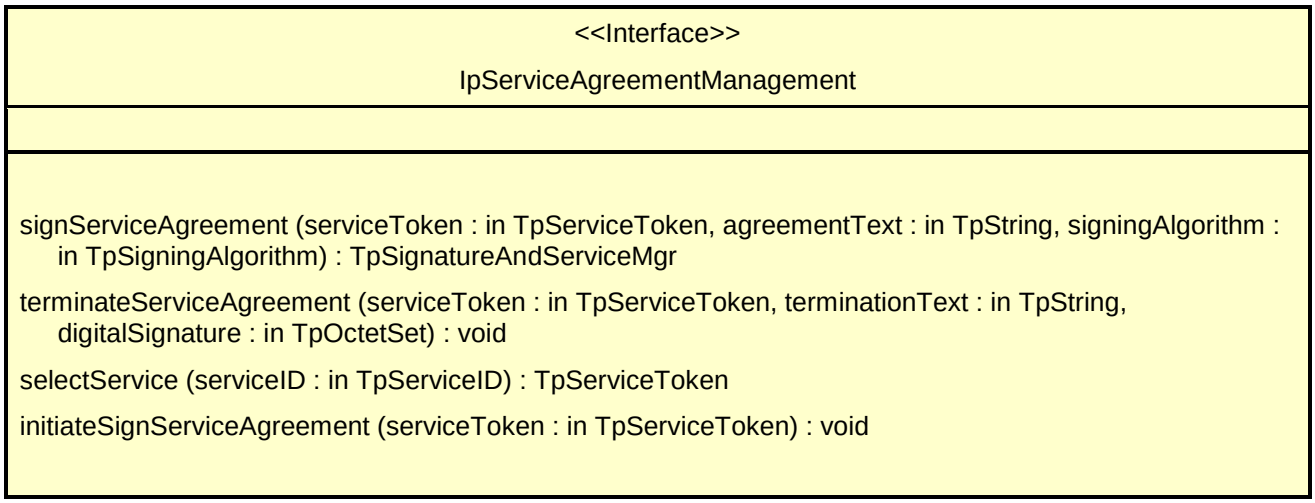
This is a signed version of a hash of ~~the service token and~~ the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using signServiceAgreement(). The framework uses this to confirm its identity to the client application. The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

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

7.3.2.2 Interface Class IpServiceAgreementManagement

Inherits from: IpInterface.



Method

signServiceAgreement()

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

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

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

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

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

Parameters

serviceToken : in TpServiceToken

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

agreementText : in TpString

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

signingAlgorithm : in TpSigningAlgorithm

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

Returns

TpSignatureAndServiceMgr

Raises

TpCommonExceptions,P_ACCESS_DENIED,P_INVALID_AGREEMENT_TEXT,P_INVALID_SERVICE_TOKEN,P_INVALID_SIGNING_ALGORITHM,P_SERVICE_ACCESS_DENIED

Method

terminateServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

terminationText : in TpString

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

digitalSignature : in TpOctetSet

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

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_TOKEN, P_INVALID_SIGNATURE

Method

selectService()

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

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

Parameters

serviceID : in TpServiceID

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

Returns

TpServiceToken

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_INVALID_SERVICE_ID, P_SERVICE_ACCESS_DENIED

Method

initiateSignServiceAgreement()

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

Parameters

serviceToken : in TpServiceToken

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

Raises

TpCommonExceptions, P_INVALID_SERVICE_TOKEN, P_SERVICE_ACCESS_DENIED

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020716**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **055** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Add text to clarify requirements on support of methods
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	It is not clear in the OSA Specifications what exactly is meant by support of a method: is it sufficient to include such code as to respond correctly to a method invocation with the exception P_METHOD_NOT_SUPPORTED, or is it required to support the functionality described and defined by the method?
Summary of change:	⌘	Add text to clause 4 to indicate that support or implementation of a method requires that the functionality of the method be supported or implemented.
Consequences if not approved:	⌘	Different vendors and application developers will each build equipment and applications which they claim to be conformant, but which will never interwork.

Clauses affected:	⌘	4		
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

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

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

4 Overview of the Framework

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

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

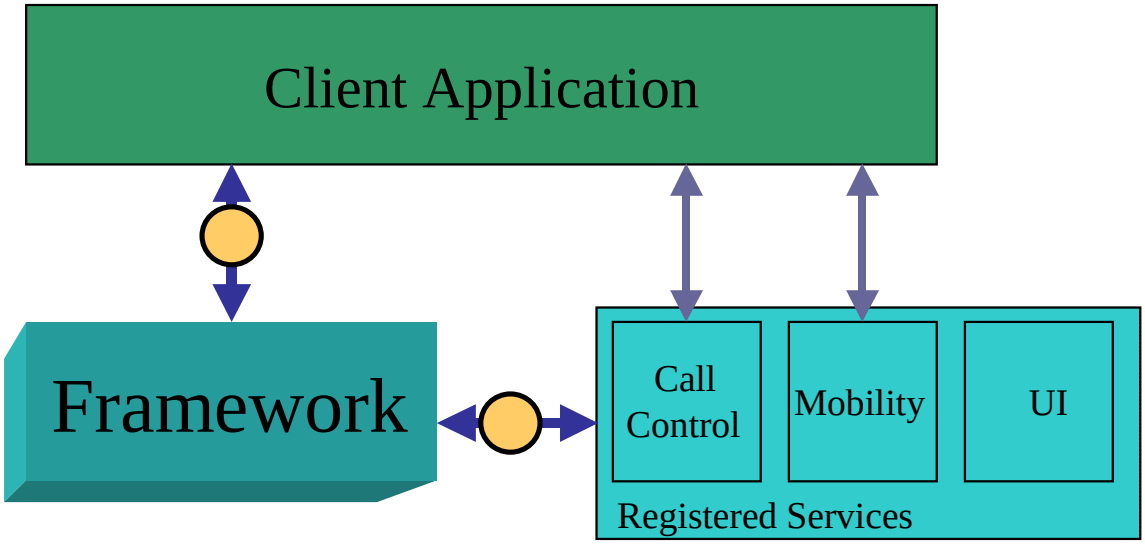


Figure:

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

Basic mechanisms between Application and Framework:

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

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

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

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

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

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

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

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

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

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.

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **056** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Introduce types and modes for generic properties
Source:	⌘	CN5
Work item code:	⌘	OSA2 ***Date:*** ⌘ 12/07/2002
Category:	⌘	**F** ***Release:*** ⌘ *REL-5*

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

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

Reason for change:	⌘	The specification is incomplete.
Summary of change:	⌘	The type and mode of the generic properties is specified.
Consequences if not approved:	⌘	Interoperability problems.

Clauses affected:	⌘	10.2
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

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

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

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

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

Introduction

Currently in the Framework specification the generic service properties are defined. It is however not defined of what type and mode they are. Not even the names of these properties are properly defined.

In this contribution the names, values and modes of the generic properties are defined.

Most general service properties are MANDATORY_READONLY properties. I.e., the service has to provide a value for the properties at registration and they cannot be changed subsequently. ~~One e~~Exceptions ~~is~~ are the Servive Name, the Service Version, and the Service Instance ID property, which is generated by the framework and can therefore not be provided by the service during registration. ~~This~~ The mode of ~~this~~ these ~~property~~ properties is READONLY.
The other exceptions are the “Operation Set” and “service version” properties. The service has to provide values for these at registration. However, this can later be restricted by the service level agreement. The mode of these properties is MANDATORY.

Additionally, the names and descriptions of the properties “Serice Instance ID” and “Service Instance Description” seem to indicate a specific service instance, while these actually refer to the service as a whole and not only to a specific instance. This was originally indicated in N5-020435.

As a final change, the property “Supported Interfaces” is ~~removed~~deprecated. It does not give any any extra information with respect to the “P_OPERATION_SET” property, since the interface is part of the method names in the operation set. This will only lead to potential inconsistencies, e.g., interface is indicated as not supported, but some methods on the interface are specified in the operation set.

Since there was no proper definition yet, there should be no backward compatibility issues.

Proposed Changes

The following changes are proposed to 29.198-03:

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	set of address ranges	{"123??*", "*.ericsson.se"}	The set of address ranges consisting of ranges 123??* and *.ericsson.se.
INTEGER_INTERVAL	interval of integers	{"5", "100"}	The integers that are between or equal to 5 and 100.
STRING_INTERVAL	interval of strings	{"Rijen", "Sophia"}	The strings that are between or equal to the strings "Rijen" and "Sophia", in lexicographical order.
INTEGER_INTEGER_MAP	map from integers to integers	{"1", "10", "2", "20", "3", "30"}	The map that maps 1 to 10, 2 to 20 and 3 to 30.

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

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

The following sections describe these general service properties in more detail. The values for the mode are defined in the type TpServiceTypePropertyMode.

10.2.1 Service Name

Property	Type	Mode	Description
P_SERVICE_NAME	STRING_SET	MANDATORY_ READONLY	This property contains the name of the service, e.g. “UserLocation”, “UserLocationCamel”, “UserLocationEmergency” or “UserStatus”.

10.2.2 Service Version

Property	Type	Mode	Description
P_SERVICE_VERSION	STRING_SET	MANDATORY	This property contains the version of the APIs, to which the service is ~~compliant, e.g.~~

			~~"2.1"~~.It is a set of strings as specified in the TpVersion type.

10.2.3 Service ~~Instance~~ ID

Property	Type	Mode	Description
P_SERVICE_ID	STRING_INTERVAL	READONLY	This property uniquely identifies a specific ~~instance of the~~ service. Note that ~~The~~ the Framework generates this property ~~.~~ value when the Service Supplier registers the service. This property should not be confused with the serviceInstanceID generated by the Framework when a Client Application signs a Service Agreement to obtain the Service Manager

10.2.4 Service ~~Instance~~ Description

Property	Type	Mode	Description
P_SERVICE_DESCRIPTION	STRING_SET	MANDATORY READONLY	This property contains a textual description of the service. It should not be interpreted as a description of a Service Instance (as identified by a serviceInstanceID generated by the Framework when a Client Application signs a Service Agreement to obtain the Service Manager).

10.2.5 Product Name

Property	Type	Mode	Description
P_PRODUCT_NAME	STRING_SET	~~MANDATORY~~ READONLY	This property contains the name of the product that provides the service, e.g. “Find It”, “Locate.com”.

10.2.6 Product Version

Property	Type	Mode	Description
P_PRODUCT_VERSION	STRING_SET	~~MANDATORY~~ READONLY	This property contains the version of the product that provides the service, e.g. “3.1.11”.

10.2.7 (DEPRECATED) Supported Interfaces

	This property contains a list of strings with interface names that the service supports, e.g. “IpUserLocation”, “IpUserStatus”. This property is deprecated and will be removed in future versions of the specification.

10.2.8 Operation Set

Property	Type	Mode	Description
P_OPERATION_SET	STRING_SET	MANDATORY	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”}.

joint API group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-020751**
Meeting #19, Montreal, CANADA, 8 – 12 July 2002

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **057** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Correction on use of NULL in Framework API
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**A**
Release:	⌘	*REL-5*

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

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

Reason for change:	⌘	OMG IDL does not support NULL as a valid value for a data type; attempts to send a null value result in a marshalling exception and a gateway can never receive the call.
Summary of change:	⌘	Occurrences of the use of NULL as a valid Framework API data value have been replaced. An empty string value for serviceID is used to refer to the Framework in the Fault Management service, and an unspecified time interval is used to indicate that the time interval is at the discretion of the interface in question.
Consequences if not approved:	⌘	Failure to correct the API shall result in vendor specific interpretation and interoperability issues.

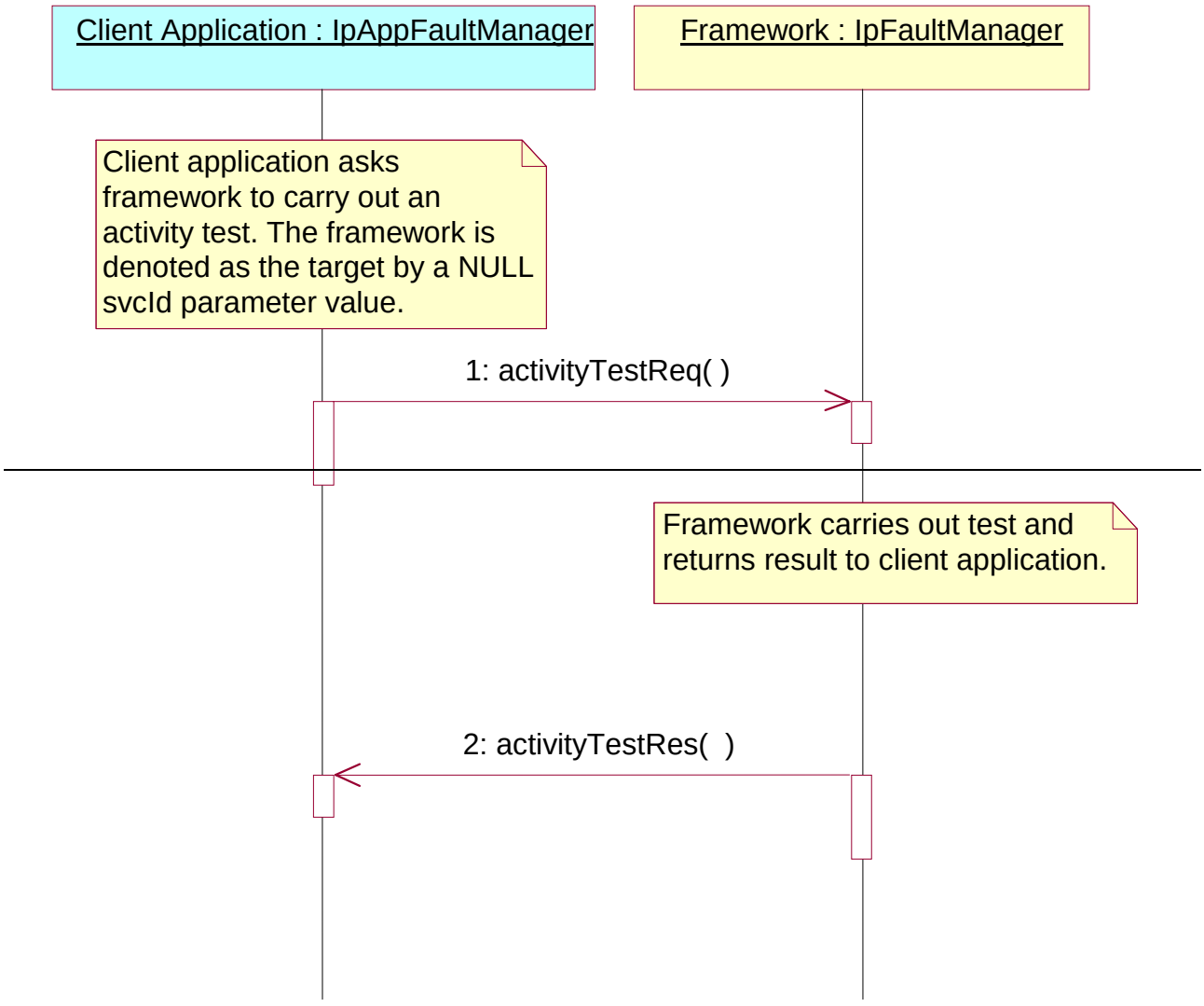
Clauses affected:	⌘	7.1.2.8; 7.3.3.1; 7.3.3.2; 7.4.3.4; 8.3.4.1; 8.3.4.2
Other specs affected:	⌘	☐ Other core specifications ⌘ ☐ Test specifications ☐ O&M Specifications
Other comments:	⌘	

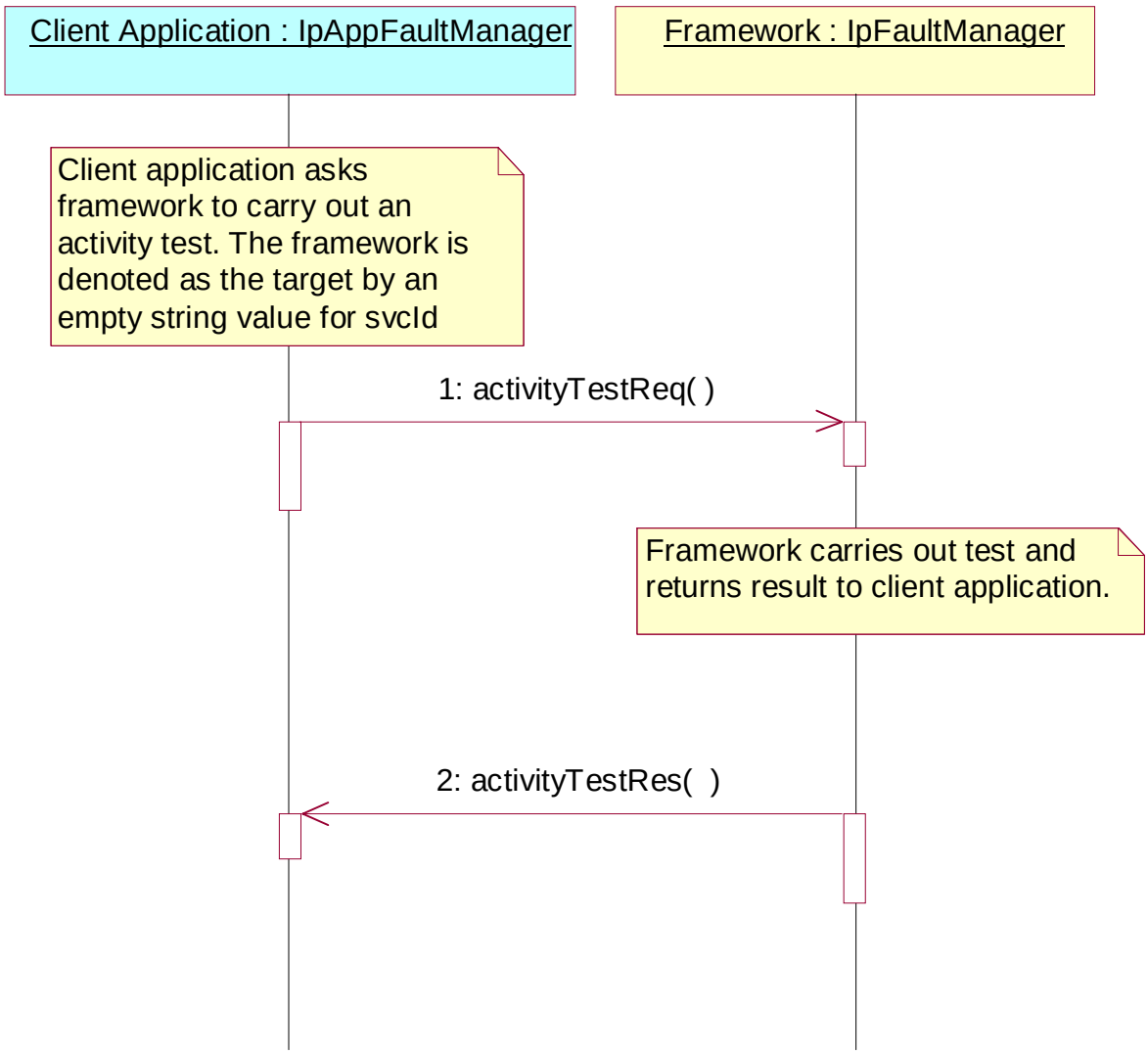
How to create CRs using this form:
Comprehensive information and tips about how to create CRs can be found at: http://www.3gpp.org/3G_Specs/CRs.htm. Below is a brief summary:

1) Fill out the above form. The symbols above marked ⌘ contain pop-up help information about the field that they are closest to.
2) Obtain the latest version for the release of the specification to which the change is proposed. Use the MS Word "revision marks" feature (also known as "track changes") when making the changes. All 3GPP specifications can be downloaded from the 3GPP server under ftp://ftp.3gpp.org/specs/ For the latest version, look for the directory name with the latest date e.g. 2001-03 contains the specifications resulting from the March 2001 TSG meetings.
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

********** START OF FIRST CHANGE **********

7.1.2.8 Fault Management: Application requests a Framework activity test





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

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

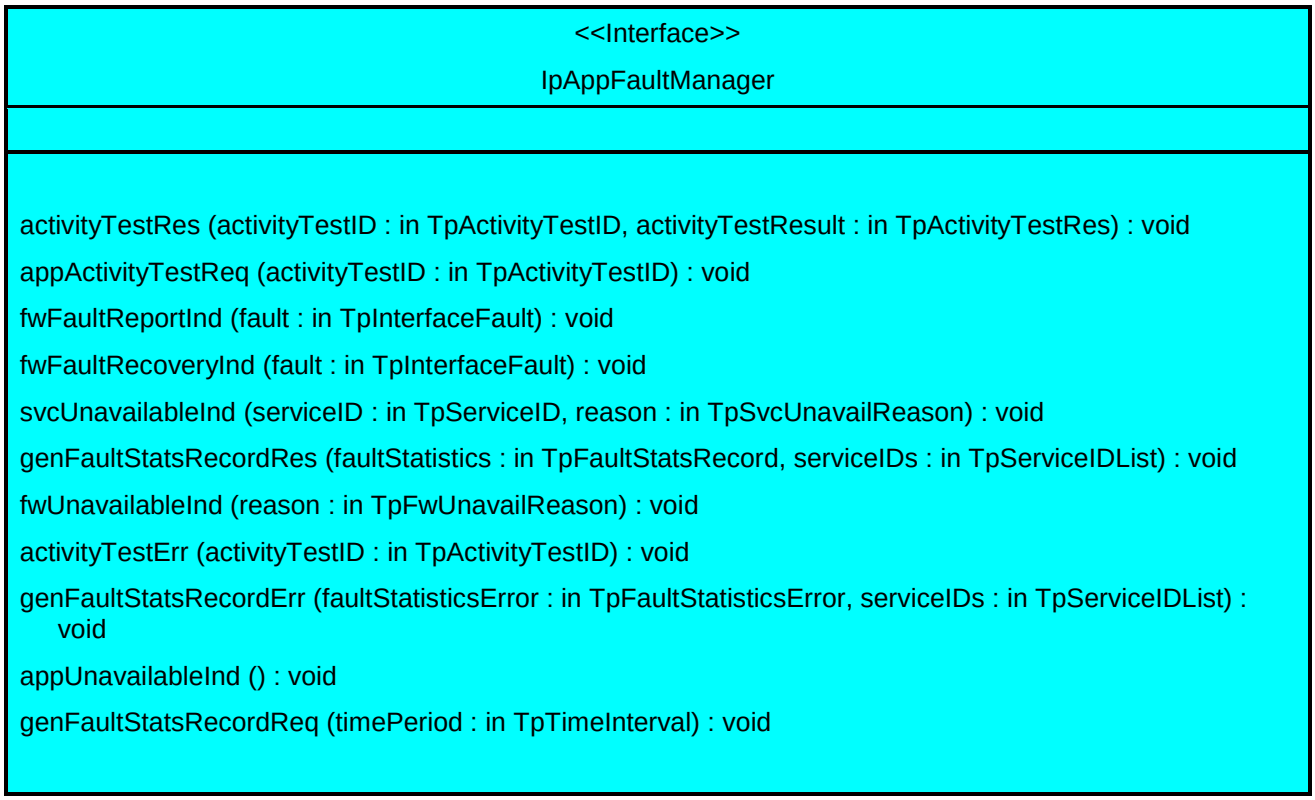
********** **END OF FIRST CHANGE** **********

********** **START OF SECOND CHANGE** **********

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
No Parameters were identified for this method

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. ~~A null~~Supplying both a start time and stop t ime as empty strings ~~value~~ leaves ~~this~~ the time period to the discretion of the client application.

********** END OF SECOND CHANGE **********

********** START OF THIRD CHANGE **********

7.3.3.2 Interface Class IpFaultManager

Inherits from: IpInterface.

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

<<Interface>> IpFaultManager
activityTestReq (activityTestID : in TpActivityTestID, svcID : in TpServiceID) : void appActivityTestRes (activityTestID : in TpActivityTestID, activityTestResult : in TpActivityTestRes) : void svcUnavailableInd (serviceID : in TpServiceID) : void genFaultStatsRecordReq (timePeriod : in TpTimeInterval, serviceIDs : in TpServiceIDList) : void appActivityTestErr (activityTestID : in TpActivityTestID) : void appUnavailableInd (serviceID : in TpServiceID) : void genFaultStatsRecordRes (faultStatistics : in TpFaultStatsRecord) : void genFaultStatsRecordErr (faultStatisticsError : in TpFaultStatisticsError) : void

Method

activityTestReq()

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

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

Parameters

activityTestID : in TpActivityTestID

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

svcID : in TpServiceID

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

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

appActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions,P_INVALID_SERVICE_ID,P_INVALID_ACTIVITY_TEST_ID

Method

svcUnavailableInd()

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

Parameters

serviceID : in TpServiceID

Identifies the service that the application can no longer use.

Raises

TpCommonExceptions ,P_INVALID_SERVICE_ID, P_UNAUTHORISED_PARAMETER_VALUE

Method

genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. ~~A null~~Supplying both a start time and stop time as empty strings ~~value~~ leaves ~~this~~ 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

********** END OF THIRD CHANGE **********

********** START OF FOURTH CHANGE **********

7.4.3.4 State Transition Diagrams for IpFaultManager

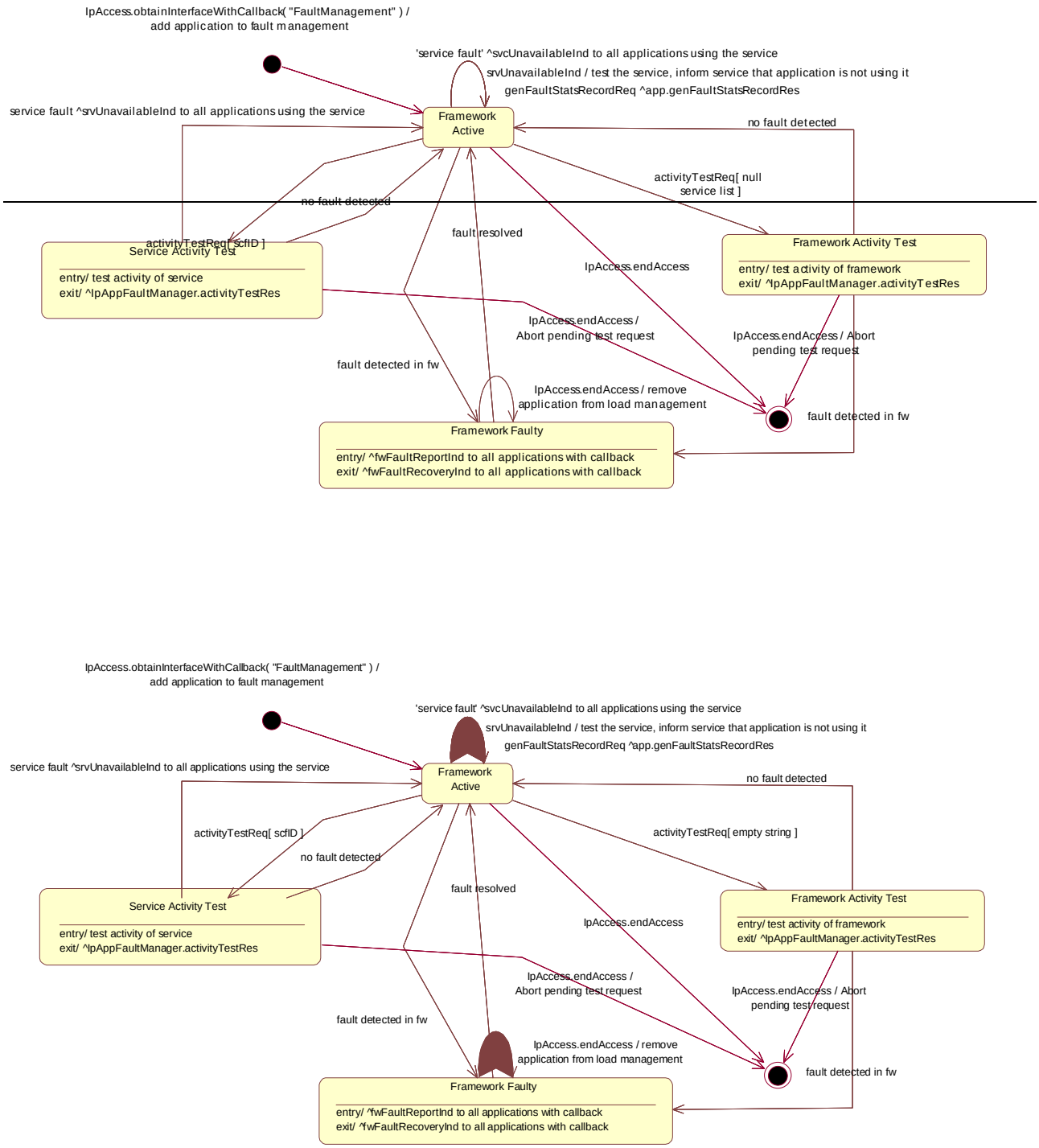


Figure : State Transition Diagram for IpFaultManager

********** END OF FOURTH CHANGE **********

********** START OF FIFTH CHANGE **********

8.3.4.1 Interface Class IpFwFaultManager

Inherits from: IpInterface.

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

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

Method

activityTestReq()

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

Parameters

activityTestID : in TpActivityTestID

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

testSubject : in TpSubjectType

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

Raises

TpCommonExceptions

Method
svcActivityTestRes()

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

Parameters

activityTestID : in TpActivityTestID

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

activityTestResult : in TpActivityTestRes

The result of the activity test.

Raises

TpCommonExceptions,P_INVALID_ACTIVITY_TEST_ID

Method
appUnavailableInd()

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

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions

Method
genFaultStatsRecordReq()

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

Parameters

timePeriod : in TpTimeInterval

The period over which the fault statistics are to be generated. ~~A null~~Supplying both a start time and stop time as empty strings ~~value~~ leaves ~~this~~ 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

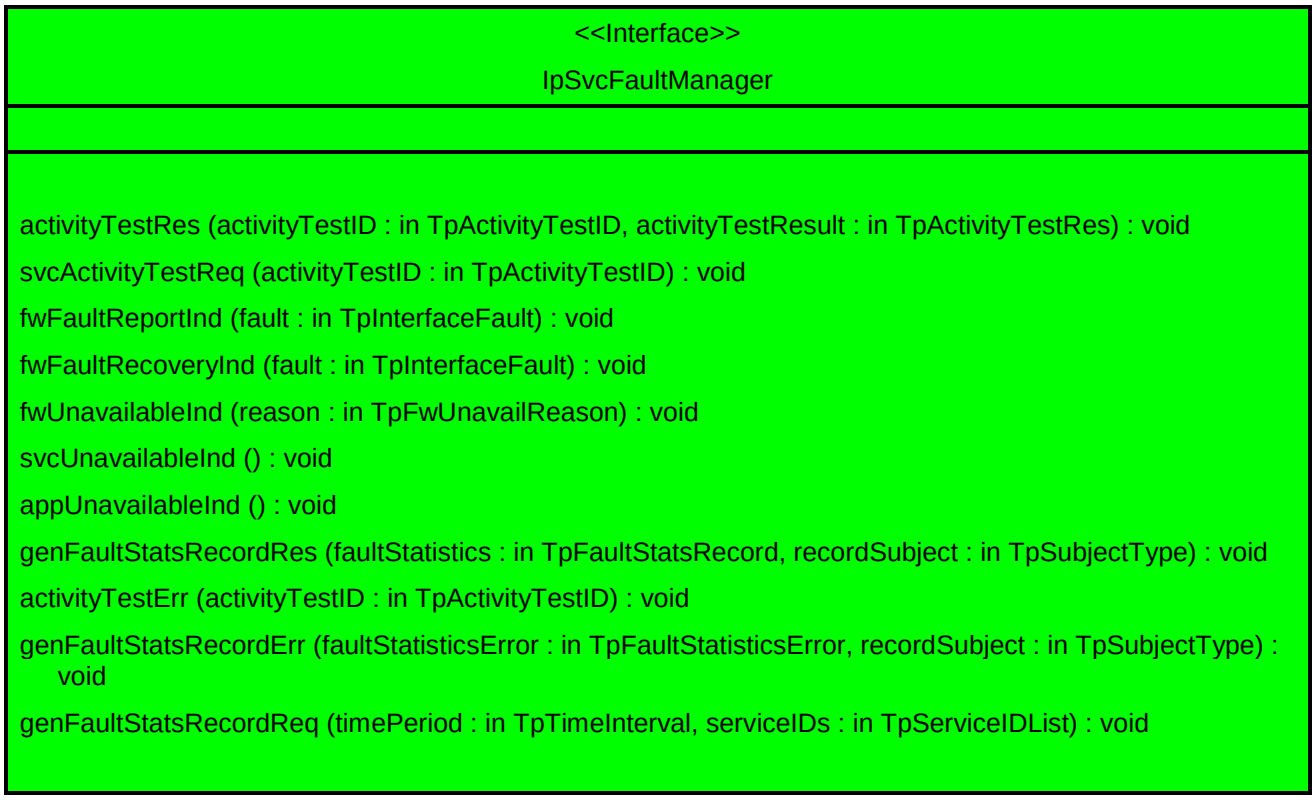
********** END OF FIFTH CHANGE **********

********** START OF SIXTH CHANGE **********

8.3.4.2 Interface Class IpSvcFaultManager

Inherits from: IpInterface.

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



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. ~~A null~~Supplying both a start time and stop time as empty strings ~~value~~ leaves ~~this~~ 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

********** END OF SIXTH CHANGE **********

CR-Form-v5

CHANGE REQUEST

⌘ **29.198-03** **CR** **058** ⌘ **rev** **-** ⌘ Current version: **5.0.0** ⌘

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

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

Title:	⌘	Add Negotiation of Authentication Mechanism for OSA level Authentication
Source:	⌘	CN5
Work item code:	⌘	OSA2
Date:	⌘	12/07/2002
Category:	⌘	**F**
Release:	⌘	*REL-5*

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

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

Reason for change: ⌘

TS 29.198-3 relies on the use of a challenge-based mechanism (CHAP as per IETF RFC 1994) for authentication of the client application by the framework, and vice-versa. CHAP is chosen as the authentication scheme when the authentication type in the initiateAuthenticate() method is set to P_OSA_AUTHENTICATION.

CHAP requires the support of the MD5 hashing algorithm, as a minimum, with an allowance for support of other algorithms, but no other algorithm is specifically referred to in RFC 1994.

However, since RFC 1994 has been issued, newer, more secure, hashing algorithms have been made available. A mechanism needs to be added to the API to permit negotiation of the hashing algorithm used, in order to take advantage of these newer algorithms.

TS 29.198-3 relies on the use of a challenge-based mechanism (CHAP as per IETF RFC 1994) for authentication of the client application by the framework, and vice-versa. CHAP is chosen as the authentication scheme when the authentication type in the initiateAuthenticate() method is set to P_OSA_AUTHENTICATION.

Because of the lack of detailed reference to RFC 1994 in TS 29.198-3, it is not clear whether CHAP-based OSA authentication must format the challenge and response in packets as described in RFC 1994 or must merely follow the rule given for MD5 processing.

If the Challenge and Response packets as defined in RFC 1994 must be used to format the challenge and the response values, then it is not clear as to what the Name field of the Challenge packet must contain. The Name field must indeed be used to identify the sending system. There is no information in the TS as to which value must be put in there.

	OSA requires that the challenge be encrypted, but this is acknowledged to be of no real value regarding the authentication process. Furthermore, use of encryption introduces complication in that OSA contains no procedures for key exchange, and no instruction as to the use of initialisation vectors and padding algorithms etc. The current description of the authentication processes in the Framework needs improvement. There are two authentication processes: one where the Framework authenticates the client, and results in the client's right to invoke requestAccess(), the other where the client authenticates the Framework. These are often mixed in the specification, where there should be no link between them. There is little description of the use of selectEncryptionMethod or of abortAuthentication, or the reasons of consequences of use of these methods. The STD for IpAPILevelAuthentications shows the two authentication processes and a strict order on their use (Framework authenticates client first). The Sequence Diagram "Initial Access" shows the reverse order of use of the authentication processes (client authenticates Framework first). The Sequence Diagram "Initial Access for Trusted Parties" should not use APILevel Authentication interfaces, if both parties know they are trusted. Also, the selectEncryptionMethod() method is not shown in the diagram, and should be required for APILevelAuthentication, since not calling it will prevent the Framework from ever (re-)authenticating the client This CR results in from a lengthy communication with SA3 and certain SA3 security experts, who were instrumental in developing the proposals contained in this CR.
Summary of change: ⌘	Add selectAuthenticationMechanism() to IpAPILevelAuthentication interface to permit the client to offer a choice of mechanisms to the Framework. Furthermore deprecate authenticate() and replace it by the a new method challenge(), which does not require encryption of the challenge string. Properly describe the format of the challenge string in the challenge() method. Deprecate selectEncryptionMethod() as encryption of the challenge string will no longer be required. Add extensible types TpAuthMechanism and TpAuthMechanismList to contain the choice of authentication mechanisms. Add an exception in case no acceptable mechanism is available to the Framework. Sequence Diagrams have been overhauled. Clarifications have been introduced into the descriptions of most authenticaiton methods to split the two authentication processes. Four state transition diagrams have been introduced to replace the existing IpAPILevelAuthentication STD.
Consequences if not approved: ⌘	OSA Authentication will be forced to rely on the old MD5 algorithm, which dates from 1992, when newer, more secure alternatives are available. Interoperability problems will result: The format of the challenge and response in the OSA authentication exchange will remain undefined and open to interpretation. Different vendors will implement their own interpretation, forcing application developers to tailor their applications for each vendor's equipement. Since the interoperability problems will be related to the first contact between an application and the Framework, these problems will have a serious impact on the adoption and success of OSA. The authentication processes form the first point of contact between a Framework and either an application or an SCF. Failure to clarify these processes sufficiently will result in significant interoperabilty problems, before any 'real' interworking between an application and the Framework, or SCFs, can take place

Clauses affected:	⌘	6.1.1, 6.3.1.5, 10.3, 11		
Other specs affected:	⌘	☐ Other core specifications ☐ Test specifications ☐ O&M Specifications	⌘	
Other comments:	⌘			

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

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

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

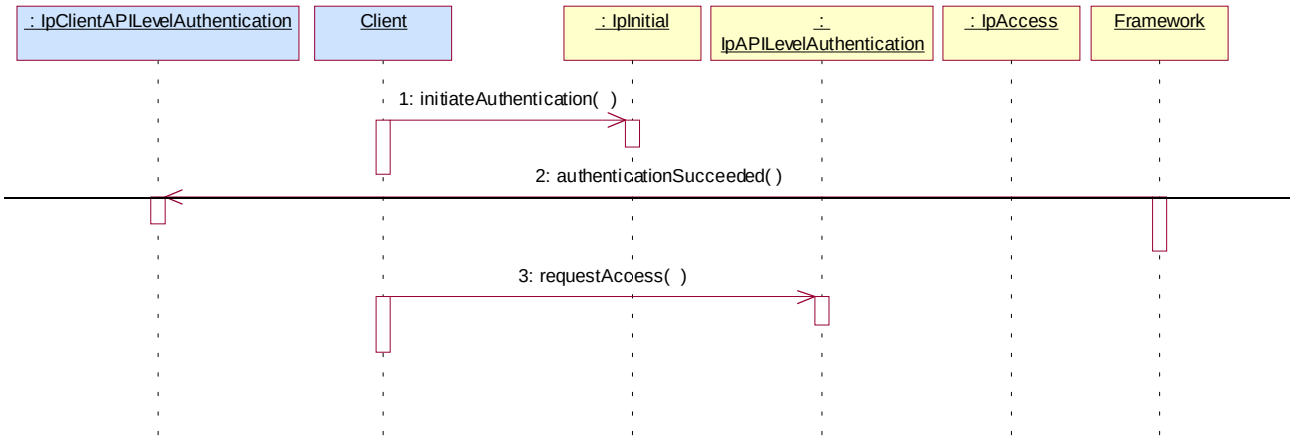
3) With "track changes" disabled, paste the entire CR form (use CTRL-A to select it) into the specification just in front of the clause containing the first piece of changed text. Delete those parts of the specification which are not relevant to the change request.

6.1 Sequence Diagrams

6.1.1 Trust and Security Management Sequence Diagrams

~~6.1.1.1Initial 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~~WithVersion ~~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.~~

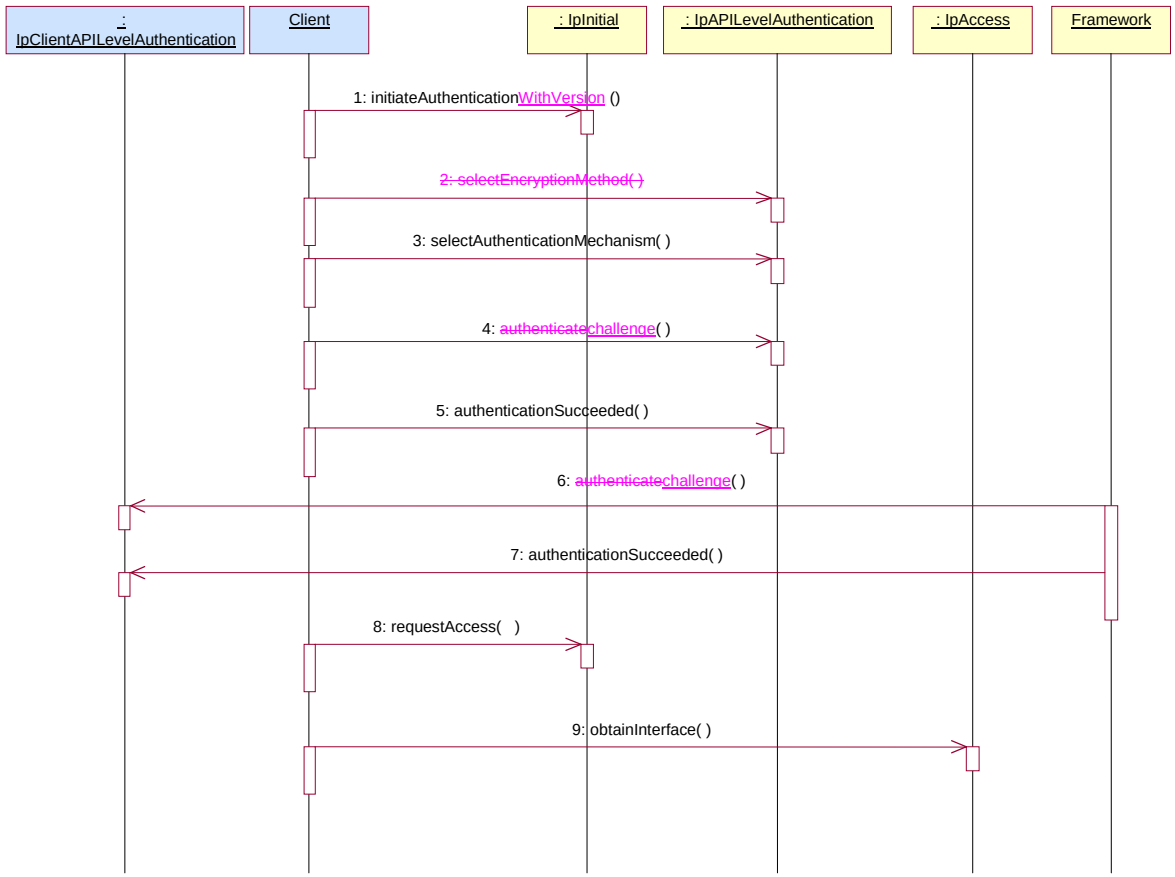
~~6.1.1.2~~6.1.1.1 Initial Access

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

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

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

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



1: Initiate Authentication

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

~~2: Select 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.~~

32: Select Authentication Mechanism

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

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

4: The client provides an indication if authentication succeeded.

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

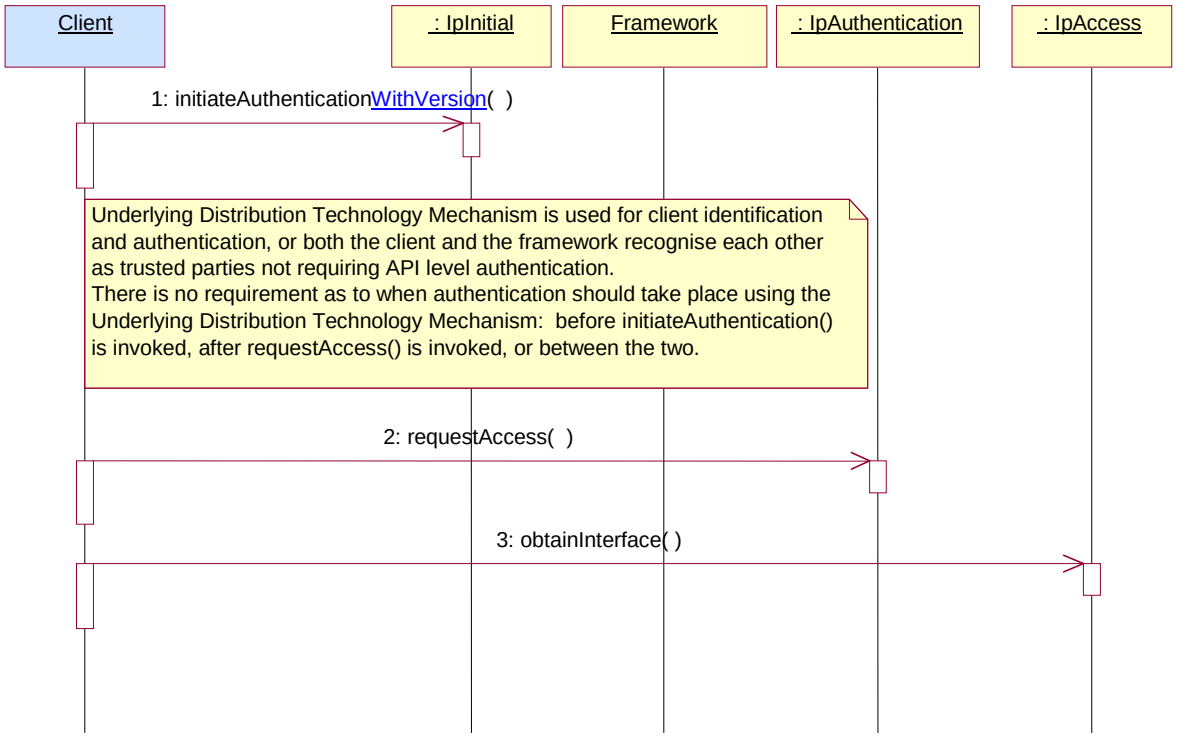
7: Request Access

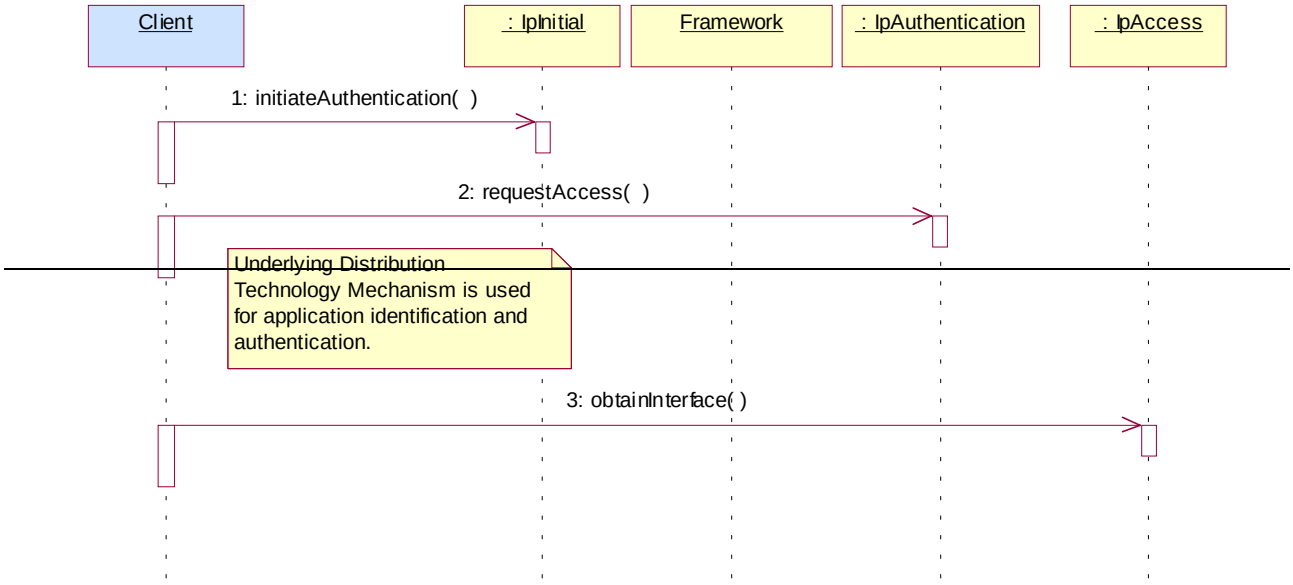
Upon successful ~~(mutual)~~ authentication of the client by the Framework, the client is permitted to invoke~~s~~ requestAccess on the Framework's API Level Authentication interface, providing in turn a reference to its own access interface. The Framework returns a reference to its access interface. The success or failure of the client's authentication of the Framework does not affect the client's right to invoke requestAccess.

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

~~6.1.1.3~~6.1.1.2 Non-API level Authentication

The following figure shows a client accessing the OSA Framework for the first time. ~~This sequence diagram illustrates the two-way mechanism by which the~~ The client and the framework have mutually authenticated one another using an underlying distribution technology mechanism, or the client and the framework recognise each other as a trusted party, not requiring authentication.





1: The client calls initiateAuthenticationWithVersion 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. What that mechanism is, if it even exists, is outside the scope of the API.

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~~6.1.1.3 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.

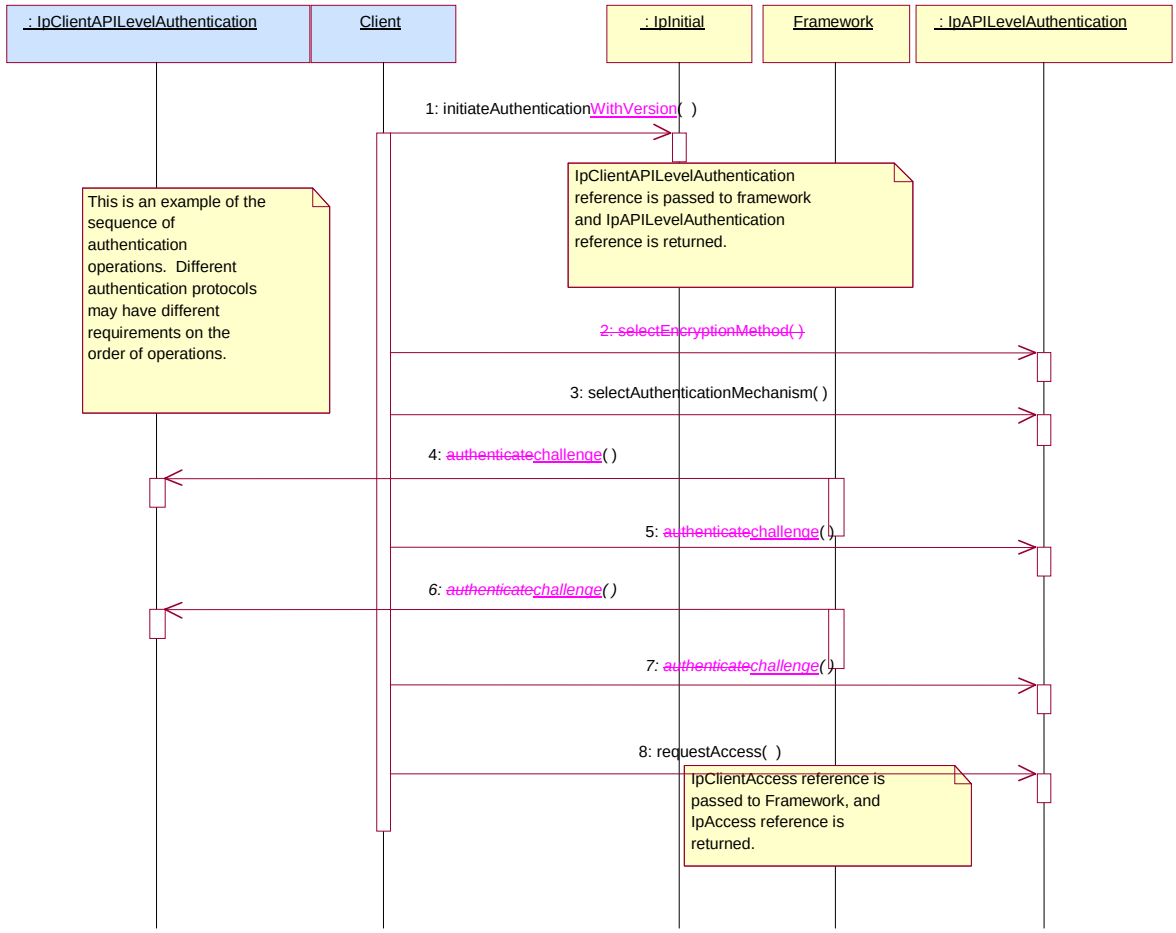
1) The client calls initiateAuthentication<u>WithVersion</u> on the OSA Framework Initial interface. This allows the client to specify the type of authentication process. This authentication process may be specific to the provider, or the implementation technology used. The initiateAuthentication<u>WithVersion</u> 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<u>WithVersion</u> 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.~~

2) The client invokes the selectAuthenticationMechanism on the Framework's API Level Authentication interface. This includes the authentication algorithms supported by the client. The framework then chooses a mechanism based on the capabilities of the client and the Framework. If the client is capable of handling more than one mechanism, then the Framework chooses one option, defined in the prescribedMethod parameter. In some instances, the authentication mechanism of the client may not fulfil the demands of the Framework, in which case, the authentication will fail, for example~~.~~ CHAP prescribes the MD5 hashing algorithm as the minimum to be supported, however the framework need not accept this algorithm.

3) The application and Framework interact to authenticate each other by usin~~h~~g the challenge method. 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. There are in fact two authentication processes: authentication of the client performed by the Framework , and authentication of the Framework performed by the client. Mutual authentication is achieved by both these processes terminating successfully. Mutual authentication may not necessarily be required, i.e. it could be that a client may not need to authenticate the Framework. There is also no required order for the execution of these two authentication processes, however, the client shall respond immediately to any challenge issued by the Framework, as the Framework might not respond to any challenge issued by the client until the Framework has successfully authenticated the client.~~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 of the other side. ~~Re-authentication does not have to be mutual.~~


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
1: initiateAuthenticationWithVersion( )
IpClientAPILevelAuthentication reference is passed to framework and IpAPILevelAuthentication reference is returned.
This is an example of the sequence of authentication operations. Different authentication protocols may have different requirements on the order of operations.
2: selectEncryptionMethod( )
3: selectAuthenticationMechanism( )
4: authenticatechallenge( )
5: authenticatechallenge( )
6: authenticatechallenge( )
7: authenticatechallenge( )
8: requestAccess( )
IpClientAccess reference is passed to Framework, and IpAccess reference is returned.

6.3 Interface Classes

6.3.1 Trust and Security Management Interface Classes

The Trust and Security Management Interfaces provide:

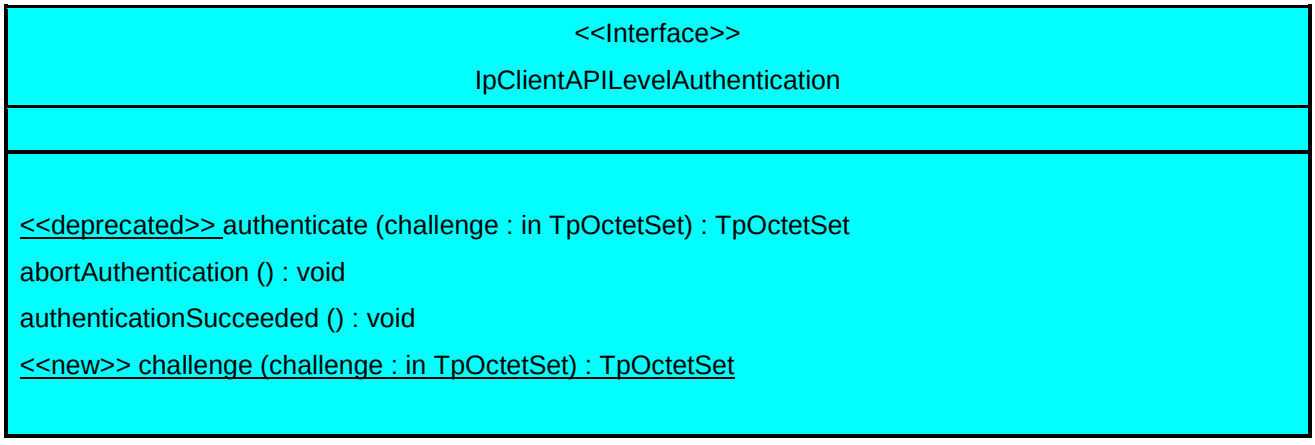
- the first point of contact for a client to access a Framework provider;
- the authentication methods for the client and Framework provider to perform an authentication protocol;
- the client with the ability to select a service capability feature to make use of;
- the client with a portal to access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

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

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

6.3.1.1 Interface Class IpClientAPILevelAuthentication

Inherits from: IpInterface.



Method
`authenticate()`

This method is deprecated and replaced by challenge(). It shall only be used when the deprecated method initiateAuthentication() is used on the IpInitial interface instead of initiateAuthenticationWithVersion(). This method will be removed in a later release of the specification.

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 of the client~~process~~ is deemed successful when the authenticationSucceeded method is invoked by the Framework.

The invocation of this method may be interleaved with authenticate() calls by the client on the IpAPILevelAuthentication interface. The client shall respond immediately to authentication challenges from the Framework, and not wait unitl the Framework has responded to any challenge the client may issue.

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 where the client is authenticating the Framework. This method is invoked if the framework wishes to abort the authentication process before it has been authenticated by the client, (unless the client responded incorrectly to a challenge in which case no further communication with the client should occur.) Calls to this method after the Framework has been authenticated by the client shall not result in an immediate removal of the Framework's authentication (the client may wish to authenticate the Framework again, however).~~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. The client may invoke requestAccess on the Framework's APILevelAuthentication interface following invocation of this method.

Parameters

No Parameters were identified for this method

Method

`challenge()`

This method is used by the framework to authenticate the client. 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 of the client~~process~~ is deemed successful when the authenticationSucceeded method is invoked by the Framework.

The invocation of this method may be interleaved with challenge() calls by the client on the IpAPILevelAuthentication interface. The client shall respond immediately to authentication challenges from the Framework, and not wait unitl the Framework has responded to any challenge the client may issue.

This method shall only be used when the method initiateAuthenticationWithVersion() is used on the IpInitial interface.

Returns <response> : This is the response of the client application to the challenge of the framework in the current sequence. The formatting of this parameter shall be according to section 4.1 of RFC 1994. A complete CHAP Response packet shall be used to carry the response string. The Response packet shall make the contents of this returned parameter. The Name field of the CHAP Response packet shall be present but not contain any useful value.

Parameters

challenge : in TpOctetSet

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

The formatting of the challenge value shall be according to section 4.1 of RFC 1994. A complete CHAP Request packet shall be used to carry the challenge value. The Name field of the CHAP Request packet shall be present but not contain any useful value.

Returns

TpOctetSet

6.3.1.3 Interface Class IpInitial

Inherits from: IpInterface.

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

<<Interface>> IpInitial
<<deprecated>> initiateAuthentication (clientDomain : in TpAuthDomain, authType : in TpAuthType) : TpAuthDomain <<new>> initiateAuthenticationWithVersion (clientDomain : in TpAuthDomain, authType : in TpAuthType, frameworkVersion : in TpVersion) : TpAuthDomain

6.3.1.3.1 Method <<deprecated>> initiateAuthentication()

This method is deprecated in this version, this means that it will be supported until the next major release of this specification.

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 registered service (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`

6.3.1.3.2 Method <<new>> initiateAuthenticationWithVersion()

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 using the new method with support for backward compatibility in the framework. The returned fwDomain authInterface will be selected to match the proposed version from the Client in the Framework response. If the Framework can't work with the proposed framework version the framework returns an error code (P_INVALID_VERSION).

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 registered service (i.e. TpServiceInstanceID) or for a service supplier (i.e. TpServiceSupplierID). It is used to identify the client domain to the framework, (see ~~authenticate~~challenge() 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 that is used when an underlying distribution technology authentication mechanism is used.

frameworkVersion : in TpVersion

This identifies the version of the Framework implemented in the client. The TpVersion is a String containing the version number. Valid version numbers are defined in the respective framework specification.

Returns

TpAuthDomain

Raises

TpCommonExceptions, P_INVALID_DOMAIN_ID, P_INVALID_INTERFACE_TYPE, P_INVALID_AUTH_TYPE, P_INVALID_VERSION

6.3.1.4 Interface Class IpAuthentication

Inherits from: IpInterface.

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

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

6.1.1.1.1 Method requestAccess()

Once the client has been authenticated by the framework~~and framework are authenticated~~, the client may invoke~~s~~ 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~~has been successfully ~~completed the authentication~~authenticated ~~process~~, then the request fails, and an error code (P_ACCESS_DENIED) is returned.

This method may be invoked by the client immediately on IpAuthentication, when API Level authentication is not being used, since there is no indication to the client at API level that it is authenticated with the Framework.

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

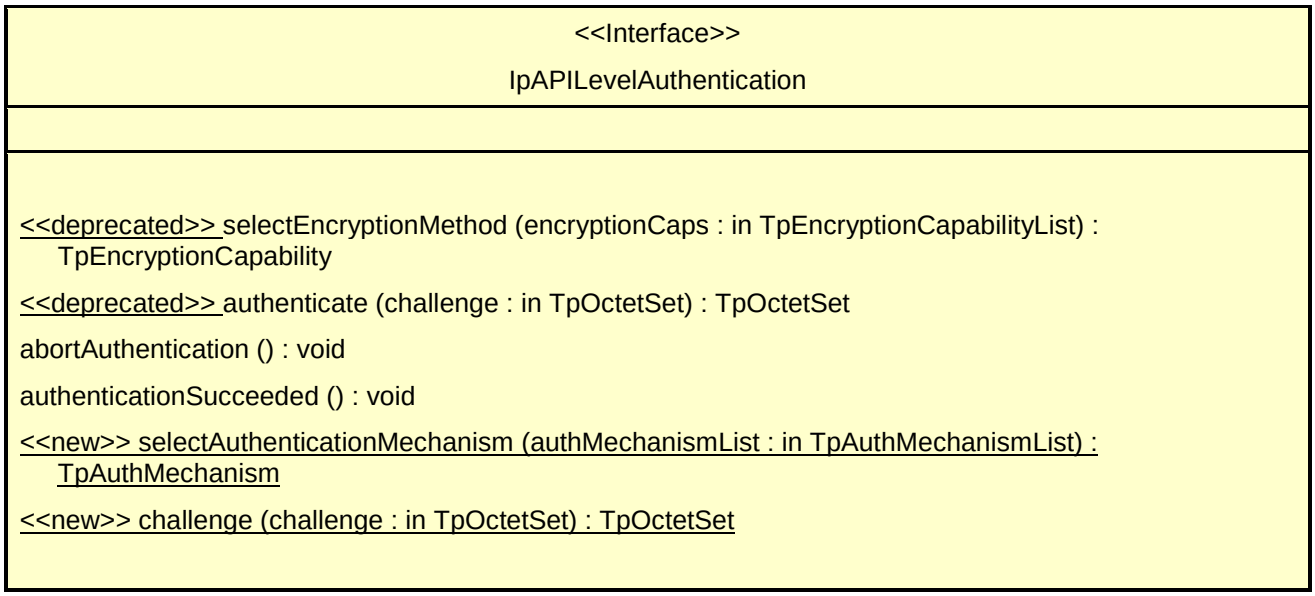
Raises

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 authenticate~~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~~. It is also used to initiate the authentication process.



Method

selectEncryptionMethod()

This method is deprecated and replaced by selectAuthenticationMechanism(). It shall only be used when the IpAPILevelAuthentication interface is obtained by using the deprecated method initiateAuthentication() instead of initiateAuthenticationWithVersion() on the IpInitial interface. This method will be removed in a later release.

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 deprecated and replaced by challenge(). It shall only be used when the IpAPILevelAuthentication interface is obtained by using the deprecated method initiateAuthentication() instead of initiateAuthenticationWithVersion() on the IpInitial interface. This method will be removed in a later release.

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 of the framework ~~process~~ is deemed successful when the authenticationSucceeded method is invoked by the client.

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 where the framework is authenticating the client. 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 before the client has been authenticated by the Framework, calls to the requestAccess operation on

IpAPILevelAuthentication will return an error code (P_ACCESS_DENIED), until the client has been properly authenticated. If this method is invoked after the client has been authenticated by the Framework, it shall not result in the immediate removal of the client's authentication. (The Framework may wish to authenticate the client again, however).

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. Calls to this method have no impact on the client's rights to call requestAccess(), which depend exclusively on the framework's successful authentication of the client.

Parameters

No Parameters were identified for this method

Raises

TpCommonExceptions, P_ACCESS_DENIED

Method

selectAuthenticationMechanism()

The client uses this method to inform the Framework of the different authentication mechanisms it supports as part of API level Authentication. The Framework will select one of the suggested authentication mechanisms and that mechanism shall be used for authentication by both Framework and Client. The authentication mechanism chosen as a result of the response to this method remains valid for an instance of IpAPILevelAuthentication and until this method is re-invoked by 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_AUTHENTICATION_MECHANISM exception.

This method shall only be used when the IpAPILevelAuthentication interface is obtained by using initiateAuthenticationWithVersion() on the IpInitial interface.

Returns: selectedMechanism. This is the authentication mechanism chosen by the Framework. The chosen mechanism shall be taken from the list of mechanisms proposed by the Client.

Parameters

authMechanismList : in TpAuthMechanismList

The list of authentication mechanisms supported by the client.

Returns

TpAuthMechanism

Raises

TpCommonExceptions, P_ACCESS_DENIED, P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM

Method

challenge()

This method is used by the client to authenticate the framework. The framework must respond with the correct responses to the challenges presented by the client. The number of exchanges is dependent on the policies of each side. The authentication of the framework is deemed successful when the authenticationSucceeded method is invoked by the client.

The invocation of this method may be interleaved with challenge() calls by the framework on the client's APILevelAuthentication interface.

This method shall only be used when the IpAPILevelAuthentication interface is obtained by using initiateAuthenticationWithVersion() on the IpInitial interface.

Returns <response> : This is the response of the framework to the challenge of the client in the current sequence. The formatting of this parameter shall be according to section 4.1 of RFC 1994. A complete CHAP Response packet shall be used to carry the response string. The Response packet shall make the contents of this returned parameter. The Name field of the CHAP Response packet shall be present but not contain any useful value.

Parameters

challenge : in TpOctetSet

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

The formatting of the challenge value shall be according to section 4.1 of RFC 1994. A complete CHAP Request packet shall be used to carry the challenge value. The Name field of the CHAP Request packet shall be present but not contain any useful value.

Returns

TpOctetSet

Raises

TpCommonExceptions, P_ACCESS_DENIED

~~6.4~~6.2 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~~6.2.1 Trust and Security Management State Transition Diagrams

~~6.4.1.1~~6.2.1.1 State Transition Diagrams for IpInitial

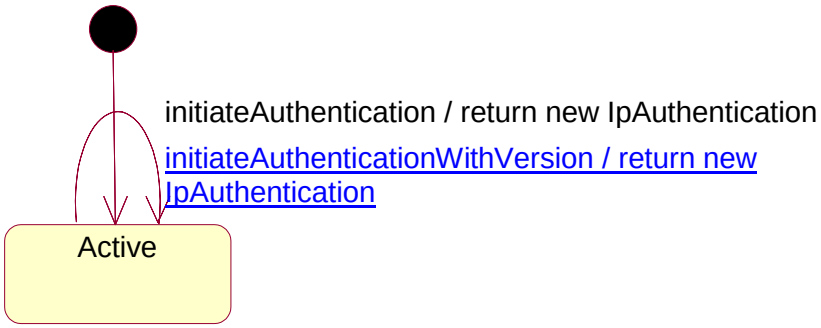


Figure : State Transition Diagram for IpInitial

~~6.4.1.1.1~~6.2.1.1.1 Active State

~~6.4.1.2~~6.2.1.2 State Transition Diagrams for IpAPILevelAuthentication

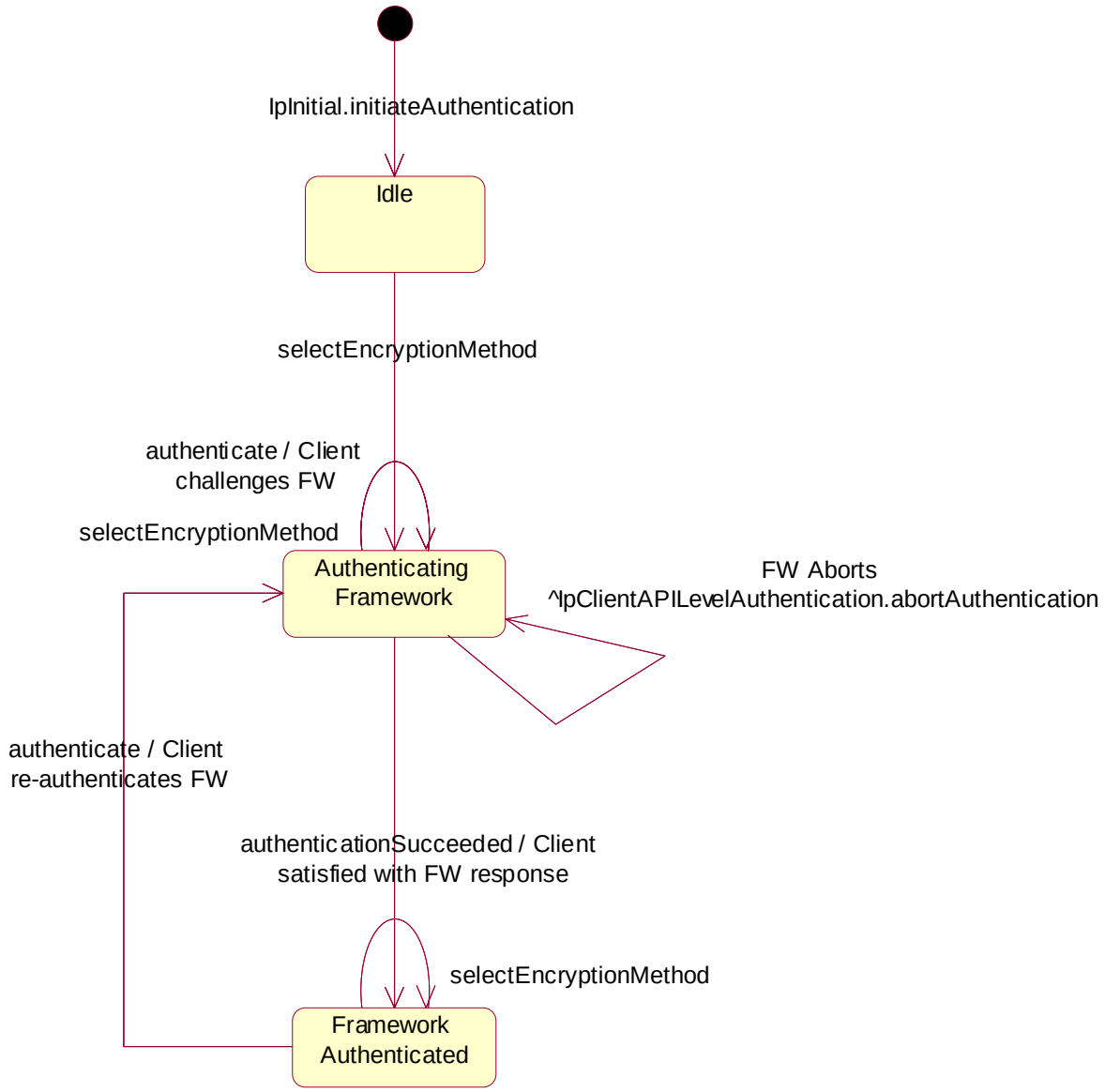
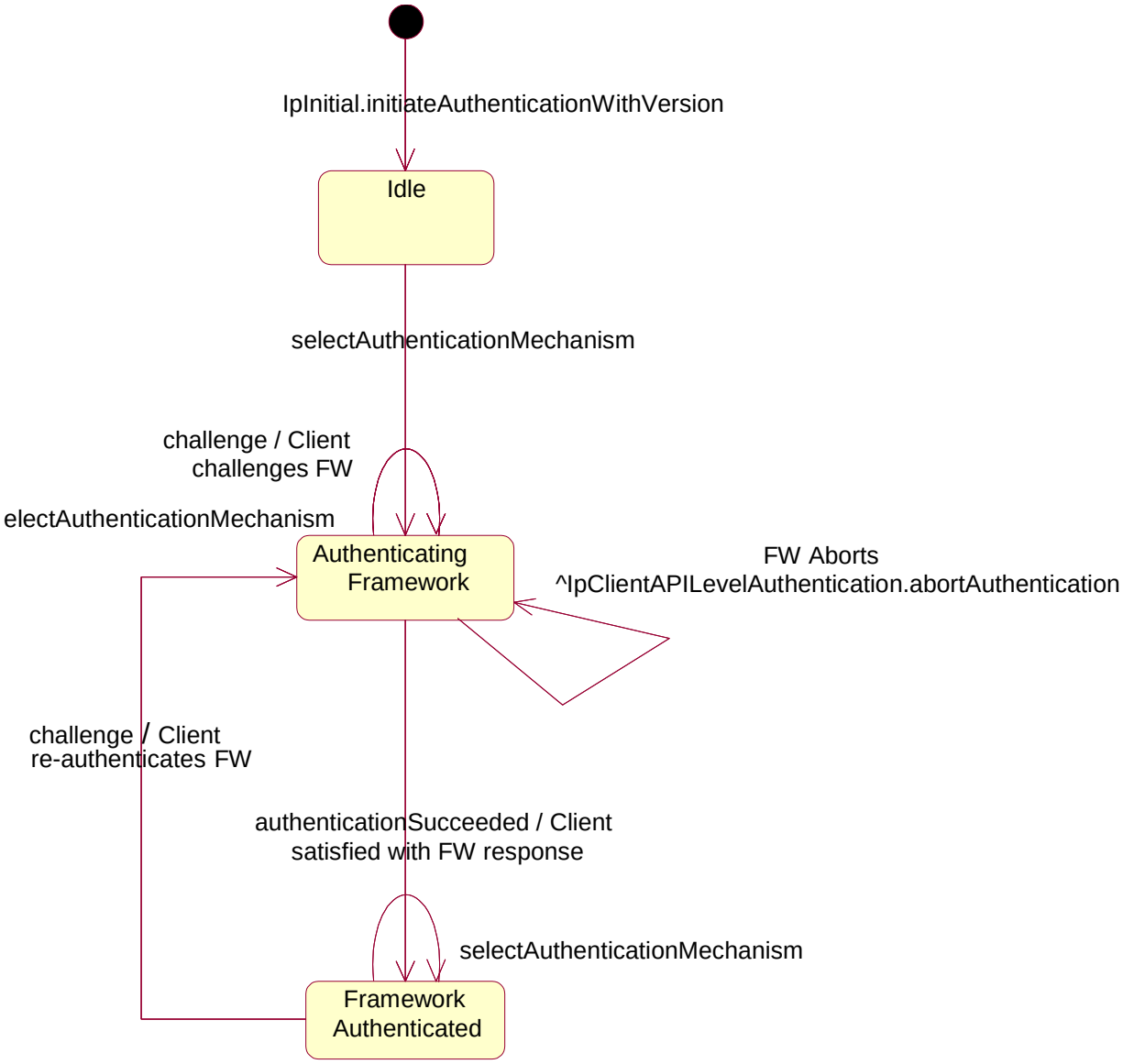


Figure : STD for IpAPILevelAuthentication: Client authenticates Framework using deprecated initiateAuthentication() and authenticate() method combination


IpInitial.initiateAuthenticationWithVersion
Idle
selectAuthenticationMechanism
challenge / Client
challenges FW
electAuthenticationMechanism
Authenticating
Framework
FW Aborts
^IpClientAPILevelAuthentication.abortAuthentication
challenge / Client
re-authenticates FW
authenticationSucceeded / Client
satisfied with FW response
selectAuthenticationMechanism
Framework
Authenticated

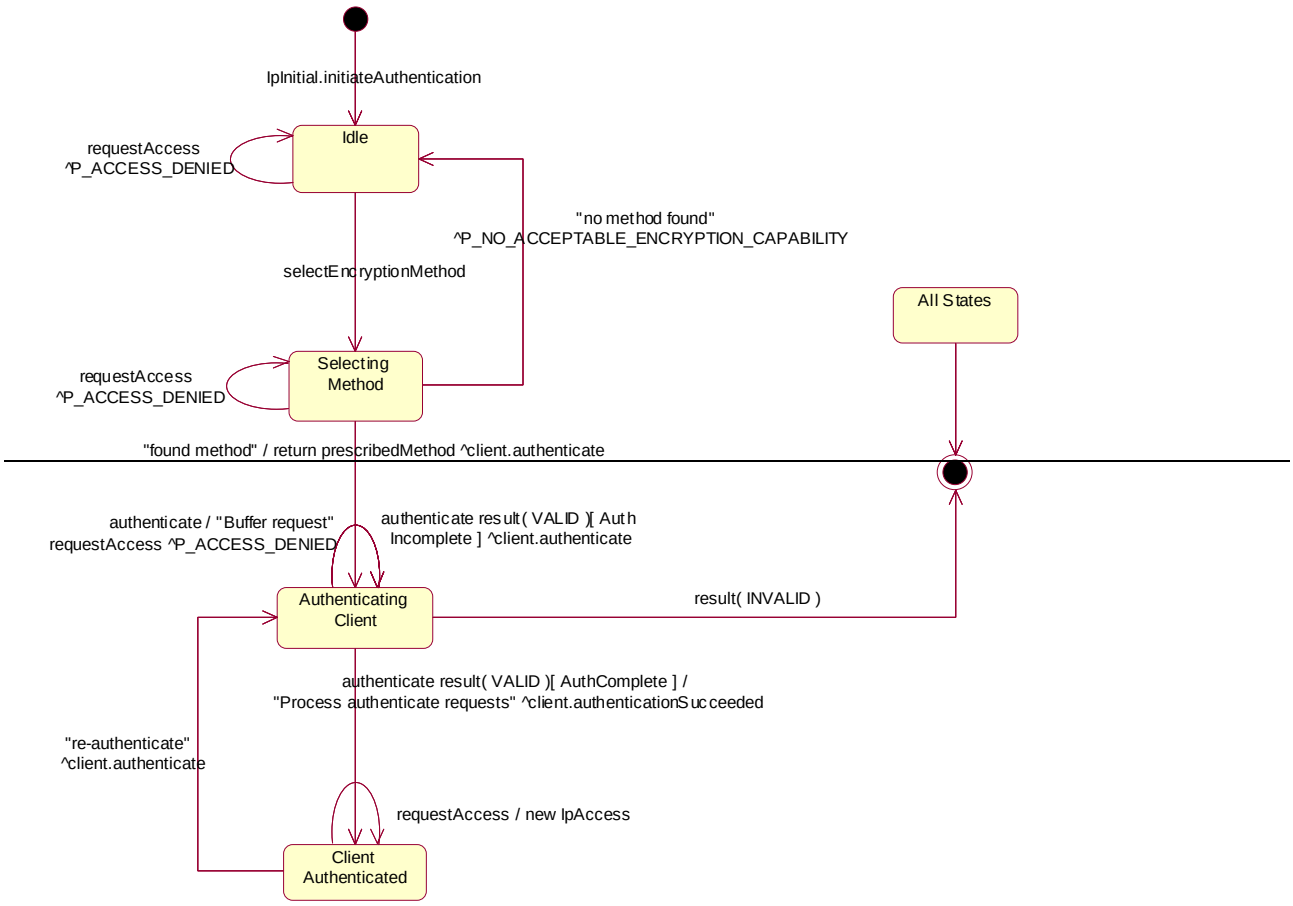


Figure : STD for IpAPILevelAuthentication: Client authenticates Framework using initiateAuthenticationWithVersion() and challenge() method combination~~State Transition Diagram for IpAPILevelAuthentication~~

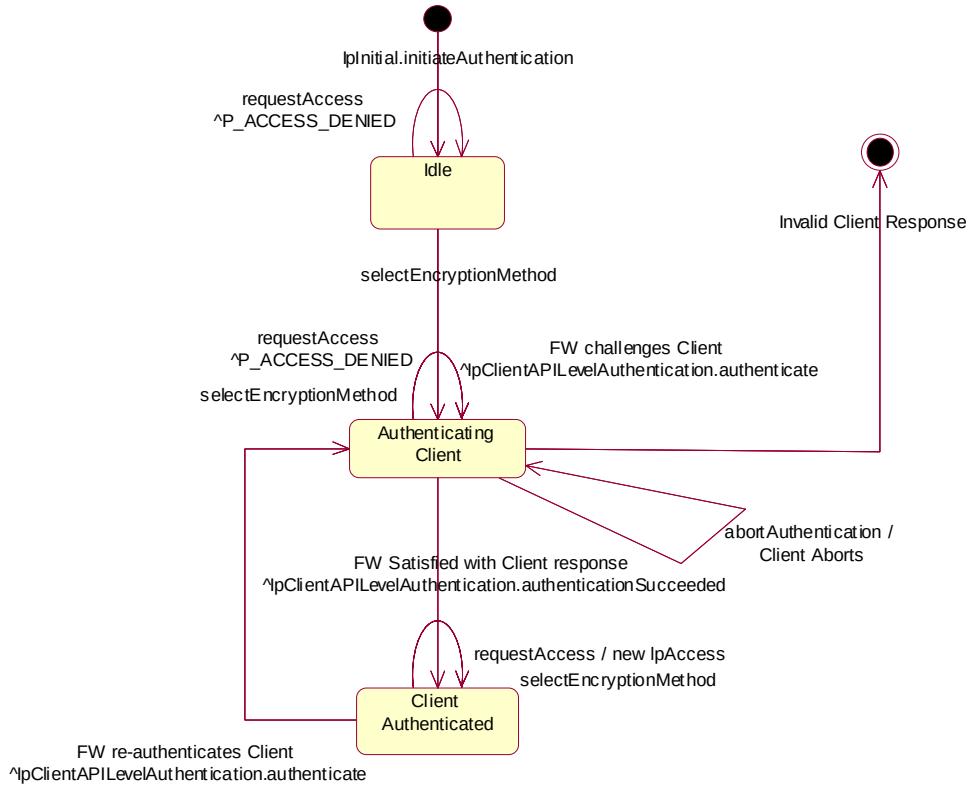


Figure : STD for IpAPILevelAuthentication: Framework authenticates Client using deprecated initiateAuthentication() and authenticate() method combination

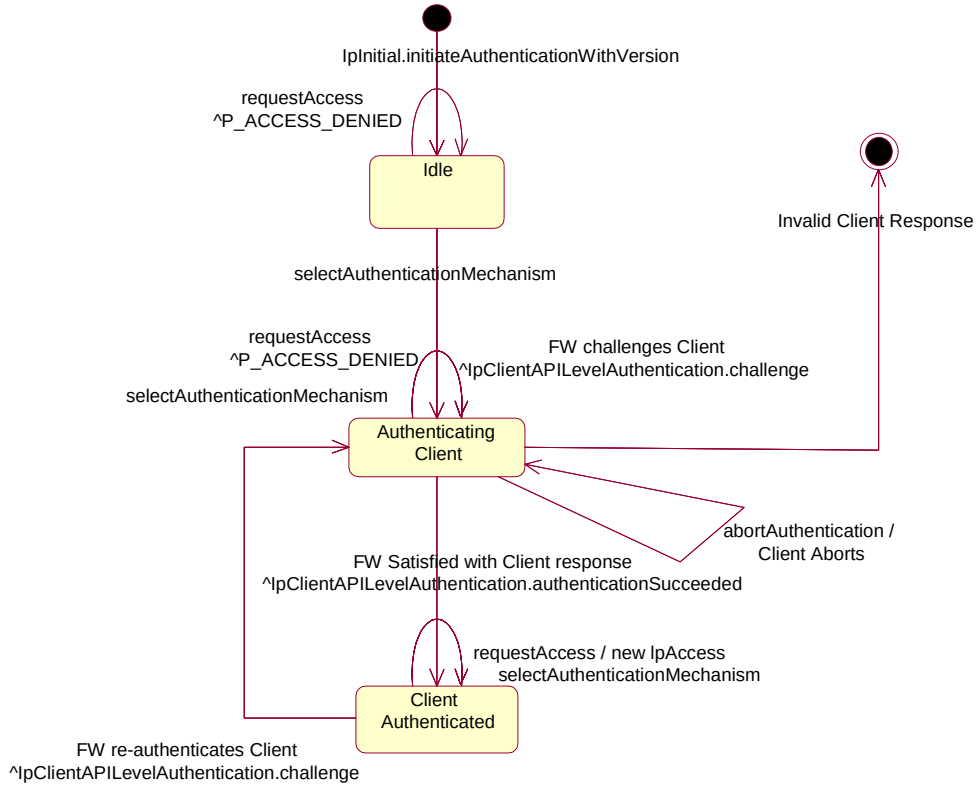


Figure : STD for IpAPILevelAuthentication: Framework authenticates Client using initiateAuthenticationWithVersion() and challenge() method combination

~~6.4.1.2.1~~6.2.1.2.1 Idle State

When the client has invoked the IpInitial initiateAuthentication or the initiateAuthenticationWithVersion method, an object implementing the IpAPILevelAuthentication interface is created. If the client used initiateAuthentication, ~~T~~the client now has to provide its encryption capabilities by invoking selectEncryptionMethod. If the client used initiateAuthenticationWithVersion, the client now has to select the authentication mechanism to be used using selectAuthenticationMechanism.

~~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.~~

6.2.1.2.2 Authenticating Framework State

When entering this state, the client requests the Framework to authenticate itself. ~~by~~ The client invok~~ing~~es the authenticate method on the Framework if it has used initiateAuthentication followed by selectEncryptionMethod (deprecated mechanism). The client invokes the challenge on the Framework if it has used selectAuthenticationMechanism followed by selectAuthenticationMechanism. The Framework may either buffer the requests and respond when the client has been authenticated, or respond immediately, depending on policy. When the client has processed the response from the authenticate request on the Framework, the response is analysed. If the response is valid but the authentication process is not yet complete, then another authenticate request or challenge is sent to the Framework. If the response is valid and the authentication process has been completed, then a transition to the state Framework Authenticated is made and the Framework is informed of its success by invoking authenticationSucceeded. At any time the Framework may abort the authentication process by calling abortAuthentication on the client's APILevelAuthentication interface. The client may also call selectEncryptionMethod to choose other encryption capabilities, or call selectAuthenticationMechanism to choose another hash algorithm.

~~6.4.1.2.3~~6.2.1.2.3 Authenticating Client State

When entering this state, the Framework requests the client to authenticate itself. ~~by~~ The Framework invoke~~sing~~ the a~~A~~uthenticate method on the client if the client~~.~~ has used initiateAuthentication followed by selectEncryptionMethod (deprecated mechanism). The Framework invokes the challenge on the client if the client has used selectAuthenticationMechanism followed by selectAuthenticationMechanism. ~~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 or challenge is sent to the client. If the response is valid and the authentication process has been completed, then a transition to the state Client Authenticated 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 or the initiateAuthenticationWithVersion method on the IpInitial interface. At any time the client may abort the authentication process by calling abortAuthentication on the Framework's IpAPILevelAuthentication interface. The client may also call selectEncryptionMethod to choose other encryption capabilities, or call selectAuthenticationMechanism to choose another hash algorithm.

6.2.1.2.4 Framework Authenticated State

This state is entered when the client indicates that the Framework has been authenticated, by calling authenticationSucceeded on the Framework's IpAPILevelAuthentication interface. The client may at any time request

re-authentication of the Framework, by calling the authenticate method if it had previously used the initiateAuthentication method on IpInitial, or by calling the challenge method if it had previously used the initiateAuthenticationWithVersion method on IpInitial, resulting in a transition back to Authenticating Framework state. The client may also call selectEncryptionMethod to choose other encryption capabilities, or call selectAuthenticationMechanism to choose another hash algorithm.

~~6.4.1.2.5~~6.2.1.2.5 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 or challenge, depending on whether initiateAuthentication or initiateAuthenticationWithVersion was previously used, is sent to the client and a transition back to the AuthenticatingClient state occurs. The client may also call selectEncryptionMethod to choose other encryption capabilities, or call selectAuthenticationMechanism to choose another hash algorithm.

~~6.4.1.3~~6.2.1.3 State Transition Diagrams for IpAccess

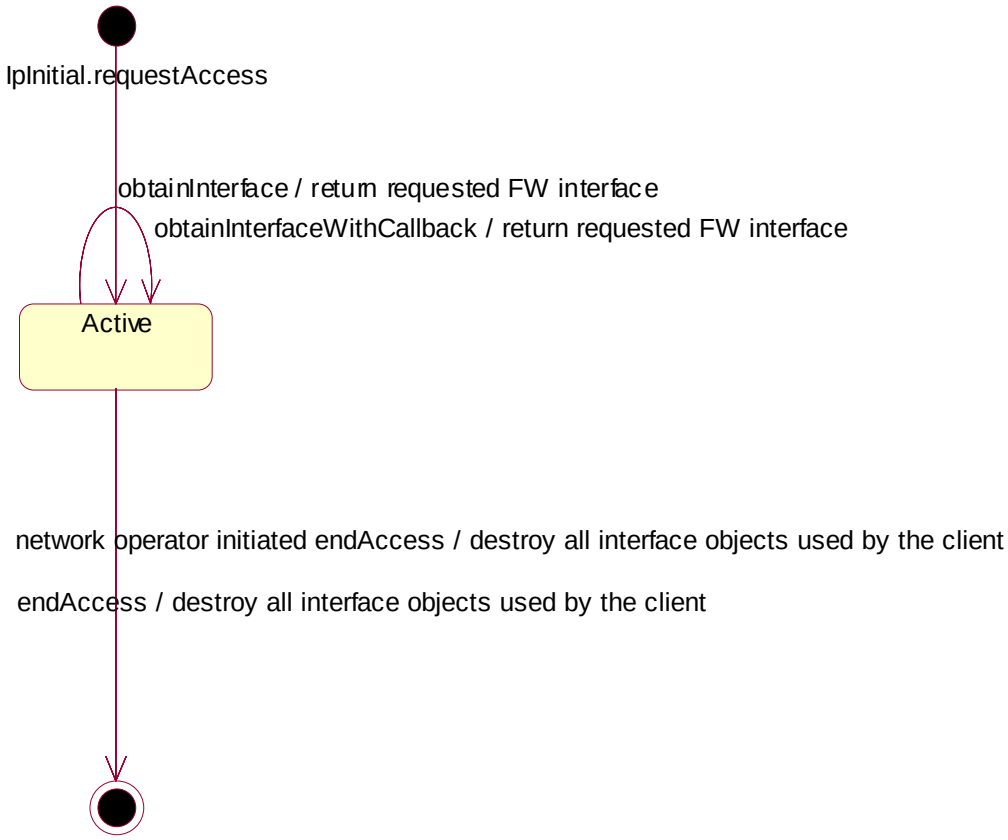


Figure : State Transition Diagram for IpAccess

~~6.4.1.3.1~~6.2.1.3.1 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.

10.3 Trust and Security Management Data Definitions

10.3.1 TpAccessType

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

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

10.3.2 TpAuthType

This data type is identical to a TpString. It identifies the type of authentication mechanism requested by the client. It provides Network operators and 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.

10.3.3 TpEncryptionCapability

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

String Value	Description
NULL	An empty (NULL) string indicates no client capabilities.
P_DES_56	A simple transfer of secret information that is shared between the client application and the Framework with protection against interception on the link provided by the DES algorithm with a 56-bit shared secret key.
P_DES_128	A simple transfer of secret information that is shared between the client entity and the Framework with protection against interception on the link provided by the DES algorithm with a 128-bit shared secret key.
P_RSA_512	A public-key cryptography system providing authentication without prior exchange of secrets using 512-bit keys.
P_RSA_1024	A public-key cryptography system providing authentication without prior exchange of secrets using 1 024-bit keys.

10.3.4 TpEncryptionCapabilityList

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

10.3.5 TpEndAccessProperties

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

10.3.6 TpAuthDomain

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

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

10.3.7 TpInterfaceName

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

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

10.3.8 TpInterfaceNameList

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

10.3.9 TpServiceToken

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

10.3.10 TpSignatureAndServiceMgr

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

Sequence Element Name	Sequence Element Type
DigitalSignature	TpOctetSet
ServiceMgrInterface	IpServiceRef

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

The ServiceMgrInterface is a reference to the SCF manager interface for the selected SCF.

10.3.11 TpSigningAlgorithm

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

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

10.3.12 TpAuthMechanism

This data type is identical to a TpString. It identifies an authentication mechanism to be used for API Level Authentication. The following values are defined:

String Value	Description
P_OSA_MD5	Authentication is based on the use of MD5 (RFC 1321) hashing algorithm to generate a response based on a shared secret and a challenge received via authenticate() method. The capability to use this algorithm is required to be supported when using CHAP (RFC 1994) but its use is not recommended.
P_OSA_HMAC_SHA1_96	Authentication is based on the use of HMAC-SHA1 (RFC 2404) hashing algorithm to generate a response based on a shared secret and a challenge received via authenticate() method.
P_OSA_HMAC_MD5_96	Authentication is based on the use of HMAC-MD5 (RFC 2403) hashing algorithm to generate a response based on a shared secret and a challenge received via authenticate() method.

10.3.13 TpAuthMechanismList

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

11 Exception Classes

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

Name	Description
P_ACCESS_DENIED	The client is not currently authenticated with the framework
P_APPLICATION_NOT_ACTIVATED	An application is unauthorised to access information and request services with regards to users that have deactivated that particular application.
P_DUPLICATE_PROPERTY_NAME	A 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~~ No encryption mechanism, which is acceptable to the framework, is ~~not~~ supported by the client
P_NO_ACCEPTABLE_AUTHENTICATION_MECHANISM	No authentication mechanism, which is acceptable to the framework, is 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