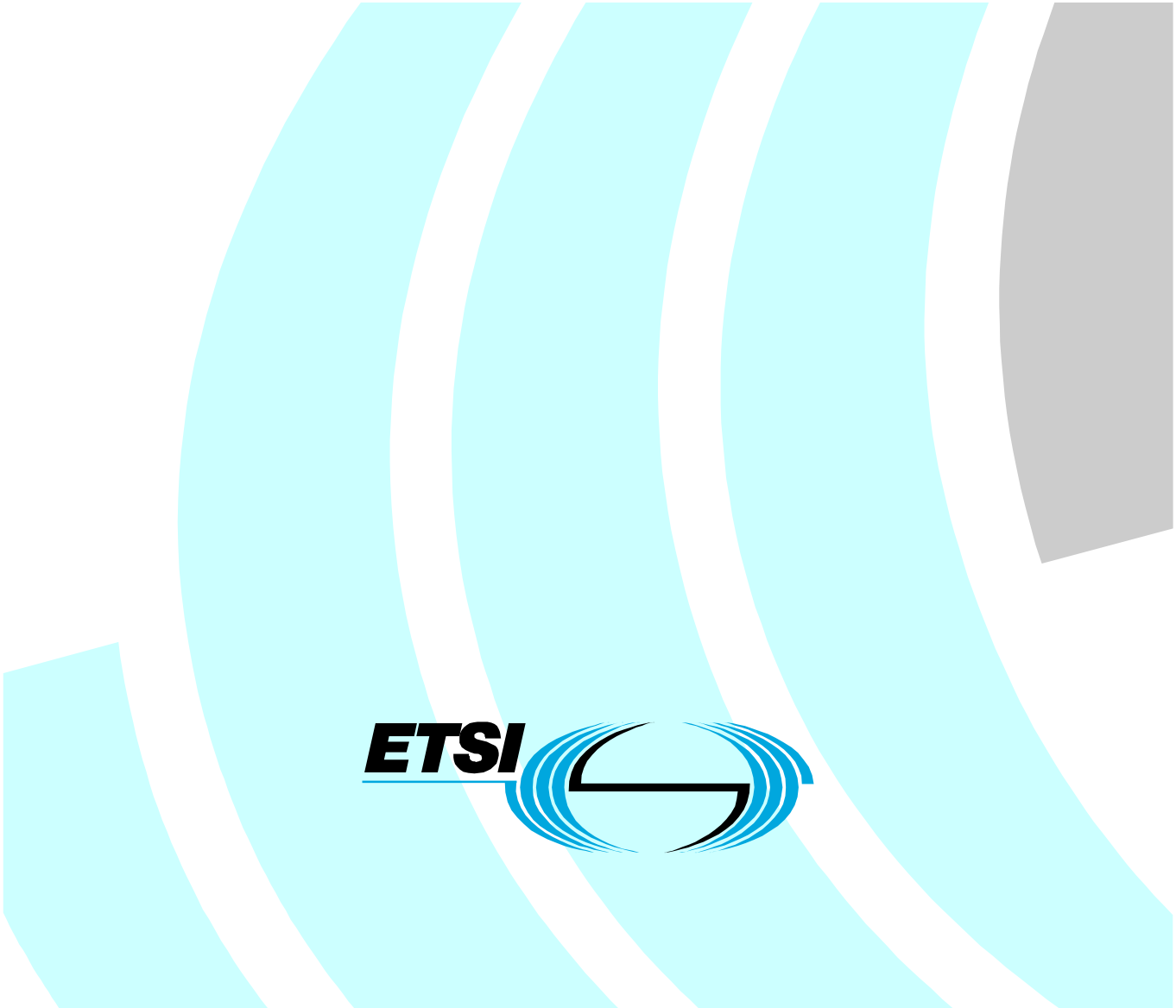
ETSI ES 202 196-3 V1.1.1 (2003-08)

ETSI Standard

Open Service Access (OSA); Application Programming Interface (API); Test Suite Structure and Test Purposes (TSS&TP); Part 3: Framework



Reference
DES/SPAN-120088-3

Keywords
API, OSA, TSS&TP

ETSI

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

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

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

Contents

Intellectual Property Rights

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

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

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Services and Protocols for Advanced Networks (SPAN).

The present document is part 3 of a multi-part deliverable. Full details of the entire series can be found in part 1 [6].

To evaluate conformance of a particular implementation, it is necessary to have a set of test purposes to evaluate the dynamic behaviour of the Implementation Under Test (IUT). The specification containing those test purposes is called a Test Suite Structure and Test Purposes (TSS&TP) specification.

1 Scope

The present document provides the Test Suite Structure and Test Purposes (TSS&TP) specification for the Framework of the Application Programming Interface for Open Service Access (OSA) defined in ES 201 915-3 [1] in compliance with the relevant requirements, and in accordance with the relevant guidance given in ISO/IEC 9646-2 [4] and ETS 300 406 [5].

2 References

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

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

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

[2] ETSI ES 202 170: "Open Service Access (OSA); Application Programming Interface (API); Implementation Conformance Statement (ICS) proforma specification for Framework and SCFs".

[3] ISO/IEC 9646-1: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts".

[4] ISO/IEC 9646-2: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 2: Abstract Test Suite specification".

[5] ETSI ETS 300 406: "Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology".

[6] ETSI ES 202 196-1: "Open Service Access (OSA); Application Programming Interface (API); Test Suite Structure and Test Purposes (TSS&TP); Part 1: Overview".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ES 201 915-3 [1], ISO/IEC 9646-1 [3] and ISO/IEC 9646-2 [4] and the following apply:

abstract test case: Refer to ISO/IEC 9646-1 [3].

Abstract Test Method (ATM): Refer to ISO/IEC 9646-1 [3].

Abstract Test Suite (ATS): Refer to ISO/IEC 9646-1 [3].

Implementation Under Test (IUT): Refer to ISO/IEC 9646-1 [3].

Lower Tester (LT): Refer to ISO/IEC 9646-1 [3].

Implementation Conformance Statement (ICS): Refer to ISO/IEC 9646-1 [3].

ICS proforma: Refer to ISO/IEC 9646-1 [3].

Implementation eXtra Information for Testing (IXIT): Refer to ISO/IEC 9646-1 [3].

IXIT proforma: Refer to ISO/IEC 9646-1 [3].

Test Purpose (TP): Refer to ISO/IEC 9646-1 [3].

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AS	Access Session
ATM	Abstract Test Method
ATS	Abstract Test Suite
FTA	Framework To Application
FTE	Framework To Enterprise
FTS	Framework To Service
FW	FrameWork
ICS	Implementation Conformance Statement
IUT	Implementation Under Test
IXIT	Implementation eXtra Information for Testing
LT	Lower Tester
SCF	Switching Control Function
SUT	System Under Test
TP	Test Purpose
TSM	Trust and Security Management
TSS	Test Suite Structure

4 Test Suite Structure (TSS)

Framework (FW)

- Framework Access Session (AS)
  - Trust and Security Management (TSM)
- Framework to Application (FTA)
  - Service discovery
  - Service agreement management
  - Integrity management
  - Event notification
- Framework to Service (FTS)
  - Service registration
  - Service instance lifecycle management
  - Service discovery
  - Integrity management
  - Event notification
- Framework to Enterprise operator (FTE)
  - Service subscription

5 Test Purposes (TP)

For each test requirement a TP is defined.

5.1 TP naming convention

TPs are numbered, starting at 01, within each group. Groups are organized according to the TSS. Additional references are added to identify the actual test suite (see table 1).

Table 1: TP identifier naming convention scheme

Identifier: <suite_id>_<group>_<nnn>		
<suite_id>	= IUT name:	"FW" for **F**rame**W**ork SCF
<group>	= group number:	two character field representing the group reference according to TSS
<nn>	= sequential number:	(01-99)

5.2 Source of TP definition

The TPs are based on ES 201 915-3 [1].

5.3 Test strategy

As the base standard ES 201 915-3 [1] contains no explicit requirements for testing, the TPs were generated as a result of an analysis of the base standard and the ICS specification ES 202 170 [2].

The TPs are only based on conformance requirements related to the externally observable behaviour of the IUT and are limited to conceivable situations to which a real implementation is likely to be faced (see ETS 300 406 [5]).

5.4 TPs for the Framework

All ICS items referred to in this clause are as specified in ES 202 170 [2] unless indicated otherwise by another numbered reference.

All parameters specified in method calls are valid unless specified.

The procedures to trigger the SCF to call methods in the application are dependant on the underlying network architecture and are out of the scope of this test specification. Those method calls are preceded by the words "Triggered action".

5.4.1 Framework Access Session API

5.4.1.1 Trust and Security Management (TSM)

Methods/Test Nr	01	02	03	04	05	06	07	08	09
initiateAuthentication	X	X	X	X	X	X	X	X	X
requestAccess	X	X	X	X	X	X	X	X	X
selectEncryptionMethod		X	X	X	X	X	X	X	
authenticate		X	X	X	X	X	X	X	
abortAuthentication						X			
AuthenticationSucceeded	X	X	X	X	X		X	X	
obtainInterface		X	X	X				X	X
obtainInterfaceWithCallback					X				
endAccess							X		
listInterfaces		X							
releaseInterface								X	

Test FW_AS_TSM_01

Summary: Initial Access for Trusted Parties, no authentication is needed, all methods, successful

Reference: ES 201 915-3 [1], clause 6.1.1.1

Precondition: Authentication not required by IUT

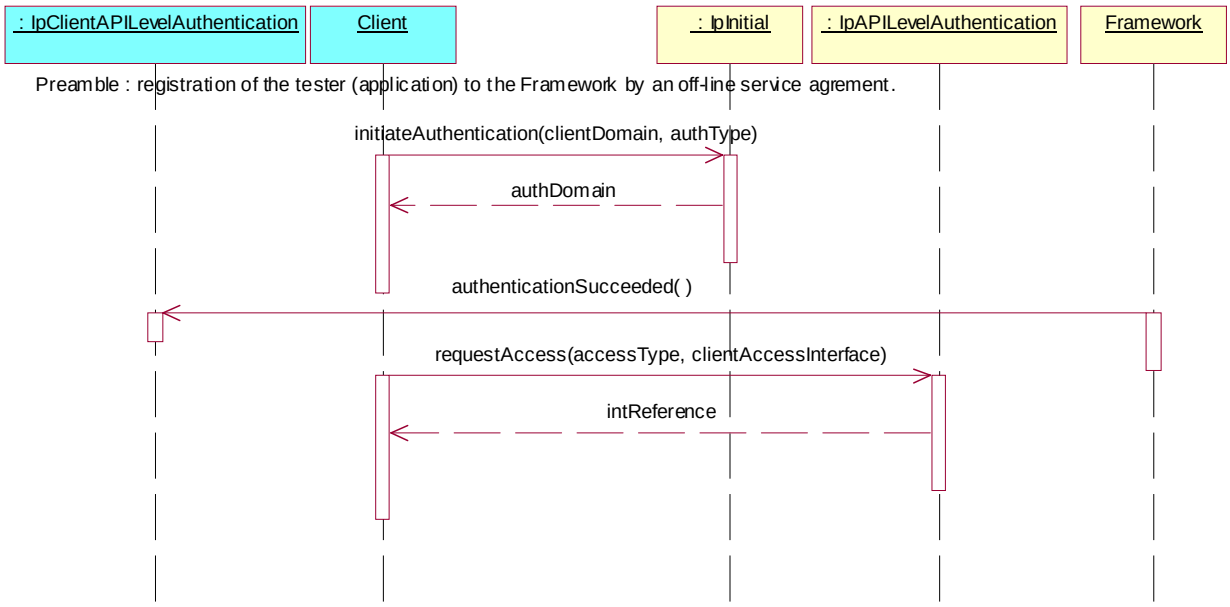
Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication()** on **IpInitial** interface.
   Parameters: clientDomain, authType
   Check: valid value of TpAuthDomain is returned

2. Triggered action: cause IUT to call **authenticationSucceeded()** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

3. Method call **requestAccess()** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned



Test FW_AS_TSM_02

Summary: API level authentication, FW authenticates the client only, all methods, successful, use of **listInterface** method to get the name of supported interfaces

Reference: ES 201 915-3 [1], clauses 6.1.1.2 and 6.1.1.4

Precondition: API level authentication required by IUT, listInterfaces supported

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of **TpEncryptionCapability** is returned

3. Triggered action: cause IUT to call authenticate method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

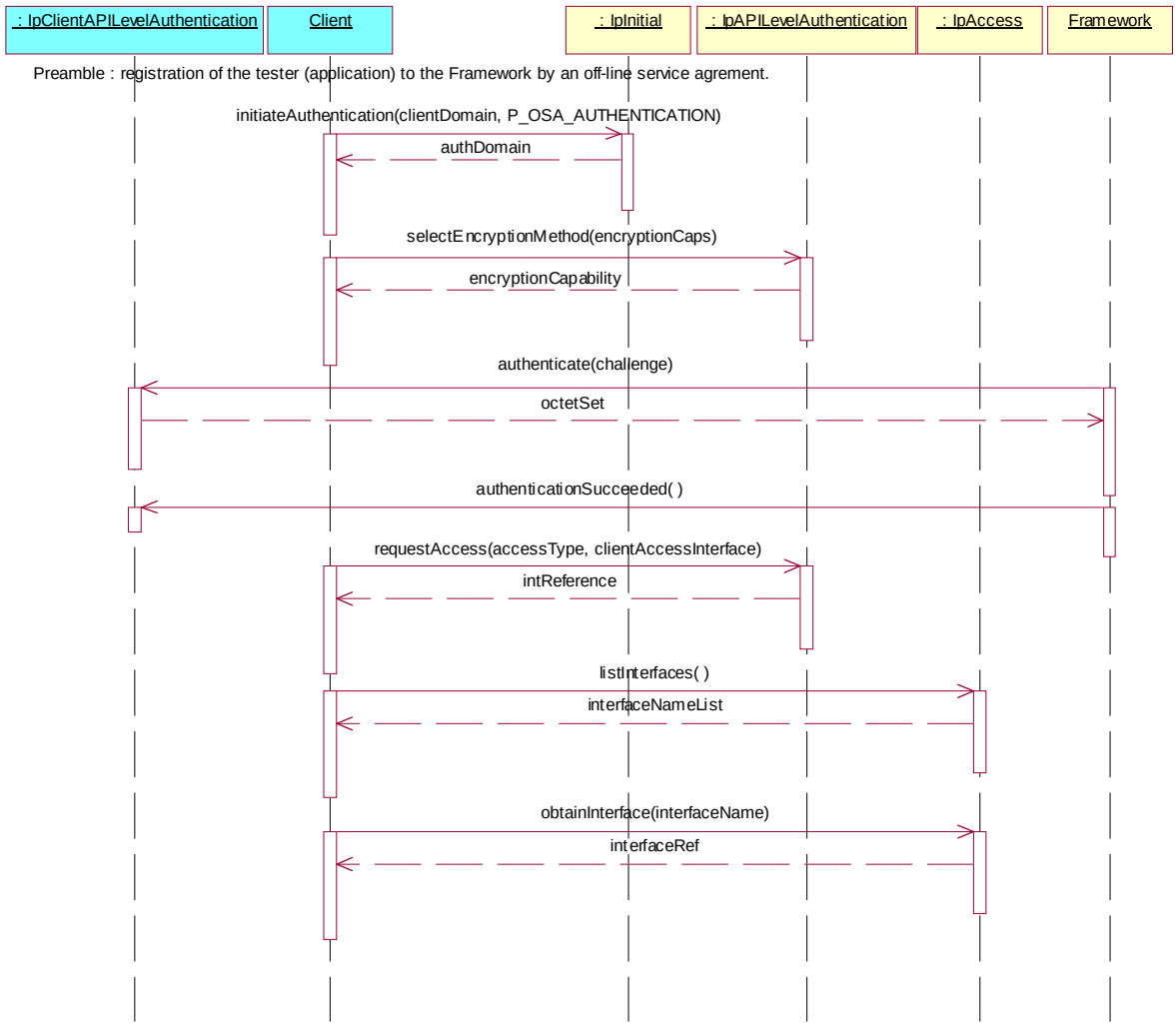
   NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: **accessType**, **clientAccessInterface**
   Check: valid value of **TpInterfaceRef** is returned

6. Method call **listInterfaces** on **IpAccess** interface.
   Parameters: none
   Check: valid value of **TpInterfaceNameList** is returned

7. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: **interfaceName** (suggest use of P_DISCOVERY)
   Check: valid value of **IpInterfaceRef** is returned



Test FW_AS_TSM_03

Summary: API level authentication, FW and client authenticate mutually, all methods, successful

Reference: ES 201 915-3 [1], clauses 6.1.1.2 and 6.1.1.4

Precondition: Authentication required by IUT

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: **clientDomain**, authType=P_OSA_AUTHENTICATION
   Check: valid value of **TpAuthDomain** is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: **encryptionCaps**
   Check: valid value of **TpEncryptionCapability** is returned

3. Triggered action: cause IUT to call **authenticate** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE 1: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **authenticate** on **IpAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of **TpOctetSet** is returned

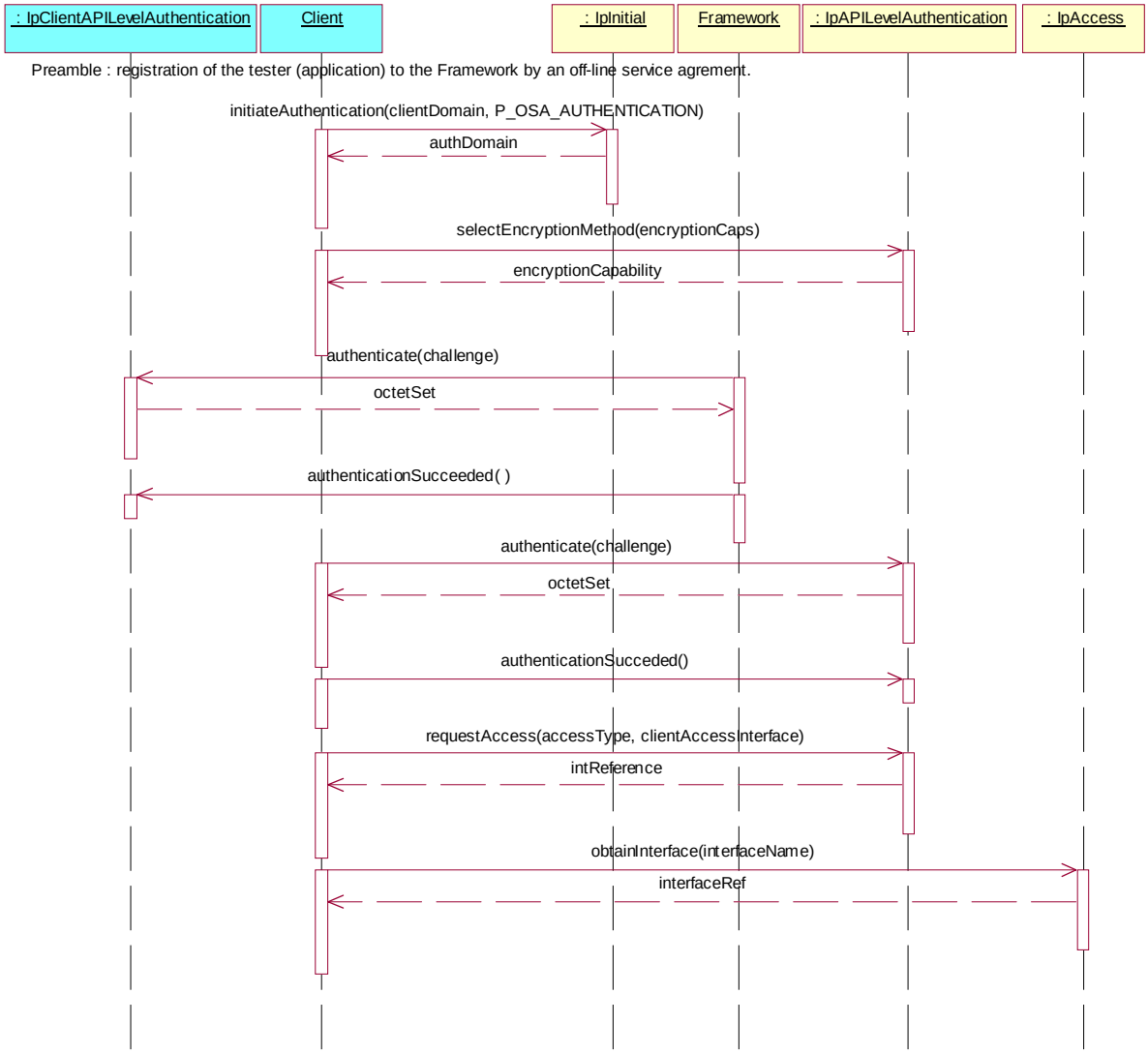
NOTE 2: This method may be repeated with different challenges as required by the tester.

6. Method call **authenticationSucceeded** on **IpAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned.

NOTE 3: The method calls 5. and 6. may interleave between the method calls 3. and 4.

7. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: **accessType**, **clientAccessInterface**
   Check: valid value of **TpInterfaceRef** is returned

8. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: **interfaceName** (suggest use of P_DISCOVERY)
   Check: valid value of **IpInterfaceRef** is returned



Test FW_AS_TSM_04

Summary: API level authentication, FW authenticates the client only, **unsuccessful call of requestAccess** method (preceding **authenticationSucceeded** method call)

Reference: ES 201 915-3 [1], clauses 6.1.1.2 and 6.1.1.4

Precondition: Authentication required by IUT.

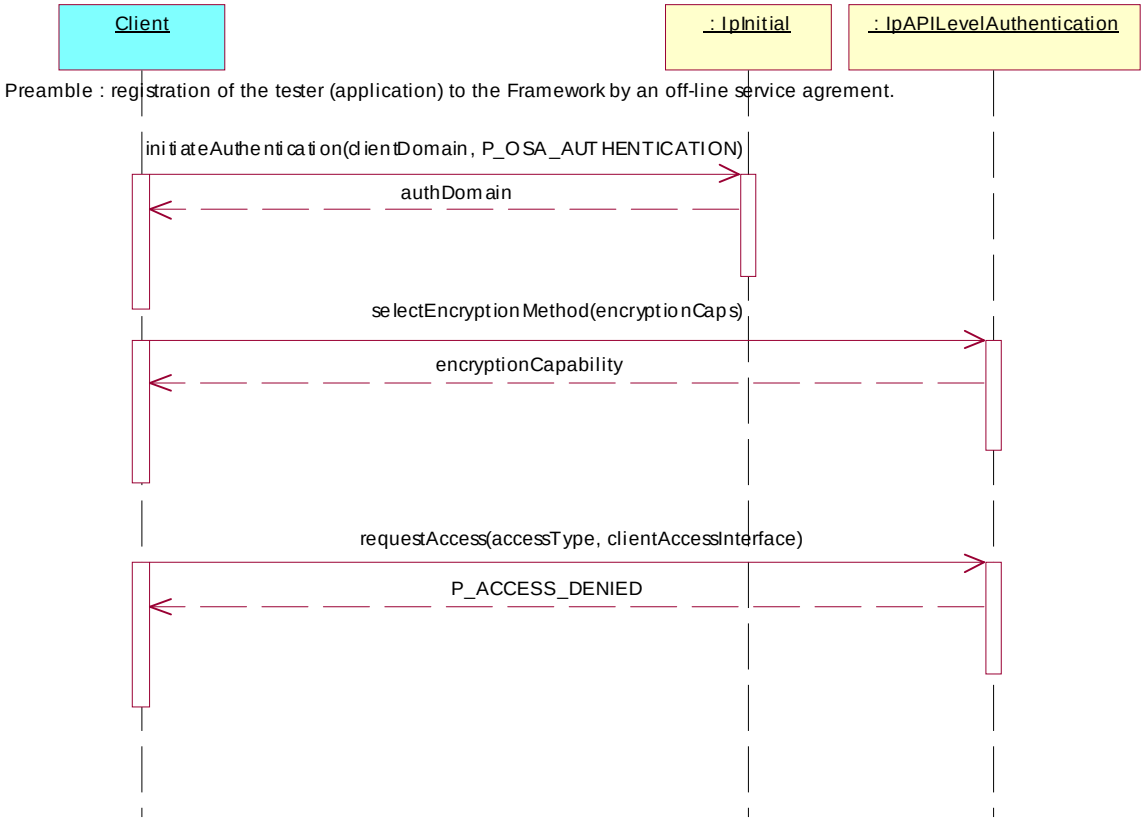
Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

3. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: P_ACCESS_DENIED value is returned



Test FW_AS_TSM_05

Summary: API level authentication, FW authenticates the client only, use of **obtainInterfaceWithCallback**, successful

Reference: ES 201 915-3 [1], clauses 6.1.1.2 and 6.1.1.4

Precondition: Authentication required by IUT

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

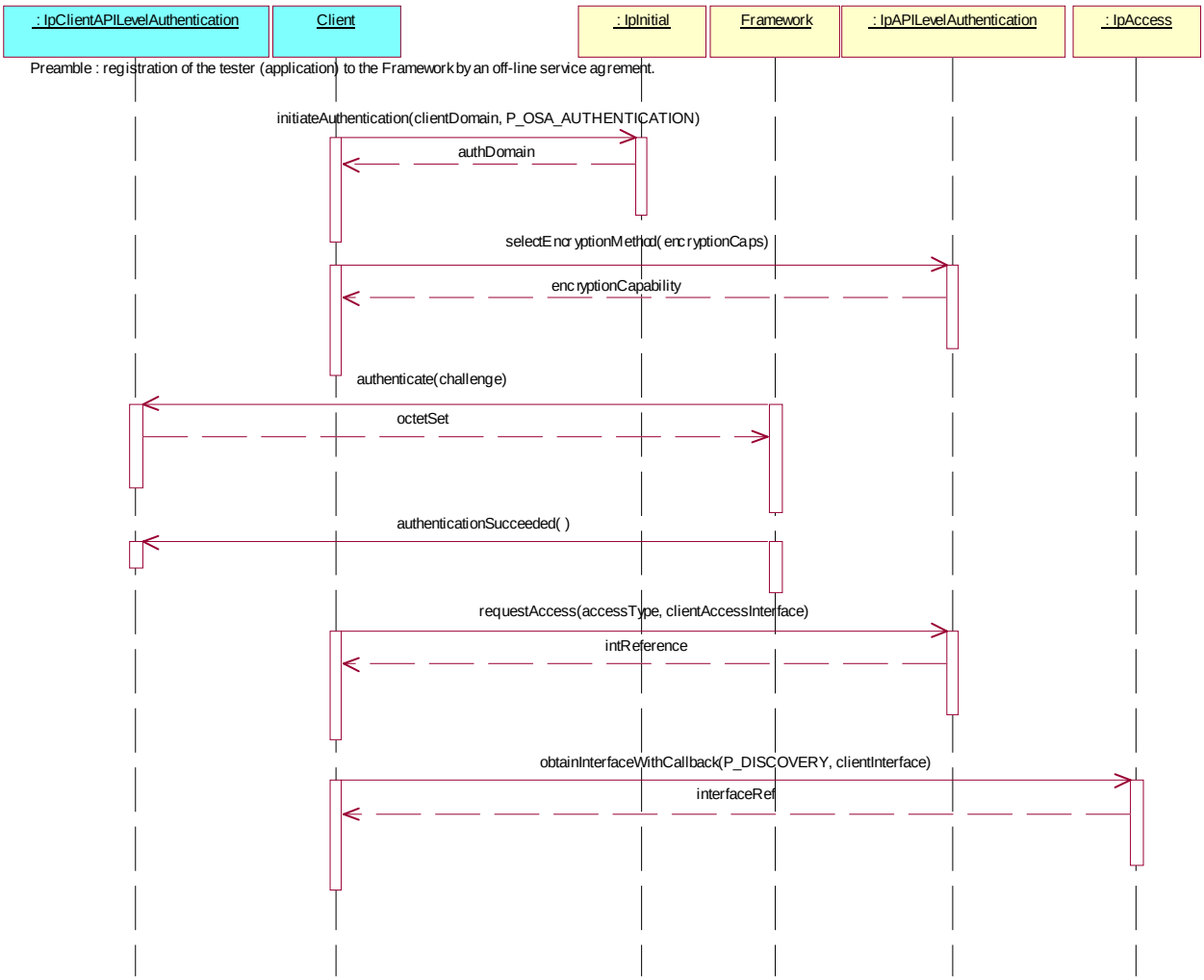
3. Triggered action: cause IUT to call authenticate method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned.

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

6. Method call **obtainInterfaceWithCallback** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_FAULT_MANAGER), clientInterface
   Check: valid value of IpInterfaceRef is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterfaceWithCallback(P_DISCOVERY, clientInterface)
interfaceRef

Test FW_AS_TSM_06

Summary: API level authentication, FW authenticates the client only and receives **abortAuthentication**, unsuccessful

Reference: ES 201 915-3 [1], clauses 6.1.1.2 and 6.1.1.4

Precondition: Authentication required by IUT

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

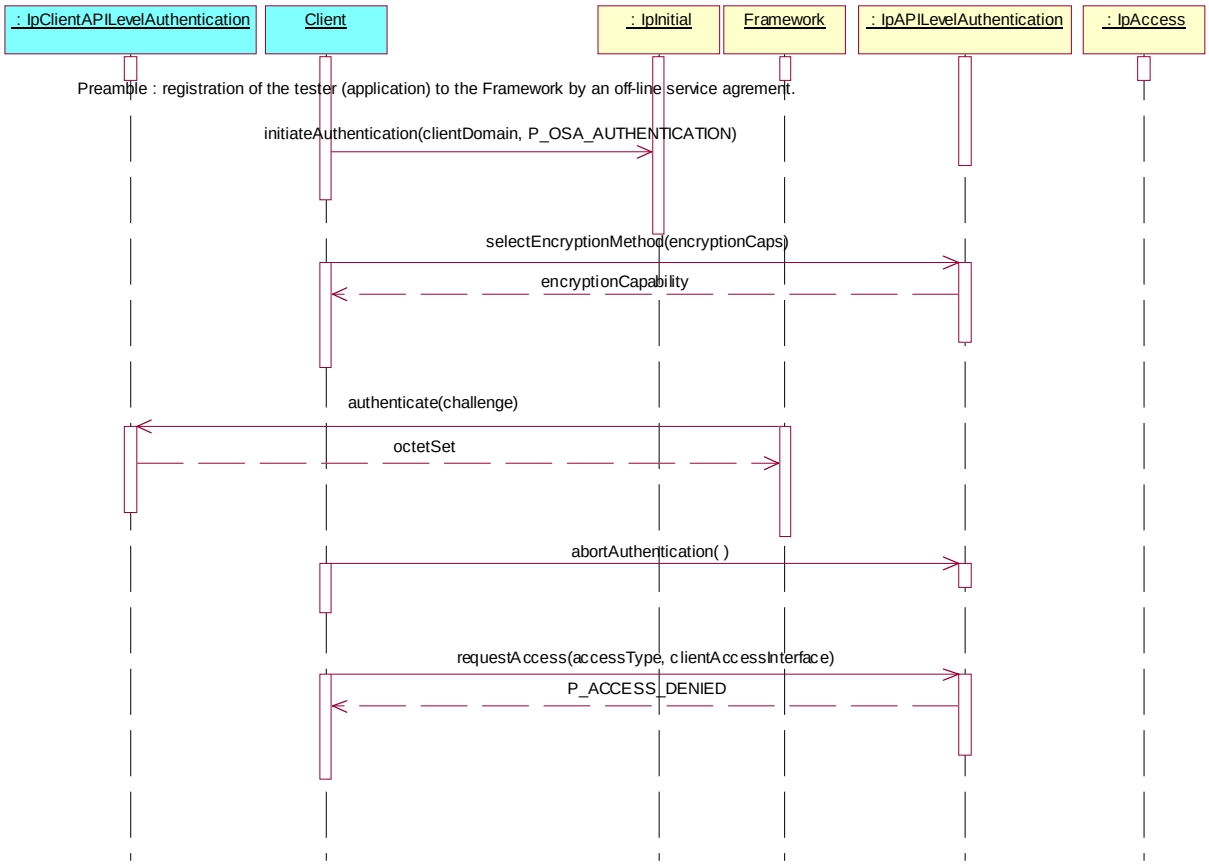
1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

3. Triggered action: cause IUT to call authenticate method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

4. Method call **abortAuthentication()** on **IpAPILevelAuthentication** interface.
   Parameters: none
   Check: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: P_ACCESS_DENIED value is returned



Test FW_AS_TSM_07

Summary: API level authentication, FW authenticates the client only, successful, checks **endAccess** method

Reference: ES 201 915-3 [1], clauses 6.1.1.2 and 6.1.1.4

Precondition: Authentication required by IUT

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

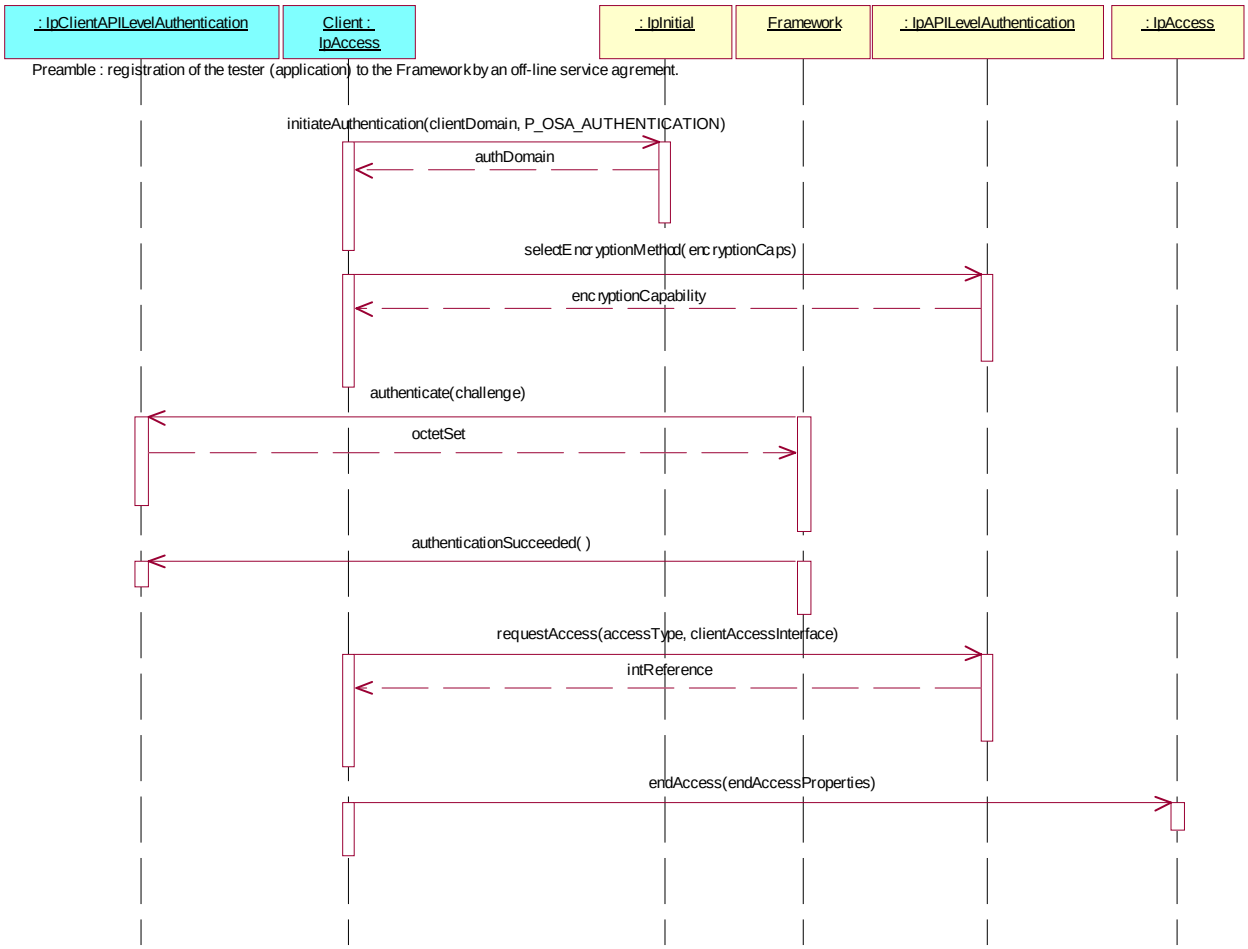
3. Triggered action: cause IUT to call authenticate method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned.

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

6. Method call **endAccess** on **IpAccess** interface.
   Parameters: endAccessProperties
   Check: no exception is returned.


: IpClientAPILevelAuthentication
Client : IpAccess
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
endAccess(endAccessProperties)

Test FW_AS_TSM_08

Summary: API level authentication, FW authenticates the client only, all methods, successful, use of **releaseInterface** method.

Reference: ES 201 915-3 [1], clauses 6.1.1.2 and 6.1.1.4

Precondition: Authentication required by IUT

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

3. Triggered action: cause IUT to call authenticate method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

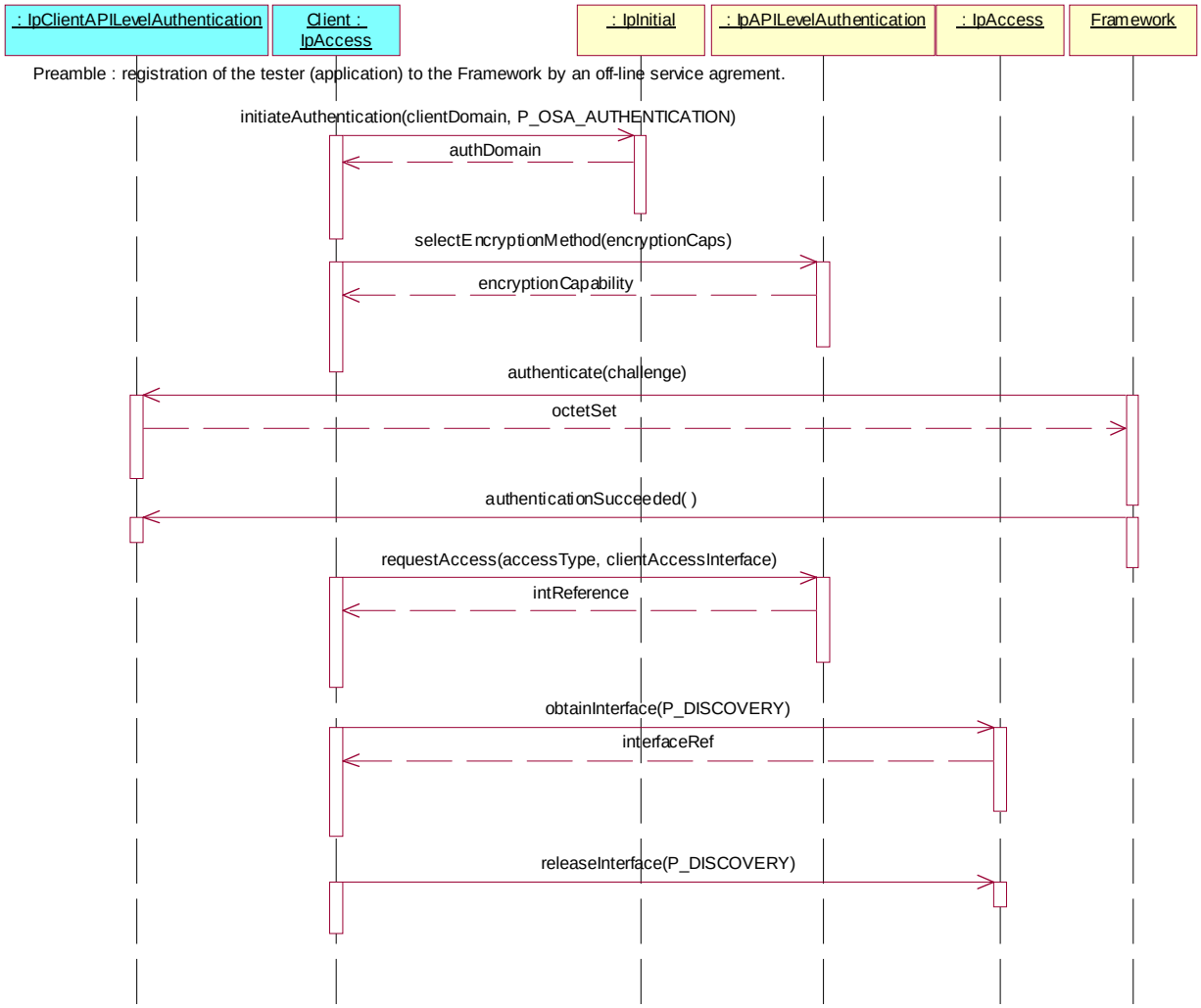
NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned.

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

6. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_DISCOVERY)
   Check: valid value of IpInterfaceRef is returned

7. Method call **releaseInterface** on **IpAccess** interface.
   Parameters: interfaceName (same value as method call nr 6)
   Check: none


: IpClientAPILevelAuthentication
Client : IpAccess
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(P_DISCOVERY)
interfaceRef
releaseInterface(P_DISCOVERY)

Test FW_AS_TSM_09

Summary: Authentication, using Underlying Distribution Technology Mechanism, all methods, successful

Reference: ES 201 915-3 [1], clause 6.1.1.3

Precondition: Underlying authentication supported

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

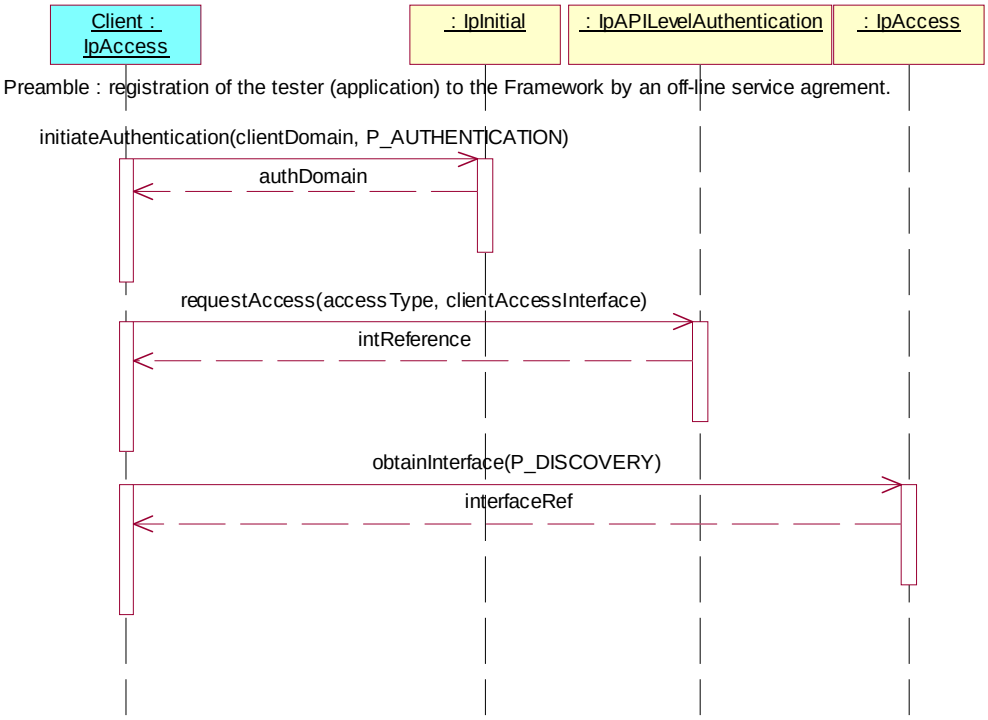
Test Sequence:

1. Perform underlying authentication between tester and IUT.

2. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_ AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

3. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

4. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_DISCOVERY)
   Check: valid value of IpInterfaceRef is returned



5.4.2 Framework to Application API

5.4.2.1 Service Discovery (SD)

Test FW_FA_SD_01

Summary: **IpServiceDiscovery** all methods, successful

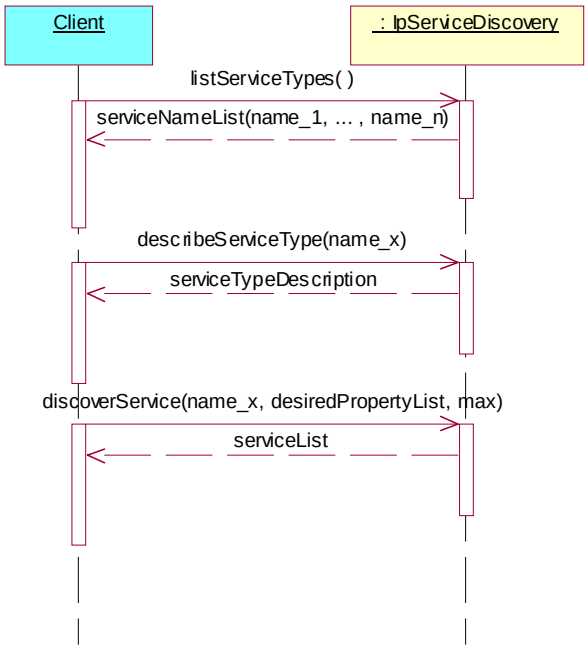
Reference: ES 201 915-3 [1], clause 7.3.1

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **describeServiceType()**
   Parameters: serviceTypeName from the list returned in 1.
   Check: valid value of TpServiceTypeDescription is returned

3. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., valid desiredPropertyList, valid max
   Check: valid value of TpServiceList is returned



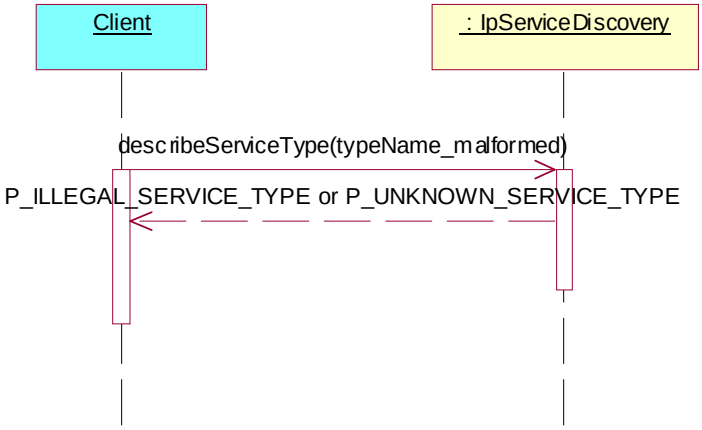
Test FW_FA_SD_02

Summary: **IpServiceDiscovery** describeServiceType, P_ILLEGAL_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 7.3.1

Test Sequence:

1. Method call **describeServiceType()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE exception is returned.



Test FW_FA_SD_03

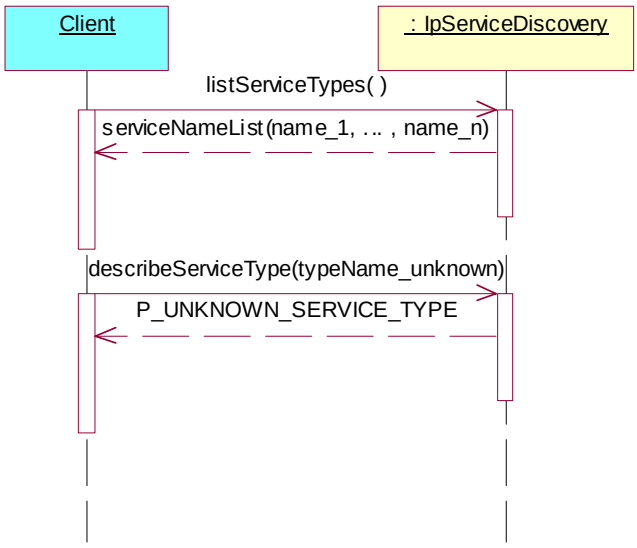
Summary: **IpServiceDiscovery** describeServiceType, P_UNKNOWN_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 7.3.1

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **describeServiceType()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE exception is returned



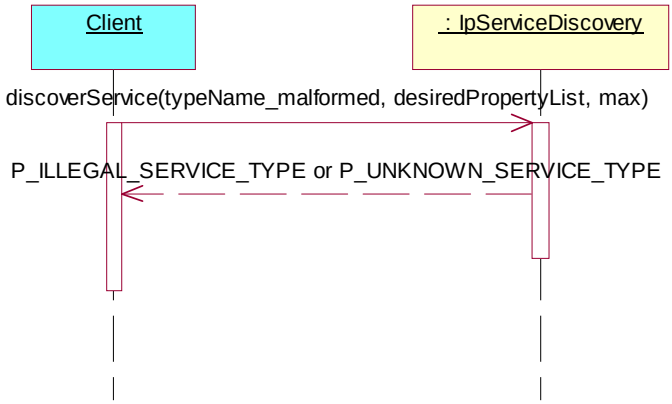
Test FW_FA_SD_04

Summary: **IpServiceDiscovery** discoverService, P_ILLEGAL_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 7.3.1

Test Sequence:

1. Method call **discoverService()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE exception is returned



Test FW_FA_SD_05

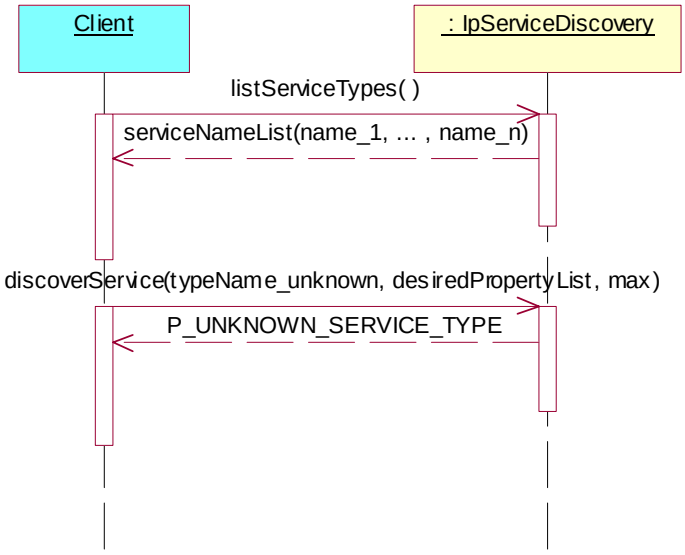
Summary: **IpServiceDiscovery** discoverService, P_UNKNOWN_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 7.3.1

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE exception is returned



Test FW_FA_SD_06

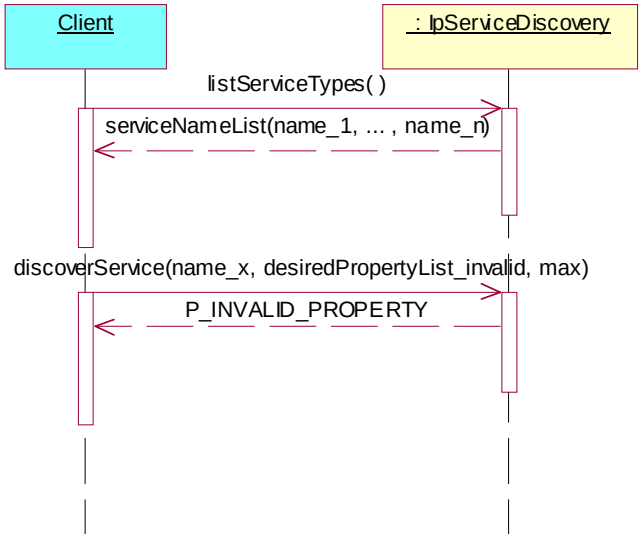
Summary: **IpServiceDiscovery** discoverService, P_INVALID_PROPERTY

Reference: ES 201 915-3 [1], clause 7.3.1

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., invalid desiredPropertyList, valid max
   Check: P_INVALID_PROPERTY exception is returned



Test FW_FA_SD_07

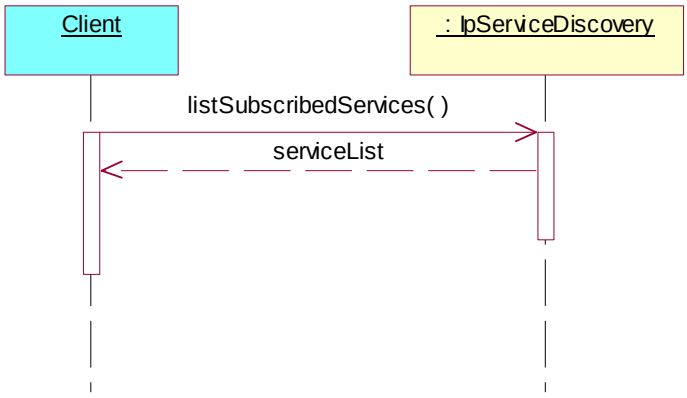
Summary: **IpServiceDiscovery** listSubscribedService

Reference: ES 201 915-3 [1], clause 7.3.1

Precondition: listSubscribedServices supported

Test Sequence:

1. Method call **listSubscribedServices()**
   Parameters: none
   Check: valid value of TpServiceList is returned



5.4.2.2 Service Agreement Management (SA)

Test FW_FA_SA_01

Summary: **IpServiceAgreementManagement**, all methods, successful

Reference: ES 201 915-3 [1], clause 7.3.2

Test Sequence:

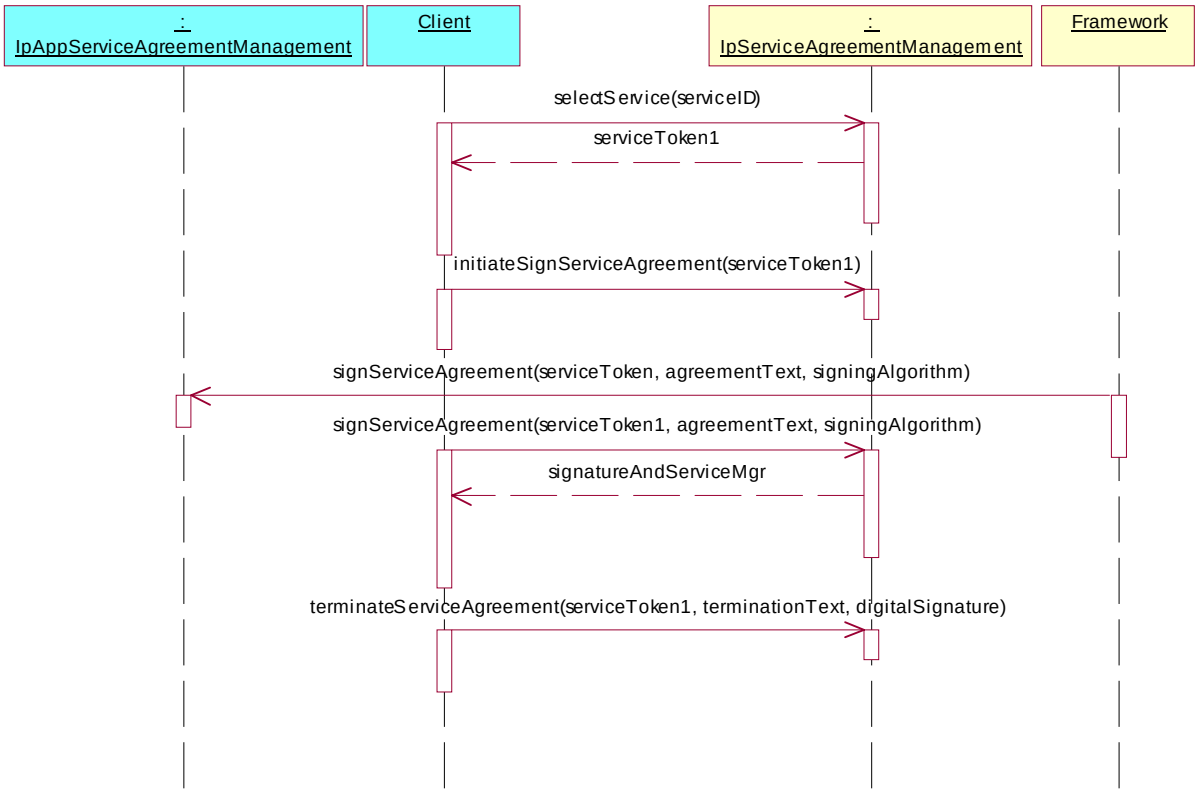
1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement ()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpSignatureAndServiceMgr is returned

5. Method call **terminateServiceAgreement()**
   Parameters: serviceToken returned in 1., terminationText, digitalSignature
   Check: no exception is returned



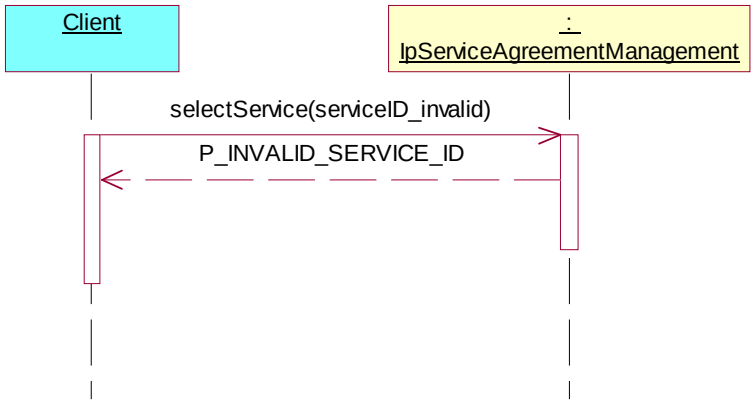
Test FW_FA_SA_02

Summary: **IpServiceAgreementManagement**, selectService, P_INVALID_SERVICE_ID

Reference: ES 201 915-3 [1], clause 7.3.2

Test Sequence:

1. Method call **selectService()**
   Parameters: invalid serviceID
   Check: P_INVALID_SERVICE_ID exception is returned



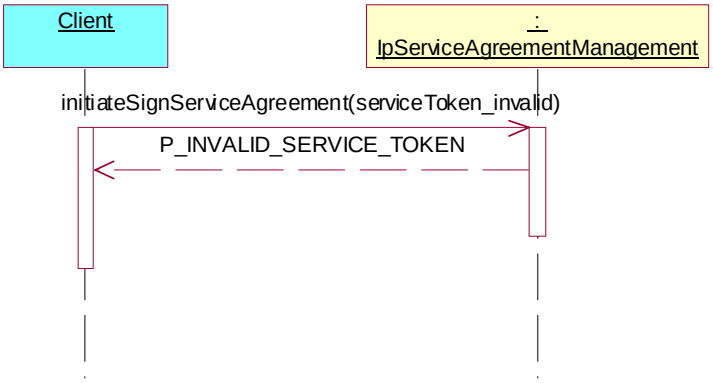
Test FW_FA_SA_03

Summary: **IpServiceAgreementManagement**, initiateSignServiceAgreement, P_INVALID_SERVICE_TOKEN

Reference: ES 201 915-3 [1], clause 7.3.2

Test Sequence:

1. Method call **initiateSignServiceAgreement()**
   Parameters: invalid serviceToken
   Check: P_INVALID_SERVICE_TOKEN exception is returned



Test FW_FA_SA_04

Summary: **IpServiceAgreementManagement**, signServiceAgreement, P_INVALID_SERVICE_TOKEN

Reference: ES 201 915-3 [1], clause 7.3.2

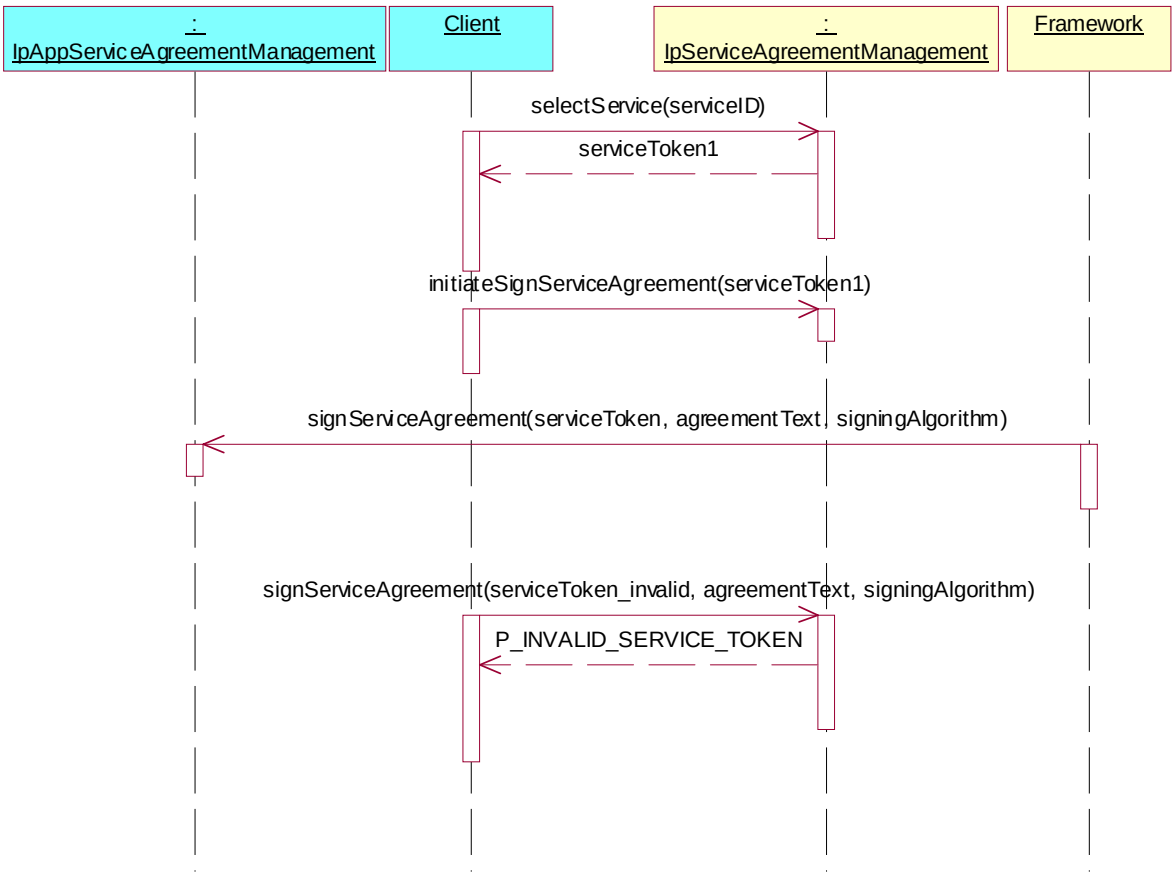
Test Sequence:

1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in step 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: invalid serviceToken, valid agreementText, valid signingAlgorithm
   Check: P_INVALID_SERVICE_TOKEN exception is returned



Test FW_FA_SA_05

Summary: **IpServiceAgreementManagement**, signServiceAgreement, P_INVALID_AGREEMENT_TEXT

Reference: ES 201 915-3 [1], clause 7.3.2

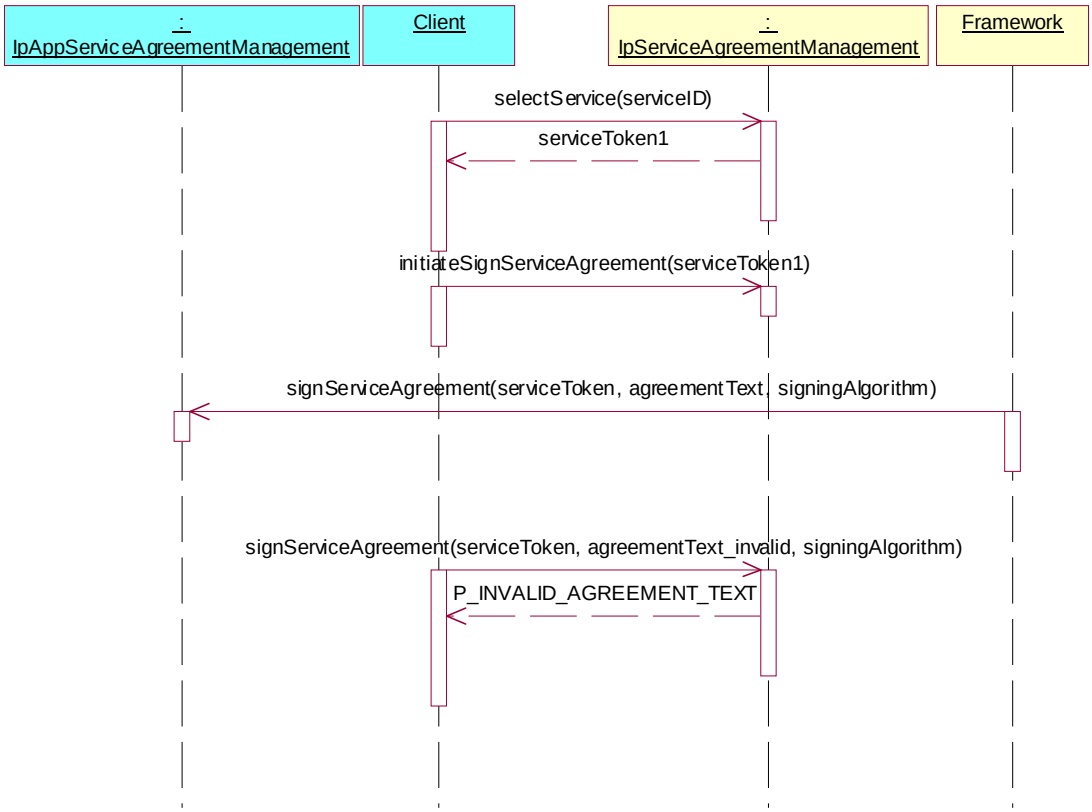
Test Sequence:

1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: TpServiceToken returned in step 1.
   Check: No exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken

4. Method call **signServiceAgreement()**
   Parameters: serviceToken, invalid agreementText, signingAlgorithm
   Check: P_INVALID_AGREEMENT_TEXT exception is returned



Test FW_FA_SA_06

Summary: **IpServiceAgreementManagement**, signServiceAgreement, P_INVALID_SIGNING_ALGORITHM

Reference: ES 201 915-3 [1], clause 7.3.2

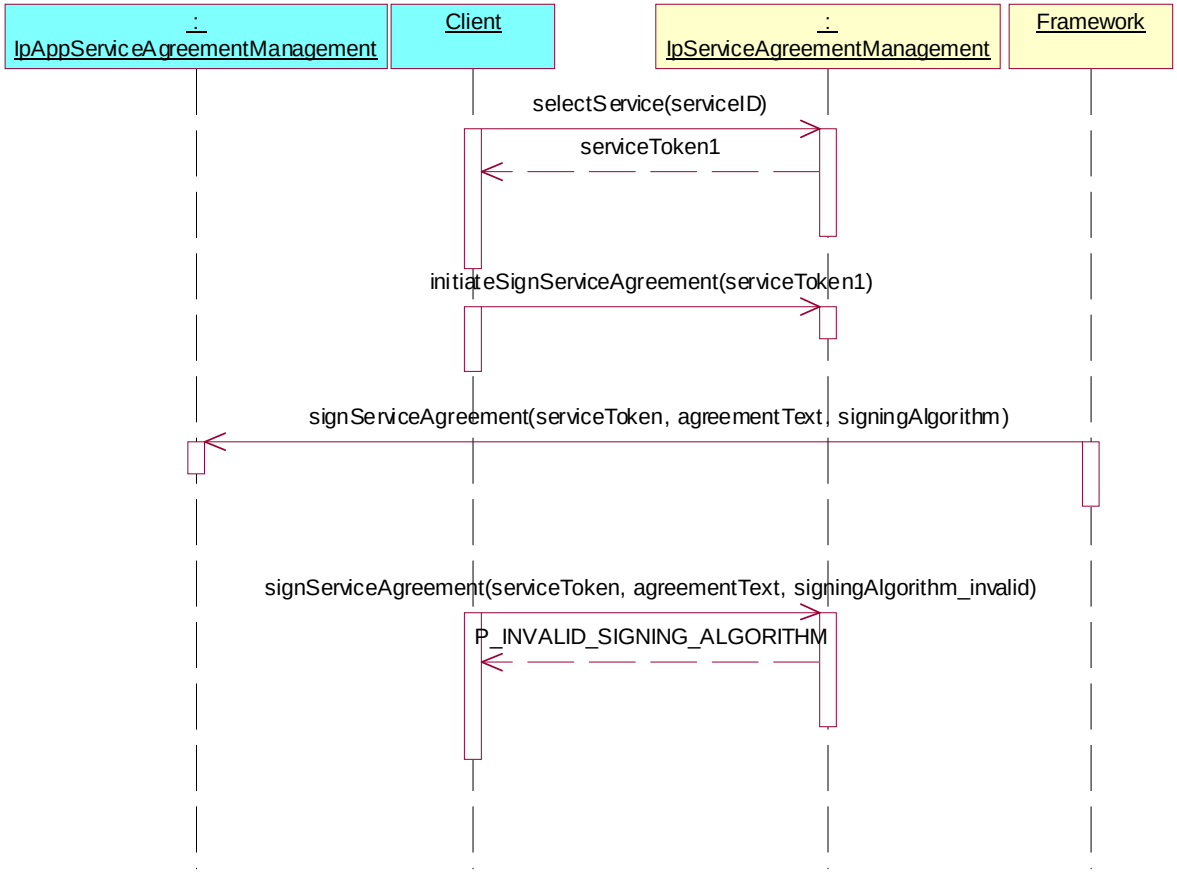
Test Sequence:

1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in step 1.
   Check: No exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement ()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken

4. Method call **signServiceAgreement()**
   Parameters: serviceToken, agreementText, invalid signingAlgorithm
   Check: P_INVALID_SIGNING_ALGORITHM exception is returned



Test FW_FA_SA_07

Summary: **IpServiceAgreementManagement**, terminateServiceAgreement, P_INVALID_SERVICE_TOKEN

Reference: ES 201 915-3 [1], clause 7.3.2

Test Sequence:

1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpSignatureAndServiceMgr is returned

5. Method call **terminateServiceAgreement()**
   Parameters: invalid serviceToken, terminationText, digitalSignature
   Check: P_INVALID_SERVICE_TOKEN exception is returned

Test FW_FA_SA_08

Summary: **IpServiceAgreementManagement**, terminateServiceAgreement, P_INVALID_SIGNATURE

Reference: ES 201 915-3 [1], clause 7.3.2

Test Sequence:

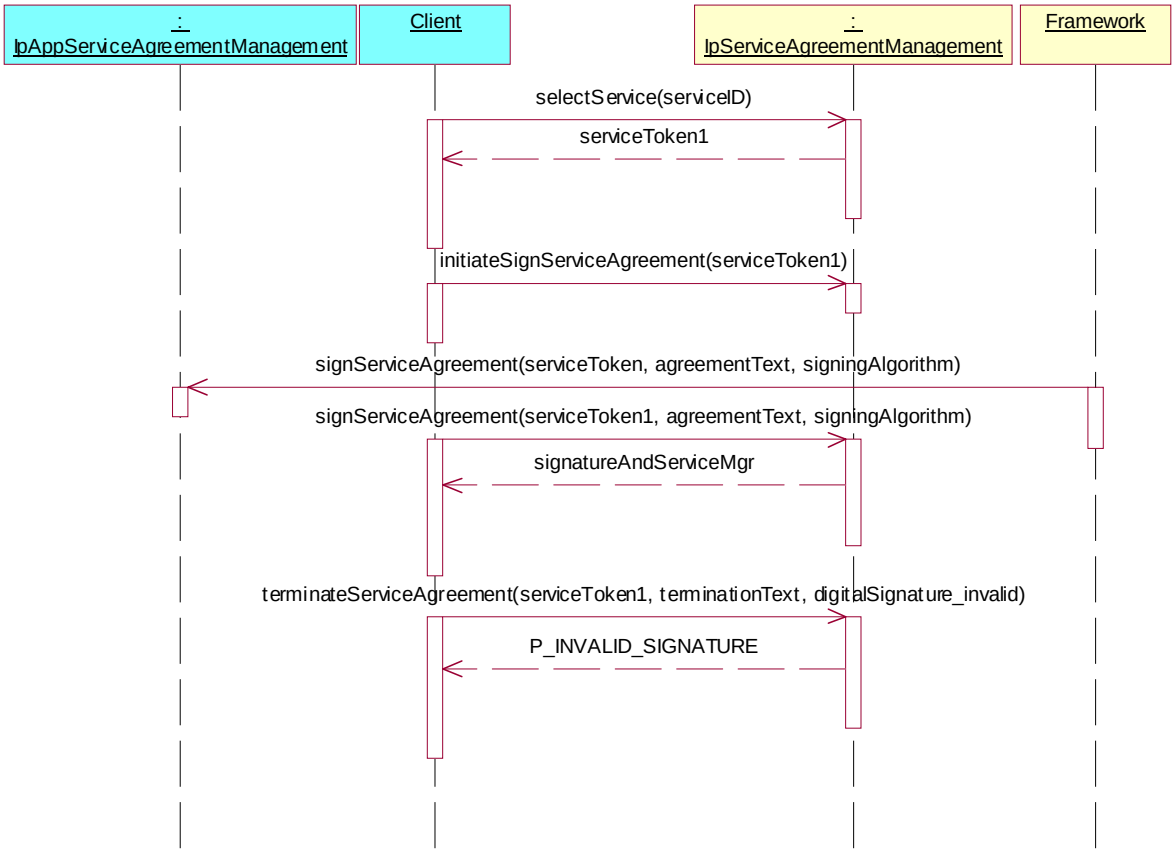
1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpSignatureAndServiceMgr is returned

5. Method call **terminateServiceAgreement()**
   Parameters: serviceToken returned in 1., terminationText, invalid digitalSignature
   Check: P_INVALID_SIGNATURE exception is returned



5.4.2.3 Integrity Management (IM)

Test FW_FA_IM_01

Summary: **IpHeartBeatMgmt**, all methods, successful

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpHeartBeatMgmt supported

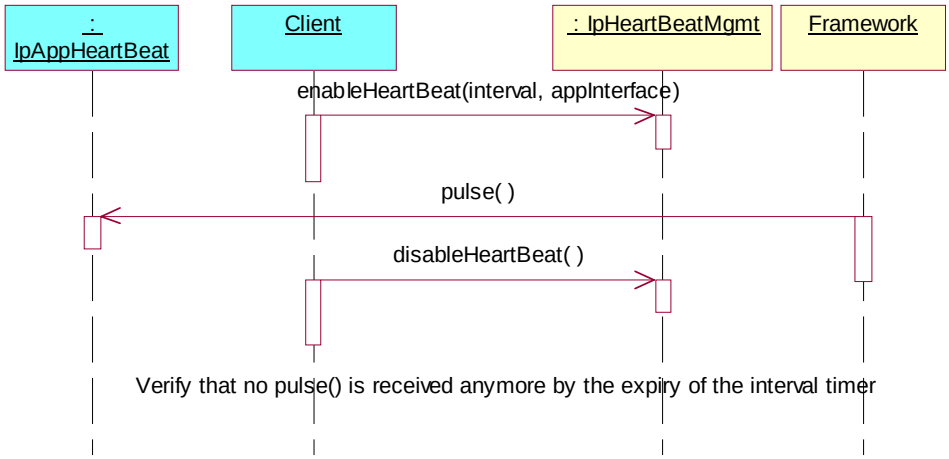
Preamble: The calling application must have a callback interface and a reference to this interface.

Test Sequence:

1. Method call **enableHeartBeat()**
   Parameters: interval, appInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to regularly call pulse() method on the tester's (Application) **IpAppHeartBeat** interface.
   Parameters: none
   Check: no exception is returned. Check also that the pulse() method is invoked at the requested interval.

3. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.



Test FW_FA_IM_02

Summary: **IpHeartBeatMgmt**, all methods, successful

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpHeartBeatMgmt, changeInterval supported

Preamble: The calling application must have a callback interface and a reference to this interface.

Test Sequence:

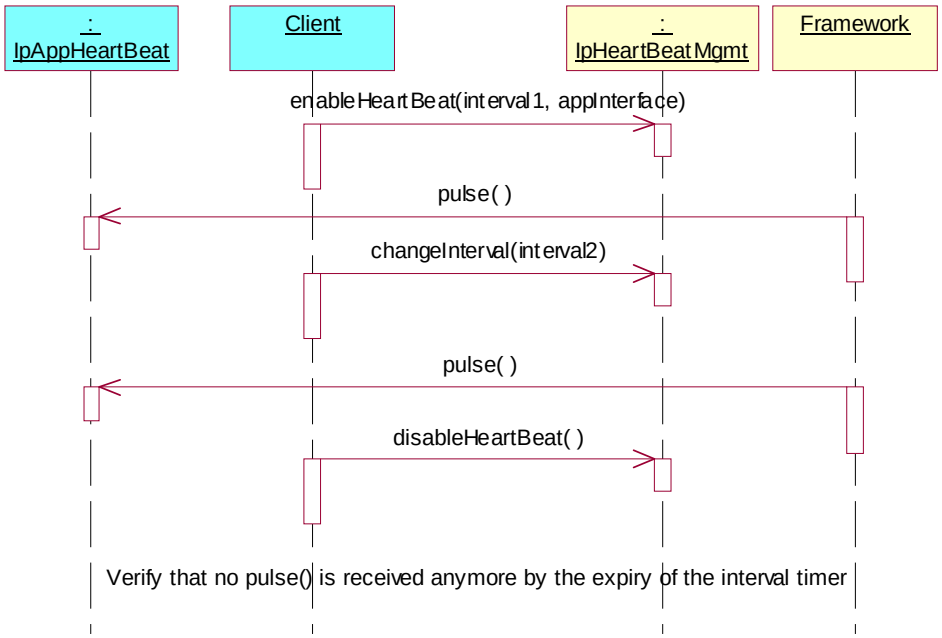
1. Method call **enableHeartBeat()**
   Parameters: interval, appInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to call pulse() method regularly on the tester's (Application) **IpAppHeartBeat** interface.
   Parameters: none
   Check: no exception is returned. Check also that the pulse() method is invoked at the requested interval.

3. Method call **changeInterval()**
   Parameters: interval
   Check: no exception is returned.

4. Triggered action: cause IUT to call pulse() method regularly on the tester's (Application) **IpAppHeartBeat** interface.
   Parameters: none
   Check: the pulse() method is invoked at the new requested interval.

5. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.



Test FW_FA_IM_03

Summary: **IpHeartBeat**, all methods, successful

Reference: ES 201 915-3 [1], clause 7.3.3

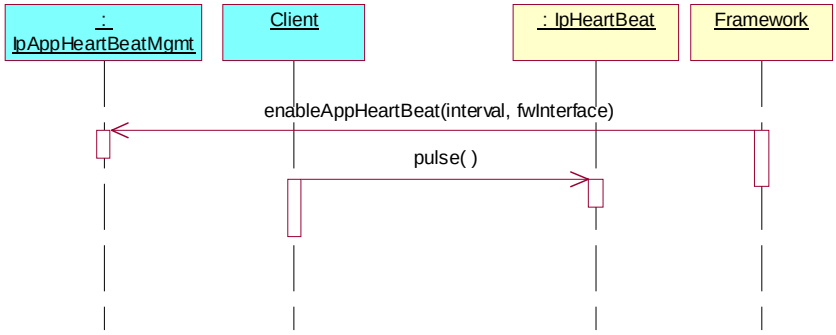
Precondition: IpHeartBeat is supported

Preamble: The calling application must have a callback interface and a reference to this interface.

Test Sequence:

1. Triggered action: cause IUT to call **enableHeartBeat()** method on the tester's (Application) **IpAppHeartBeatMgmt** interface.
   Parameters: interval, fwInterface

2. Method call **pulse()**
   Parameters: none
   Check: no exception



Test FW_FA_IM_04

Summary: **IpFaultManager** activityTestReq, successful

Reference: ES 201 915-3 [1], clause 7.3.3

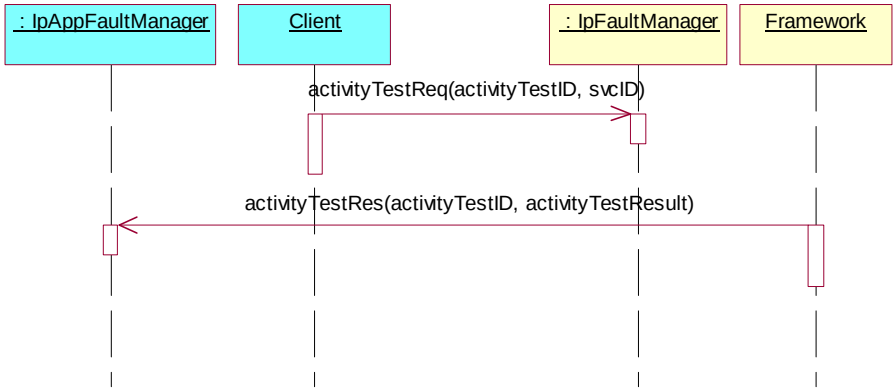
Precondition: IpFaultManager, activitiyTestReq supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: activityTestID, svcID
   Check: no exception is returned

2. Triggered action: cause IUT to call **activityTestRes ()** method on the tester's (Application) IpAppFaultManager interface.
   Parameters: activityTestID, activityTestResult



Test FW_FA_IM_05

Summary: **IpFaultManager** activityTestReq on Framework, successful

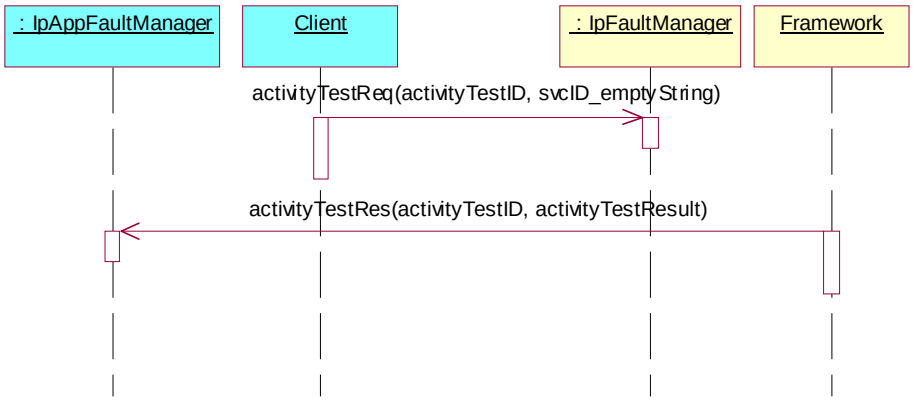
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, activitiyTestReq supported

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: **activityTestID**, svcID with emptystring value
   Check: no exception is returned

2. Triggered action: cause IUT to call **activityTestRes()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID, activityTestResult



Test FW_FA_IM_06

Summary: **IpFaultManager** activityTestReq , P_INVALID_SERVICE_ID exception

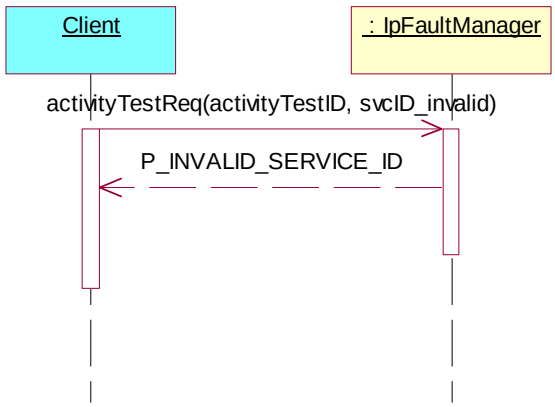
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, activitiyTestReq supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: activityTestID, invalid svcID
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_07

Summary: **IpFaultManager** genFaultStatsRecordReq, successful

Reference: ES 201 915-3 [1], clause 7.3.3

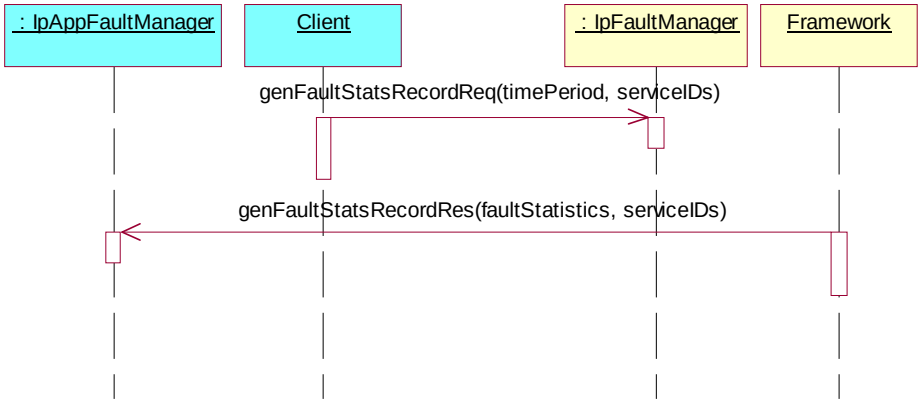
Precondition: IpFaultManager, **getFaultStatsRecordReq()** supported

Preamble: he application must have registered at least one service.

Test Sequence:

1. Method call **genFaultStatsRecordReq()**
   Parameters: timePeriod, serviceIDs
   Check: no exception is returned

2. Triggered action: cause IUT to call **genFaultStatsRecordRes()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: faultStatistics, ServiceIDs



Test FW_FA_IM_08

Summary: **IpFaultManager** genFaultStatsRecordReq on Framework, successful

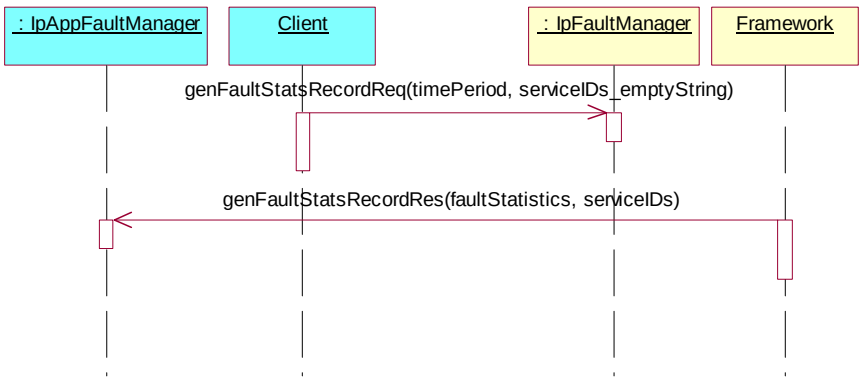
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, **getFaultStatsRecordReq()** supported

Test Sequence:

1. Method call **genFaultStatsRecordReq()**
   Parameters: timePeriod, serviceIDs with emptystring value
   Check: no exception is returned

2. Triggered action: cause IUT to call **genFaultStatsRecordRes()** method on the tester's (Application) IpAppFaultManager interface.
   Parameters: faultStatistics,ServiceIDs



Test FW_FA_IM_09

Summary: **IpFaultManager** genFaultStatsRecordReq, P_INVALID_SERVICE_ID exception

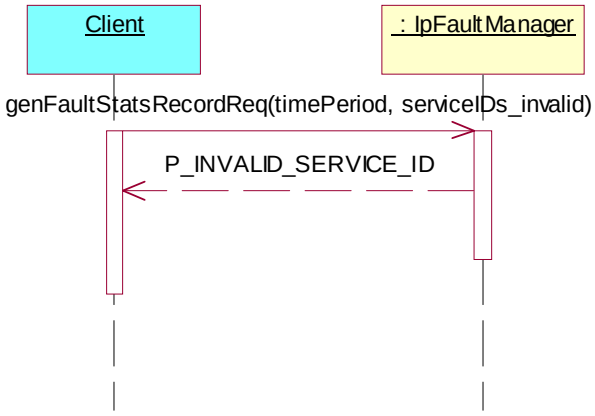
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, **getFaultStatsRecordReq()** supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **genFaultStatsRecordReq()**
   Parameters: timePeriod, invalid serviceIDs
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_10

Summary: **IpFaultManager** svcUnavailableInd, successful

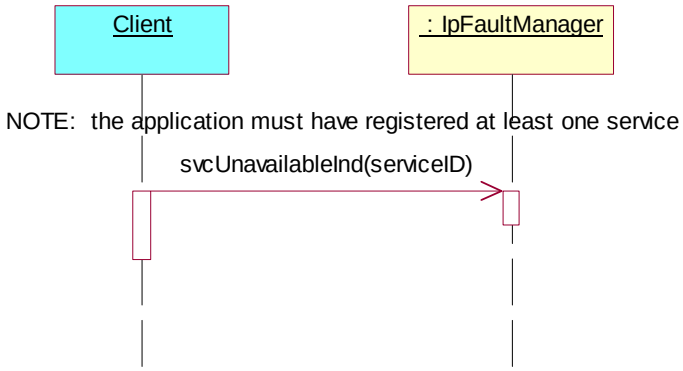
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, **svcUnavailableInd()** supported

Preamble: he application must have registered at least one service.

Test Sequence:

1. Method call **svcUnavailableInd()**
   Parameters: serviceID
   Check: no exception is returned



Test FW_FA_IM_11

Summary: **IpFaultManager** svcUnavailableInd, P_INVALID_SERVICE_ID exception

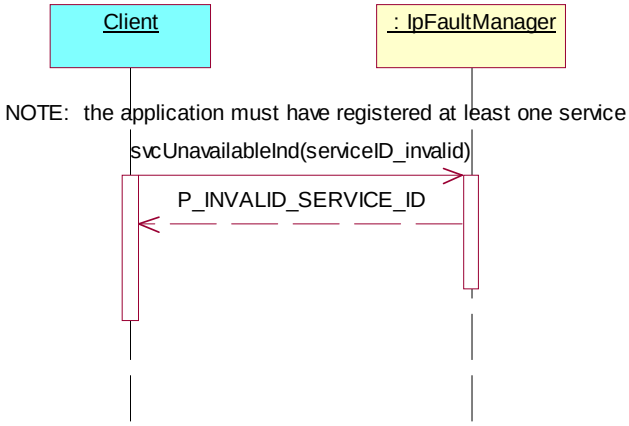
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, **svcUnavailableInd()** supported

Preamble: The application must have register at least one service.

Test Sequence:

1. Method call **svcUnavailableInd()**
   Parameters: invalid serviceID
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_12

Summary: **IpFaultManager** appActivityTestRes, successful

Reference: ES 201 915-3 [1], clause 7.3.3

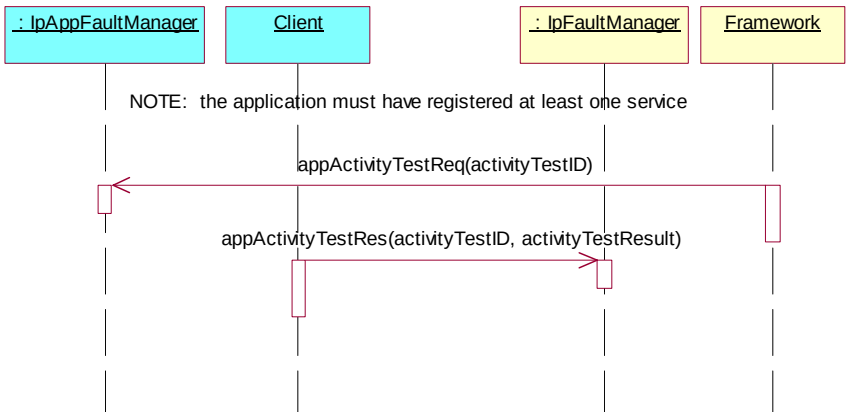
Precondition: IpFaultManager, **appActivityTestRes()** supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID

2. Method call **appActivityTestRes()**
   Parameters: activityTestID, activityTestResult
   Check: no exception is returned



Test FW_FA_IM_13

Summary: **IpFaultManager** appActivityTestRes, P_INVALID_ACTIVITY_TEST_ID Exception

Reference: ES 201 915-3 [1], clause 7.3.3

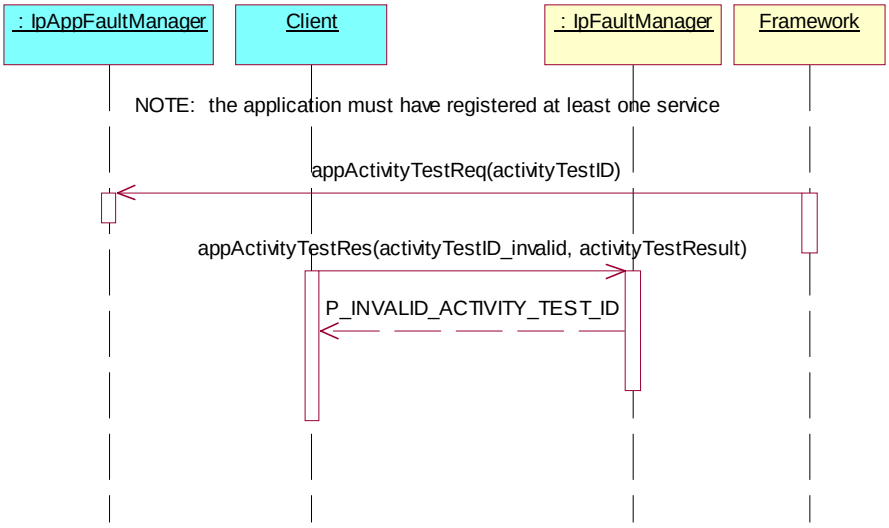
Precondition: IpFaultManager, **appActivityTestRes()** supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID

2. Method call **appActivityTestRes()**
   Parameters: invalid activityTestID, activityTestResult
   Check: P_INVALID_ACTIVITY_TEST_ID exception is returned



Test FW_FA_IM_14

Summary: **IpFaultManager** appActivityTestErr, successful

Reference: ES 201 915-3 [1], clause 7.3.3

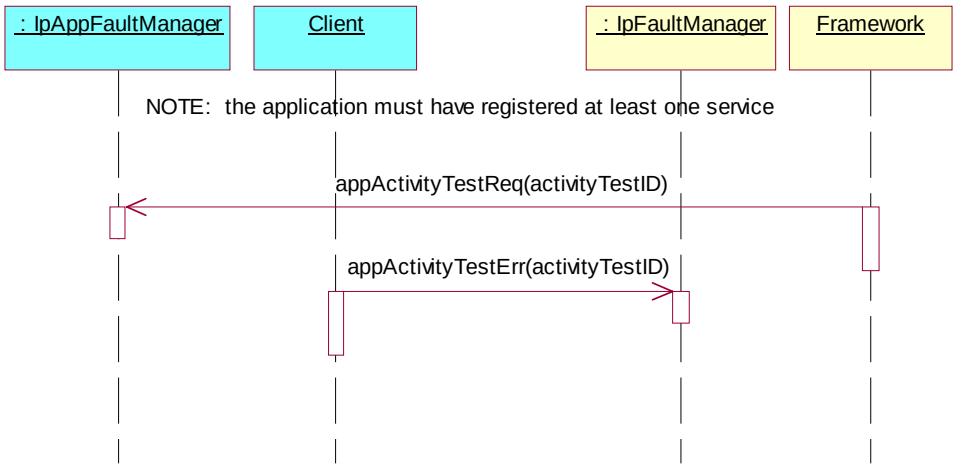
Precondition: IpFaultManager, **appActivityTestRes()** supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID

2. Method call **appActivityTestErr()**
   Parameters: activityTestID
   Check: no exception is returned



Test FW_FA_IM_15

Summary: **IpFaultManager** appActivityTestErr, P_INVALID_ACTIVITY_TEST_ID exception

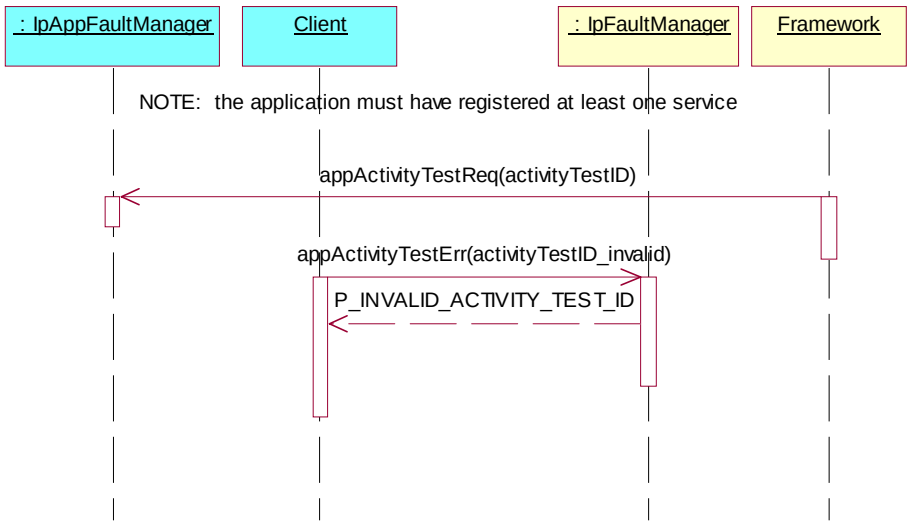
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, **appActivityTestRes()** supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID
2. Method call **appActivityTestErr()**
   Parameters: invalid activityTestID
   Check: P_INVALID_ACTIVITY_TEST_ID exception is returned



Test FW_FA_IM_16

Summary: **IpFaultManager** appUnavailableInd, successful

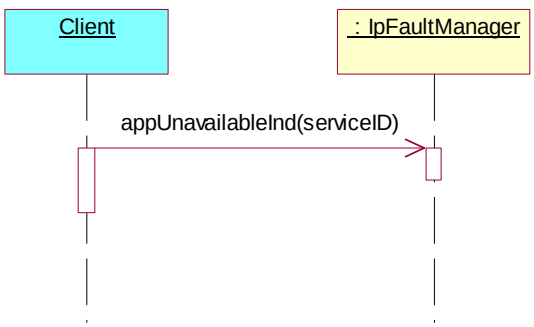
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpFaultManager, **appUnavailableInd()** supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **appUnavailableInd()**
   Parameters: serviceID
   Check: no exception is returned



Test FW_FA_IM_17

Summary: **IpLoadManager** createLoadLevelNotification and destroyLoadLevelNotification methods, successful

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpLoadManager, reateLoadLevelNotification and destroyLoadLevelNotification supported

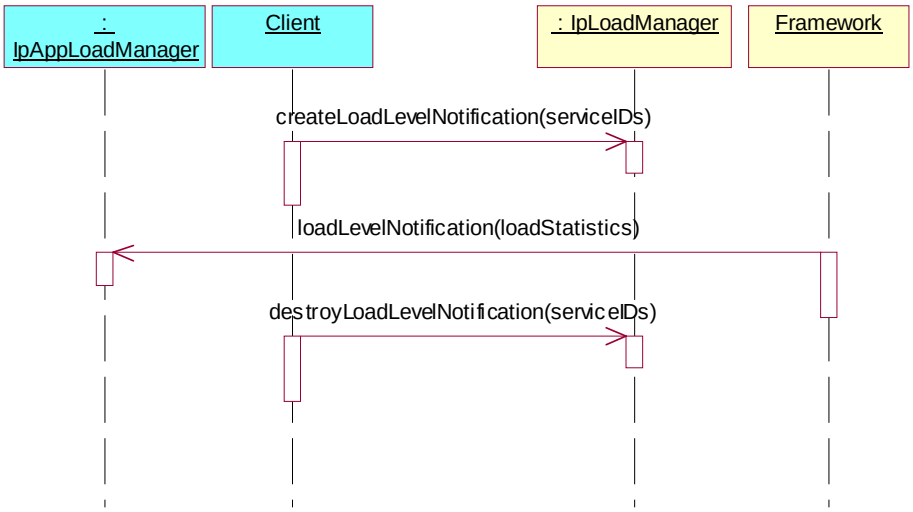
Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned

2. Triggered action: cause IUT to call **loadLevelNotification()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatistics

3. Method call **destroyLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned



Test FW_FA_IM_18

Summary: **IpLoadManager** All methods, successful

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpLoadManager, notifications with suspendNotification and resumeNotification supported.

Preamble: The application must have registered at least one service.

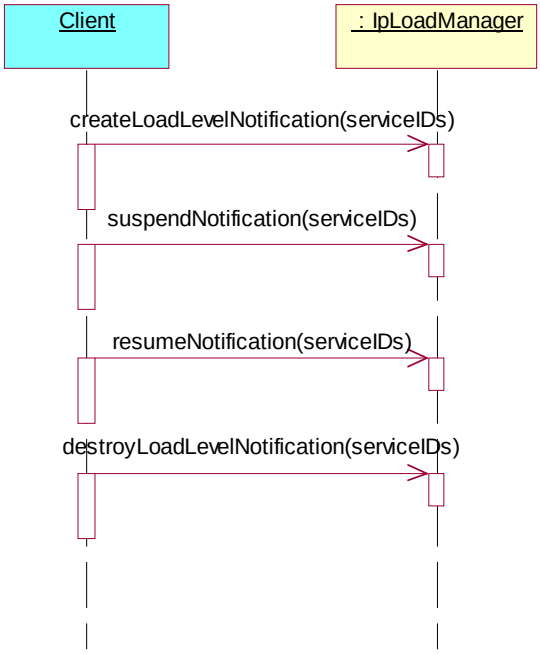
Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned

2. Method call **suspendNotification()**
   Parameters: serviceIDs
   Check: no exception is returned

3. Method call **resumeNotification()**
   Parameters: serviceIDs
   Check: no exception is returned

4. Method call **destroyLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned



Test FW_FA_IM_19

Summary: **IpLoadManager** All methods on Framework, successful

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpLoadManager, notifications with suspendNotification and resumeNotification supported

Preamble: The application must have registered at least one service.

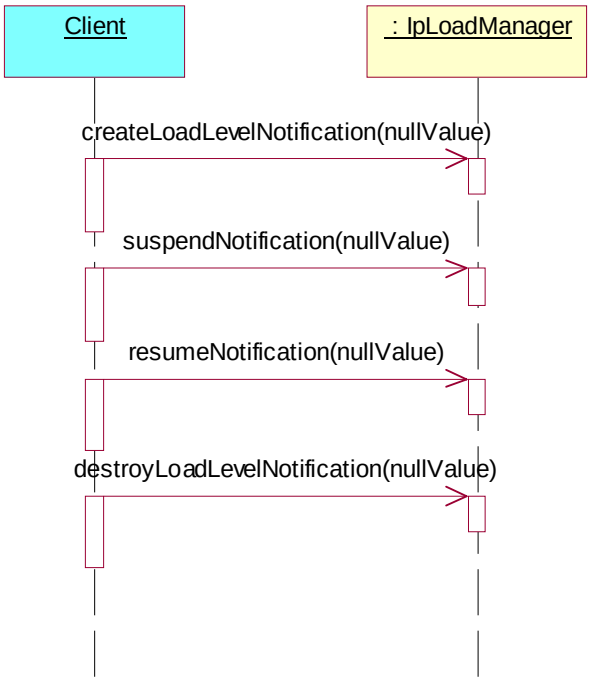
Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **suspendNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

3. Method call **resumeNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

4. Method call **destroyLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned



Test FW_FA_IM_20

Summary: **IpLoadManager** createLoadLevelNotification, P_INVALID_SERVICE_ID

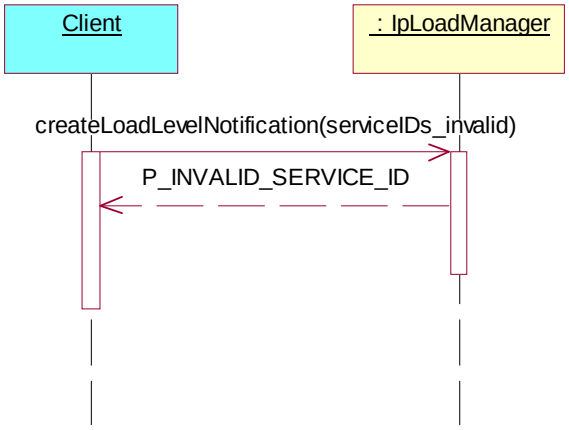
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpLoadManager, createLoadLevelNotification supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_21

Summary: **IpLoadManager** suspendNotification, P_INVALID_SERVICE_ID

Reference: ES 201 915-3 [1], clause 7.3.3

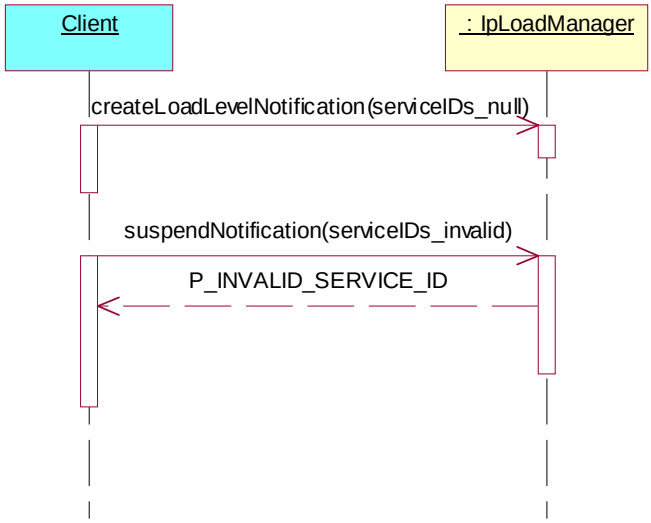
Precondition: IpLoadManager, notifications with suspendNotification supported

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **suspendNotification ()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_22

Summary: **IpLoadManager** resumeNotification, P_INVALID_SERVICE_ID

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpLoadManager, notifications with suspendNotification and resumeNotification supported.

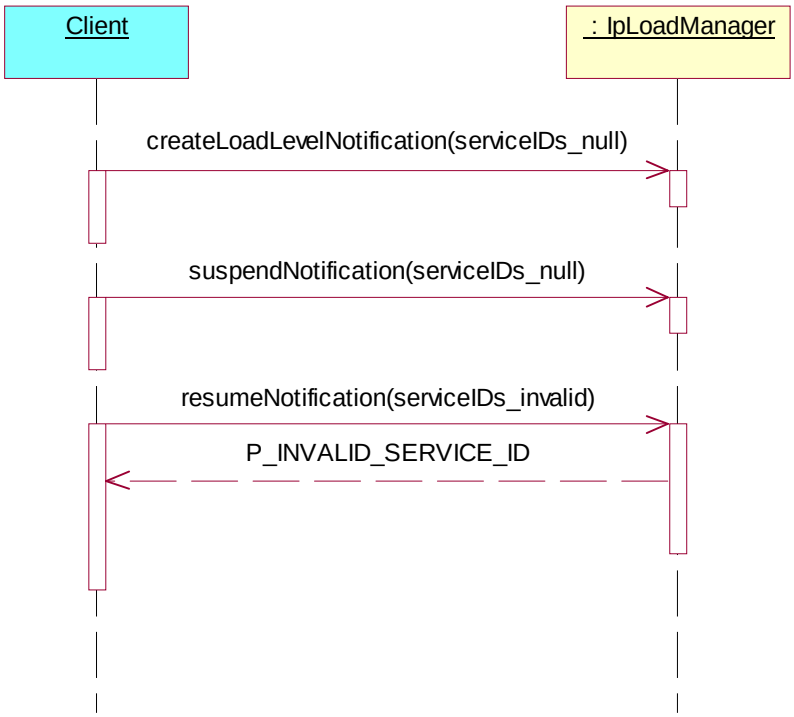
Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **IpClientAPILevelAuthentication ()**
   Parameters: serviceIDs is null
   Check: no exception is returned

3. Method call **resumeNotification ()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_23

Summary: **IpLoadManager** destroyLoadLevelNotification, P_INVALID_SERVICE_ID

Reference: ES 201 915-3 [1], clause 7.3.3

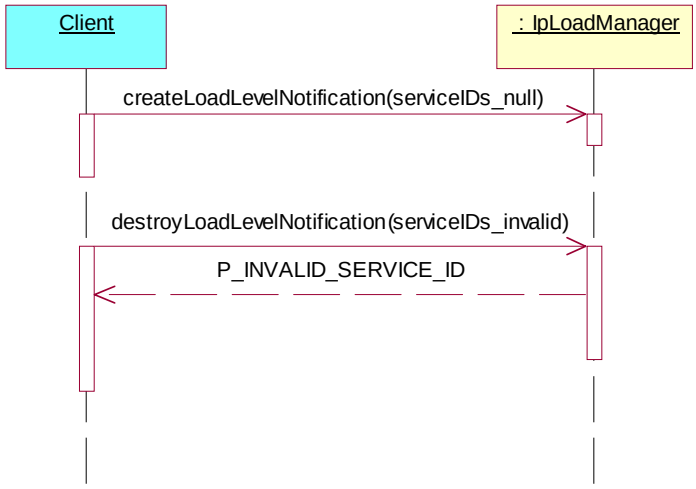
Precondition: IpLoadManager, notifications supported.

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **destroyLoadLevelNotification ()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_24

Summary: **IpLoadManager** reportLoad, successful

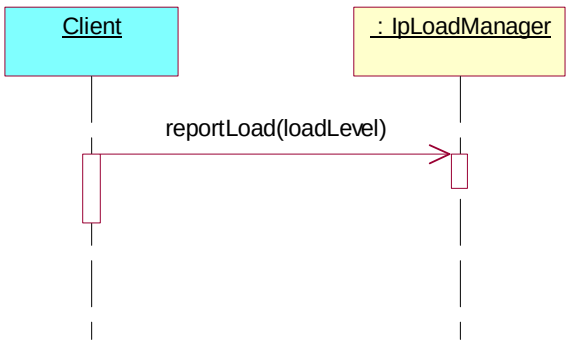
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpLoadManager, reportLoad supported.

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **reportLoad ()**
   Parameters: loadLevel
   Check: no exception is returned



Test FW_FA_IM_25

Summary: **IpLoadManager** queryLoadReq, successful

Reference: ES 201 915-3 [1], clause 7.3.3

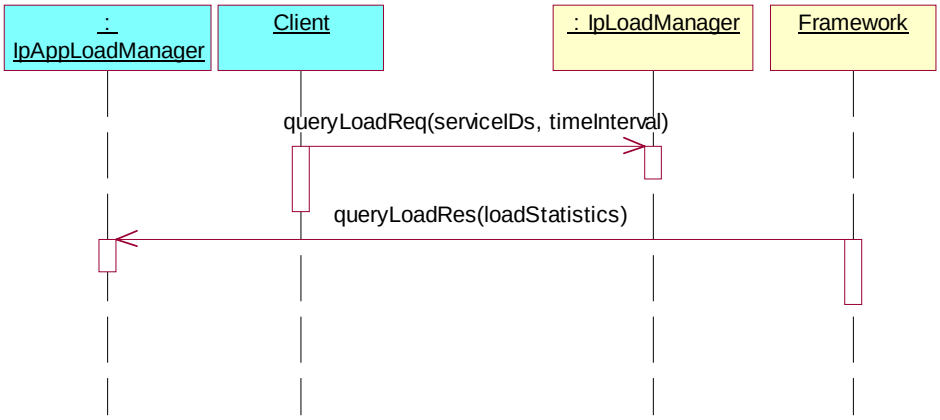
Precondition: IpLoadManager, queryLoadReq supported.

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **queryLoadReq()**
   Parameters: serviceIDs, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **queryLoadRes ()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatistics



Test FW_FA_IM_26

Summary: **IpLoadManager** queryLoadReq on Framework, successful

Reference: ES 201 915-3 [1], clause 7.3.3

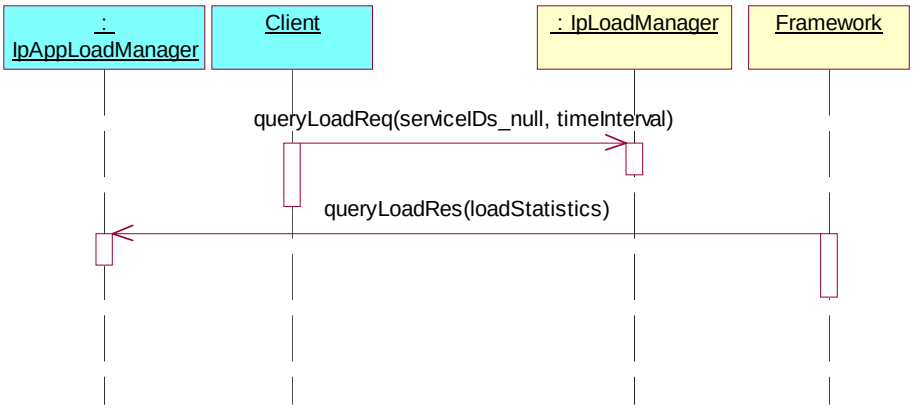
Precondition: IpLoadManager, queryLoadReq supported.

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **queryLoadReq()**
   Parameters: serviceIDs as null, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **queryLoadRes ()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatistics



Test FW_FA_IM_27

Summary: **IpLoadManager** queryLoadReq, P_INVALID_SERVICE_ID

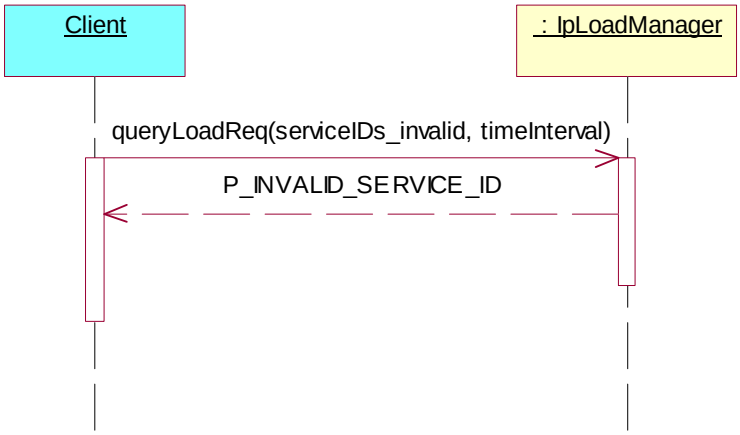
Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpLoadManager, queryLoadReq supported.

Preamble: The application must have registered at least one service.

Test Sequence:

1. Method call **queryLoadReq()**
   Parameters: invalid serviceIDs, timeInterval
   Check: P_INVALID_SERVICE_ID exception is returned



Test FW_FA_IM_28

Summary: **IpLoadManager** queryAppLoadRes, successful

Reference: ES 201 915-3 [1], clause 7.3.3

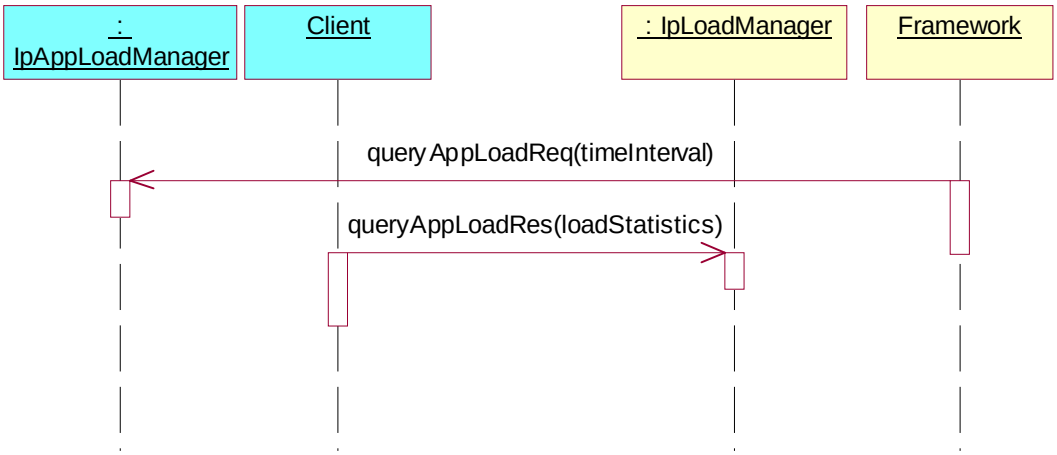
Precondition: IpLoadManager, queryAppLoadRes supported.

Preamble: The application must have registered at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **queryAppLoadReq ()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: timeInterval

2. Method call **queryAppLoadRes ()**
   Parameters: loadStatistics
   Check: no exception is returned



Test FW_FA_IM_29

Summary: **IpLoadManager** queryAppLoadErr, successful

Reference: ES 201 915-3 [1], clause 7.3.3

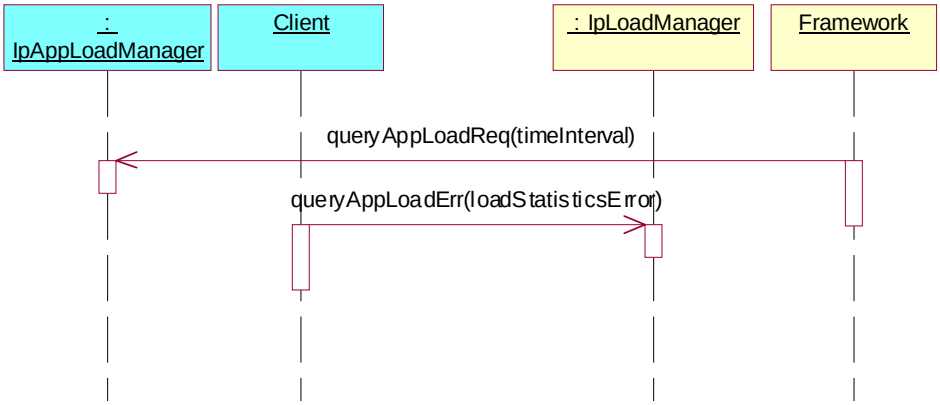
Precondition: IpLoadManager, queryAppLoadRes supported.

Preamble: The application must have registered at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **queryAppLoadReq()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: timeInterval

2. Method call **queryAppLoadErr()**
   Parameters: loadStatisticsError
   Check: no exception is returned



Test FW_FA_IM_30

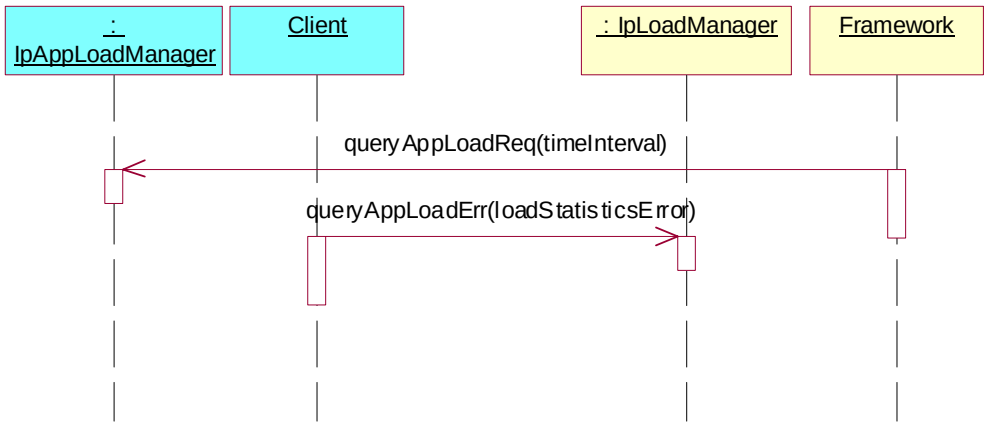
Summary: **IpOAM**, systemDateTimeQuery, successful

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpOAM supported

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: clientDateAndTime
   Check: valid value of TpDateAndTime is returned



Test FW_FA_IM_31

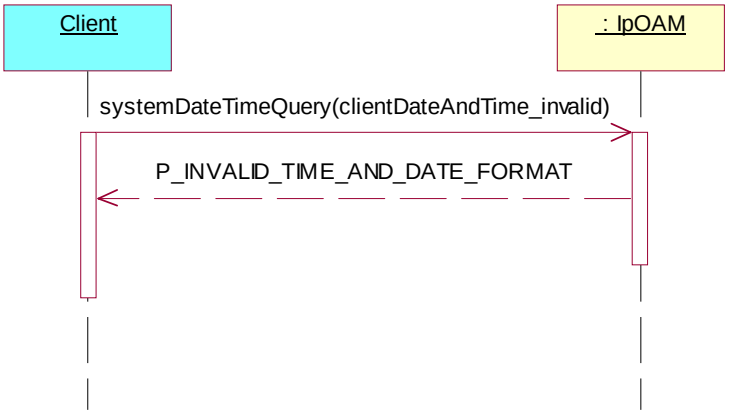
Summary: **IpOAM**, systemDateTimeQuery, P_INVALID_TIME_AND-DATE_FORMAT exception

Reference: ES 201 915-3 [1], clause 7.3.3

Precondition: IpOAM supported

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: invalid clientDateAndTime
   Check: P_INVALID_TIME_AND-DATE_FORMAT is returned



5.4.2.4 Event Notification (EN)

Test FW_FA_EN_01

Summary: **IpEventNotification**, create and destroy methods, successful

Reference: ES 201 915-3 [1], clause 7.3.4

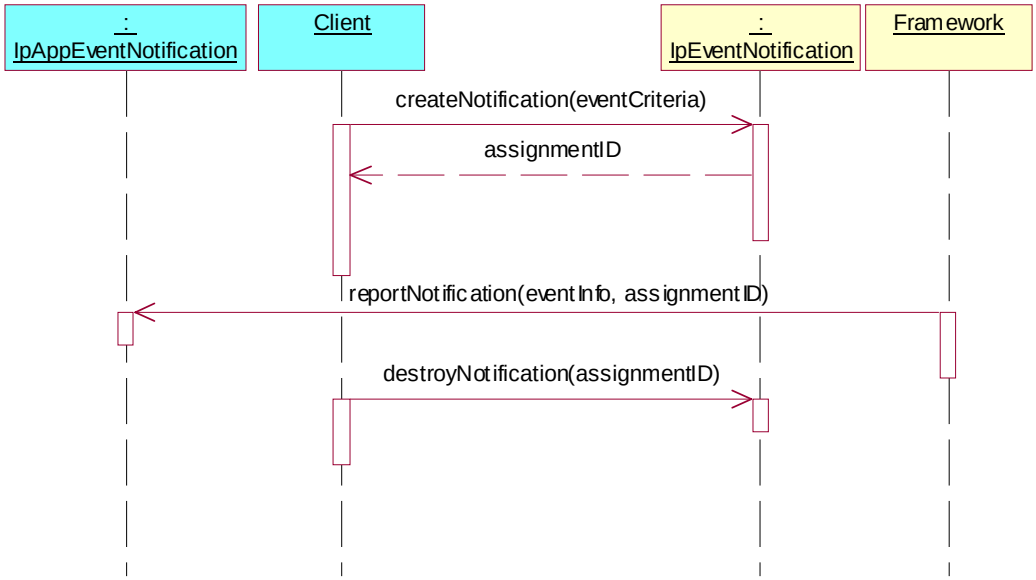
Precondition: IpEventNotification supported

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Application) **IpAppEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned



Test FW_FA_EN_02

Summary: **IpEventNotification**, all methods, successful

Reference: ES 201 915-3 [1], clause 7.3.4

Precondition: IpEventNotification supported

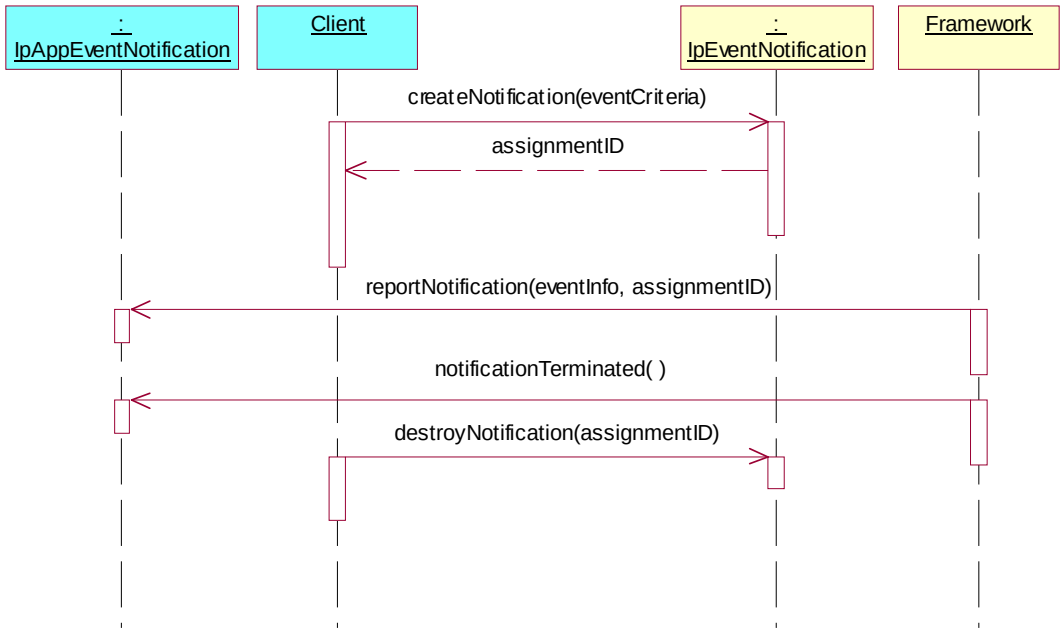
Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Application) **IpAppEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Triggered action: cause IUT to call **notificationTerminated()** method on the tester's (Application) **IpSvcEventNotification** interface.
   Parameters: none.
   Check: none.

4. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned



Test FW_FA_EN_03

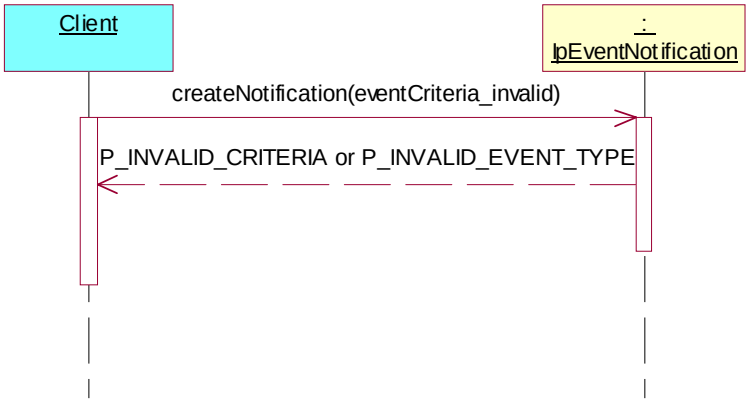
Summary: **IpEventNotification**, createNotification, P_INVALID_CRITERIA

Reference: ES 201 915-3 [1], clause 7.3.4

Precondition: IpEventNotification supported

Test Sequence:

1. Method call **createNotification()**
   Parameters: invalid eventCriteria
   Check: P_INVALID_ CRITERIA or P_INVALID_EVENT_TYPE exception is returned



Test FW_FA_EN_04

Summary: **IpEventNotification**, destroyNotification, P_INVALID_ ASSIGNMENT_ID

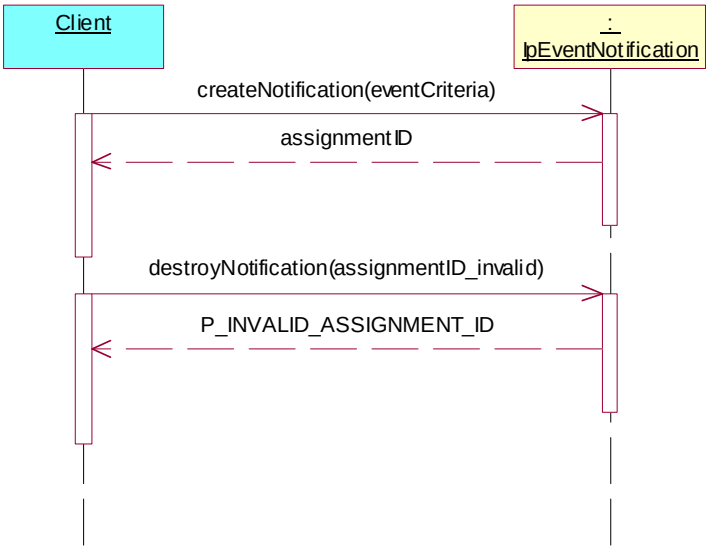
Reference: ES 201 915-3 [1], clause 7.3.4

Precondition: IpEventNotification supported

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Method call **destroyNotification()**
   Parameters: invalid assignmentID
   Check: P_INVALID_ ASSIGNMENT_ID exception is returned



5.4.3 Framework to Service API

5.4.3.1 Service Registration (SR)

Test FW_FS_SR_01

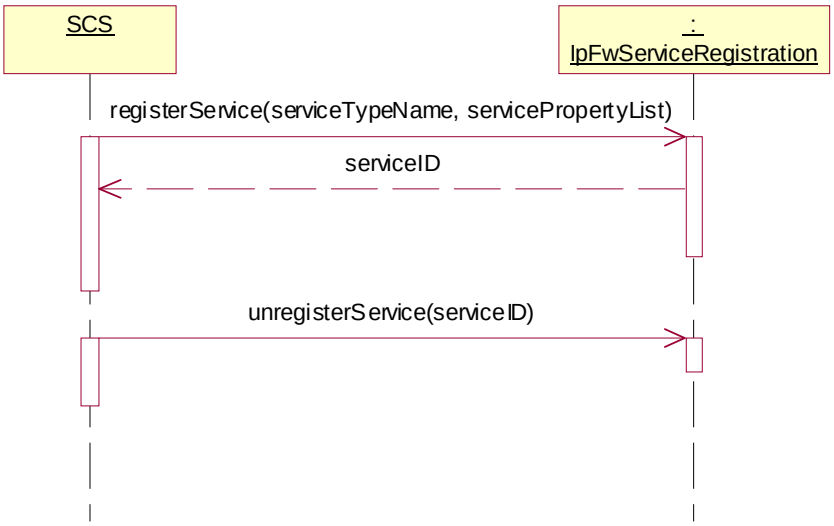
Summary: **IpFwServiceRegistration**, registerService and unregisterService methods, successful

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList
   Check: valid value of TpServiceID is returned

2. Method call **unregisterService()**
   Parameters: serviceID given in 1.
   Check: no exception is returned



Test FW_FS_SR_02

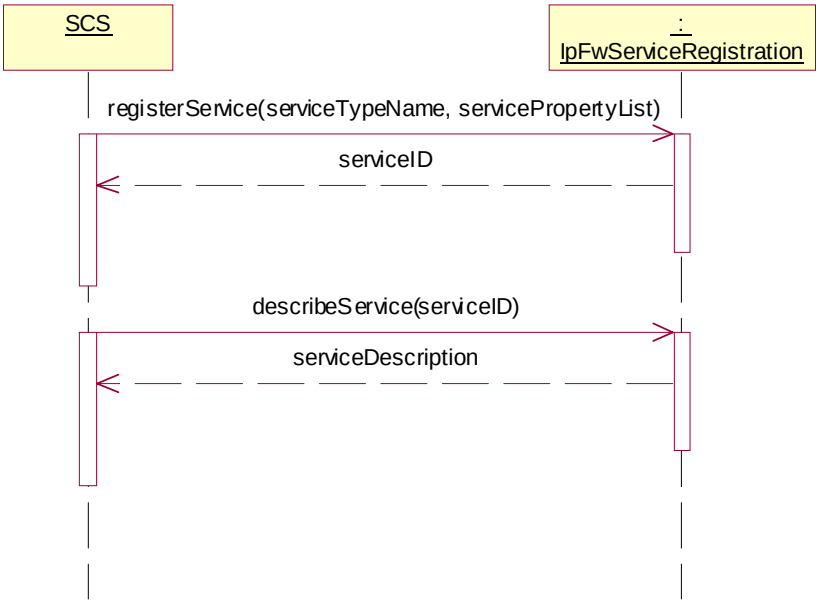
Summary: **IpFwServiceRegistration**, describeService method, successful

Reference: ES 201 915-3 [1], clause 9.3.1

Preamble: the service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **describeService()**
   Parameters: serviceID as returned by the registerService method
   Check: valid value of TpServiceDescription is returned



Test FW_FS_SR_03

Summary: **IpFwServiceRegistration**, announceServiceAvailability and unannounceService methods, successful

Reference: ES 201 915-3 [1], clause 9.3.1

Preamble: the service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **announceServiceAvailability()**
   Parameters: serviceID as returned by the registerService method,
   serviceInstanceLifeCycleManagerRef as returned by the createServiceManager method
   Check: no exception is returned.

2. Method call **unannounceService()**
   Parameters: serviceID as returned by the registerService method,
   Check: no exception is returned.

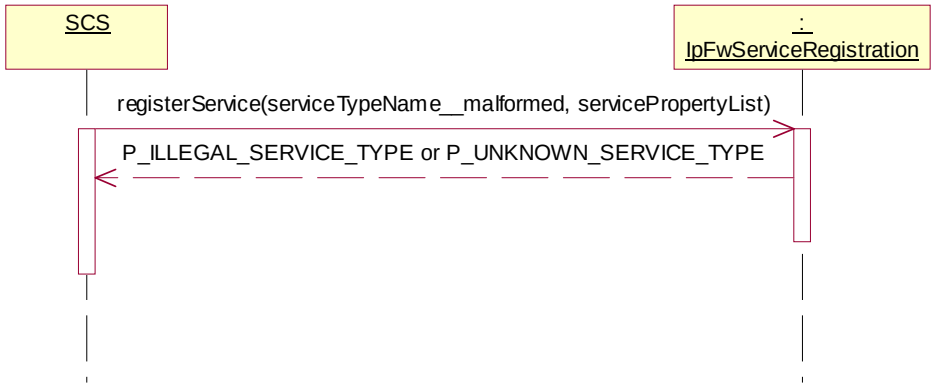
Test FW_FS_SR_04

Summary: **IpFwServiceRegistration**, registerService methods, P_ILLEGAL_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName malformed, servicePropertyList
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE exception is returned



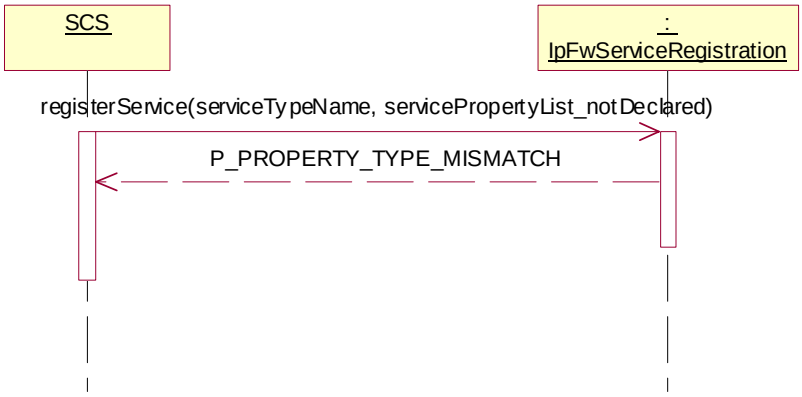
Test FW_FS_SR_05

Summary: **IpFwServiceRegistration**, registerService methods, P_PROPERTY_TYPE_MISMATCH

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList with a type of a property values not the same as the declared service type.
   Check: P_PROPERTY_TYPE_MISMATCH exception is returned



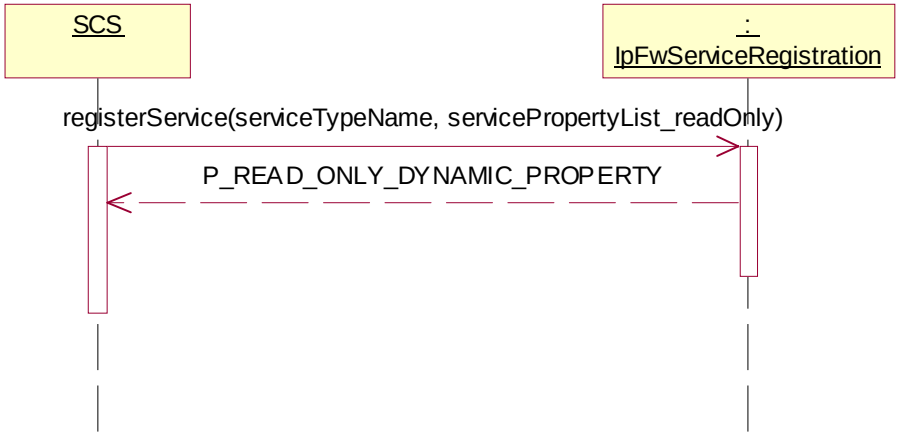
Test FW_FS_SR_06

Summary: **IpFwServiceRegistration**, registerService methods, P_READ_ONLY_DYNAMIC_PROPERTY

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList with assignment of a dynamic value to a read only property.



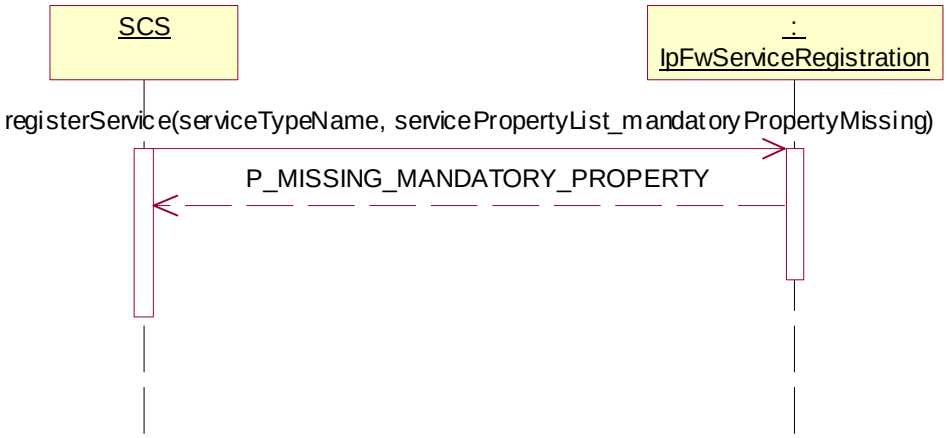
Test FW_FS_SR_07

Summary: **IpFwServiceRegistration**, registerService methods, P_MISSING_MANDATORY_PROPERTY

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList with a mandatory property missing.
   Check: P_MISSING_MANDATORY_PROPERTY exception is returned



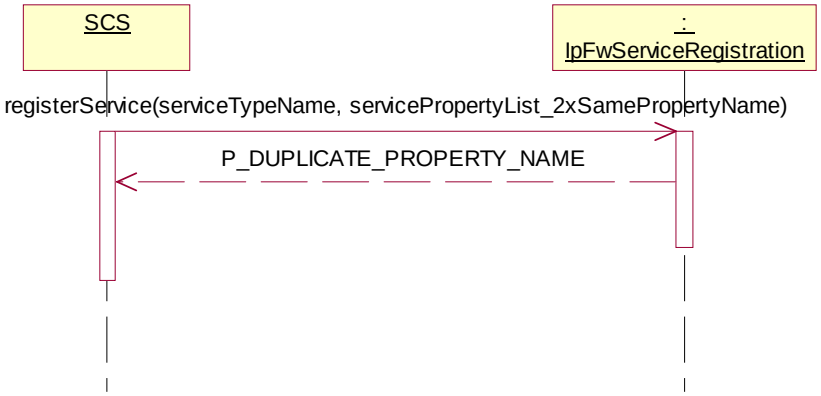
Test FW_FS_SR_08

Summary: **IpFwServiceRegistration**, registerService methods, P_DUPLICATE_PROPERTY_NAME

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList including two properties with the same property name.
   Check: P_DUPLICATE_PROPERTY_NAME exception is returned



Test FW_FS_SR_09

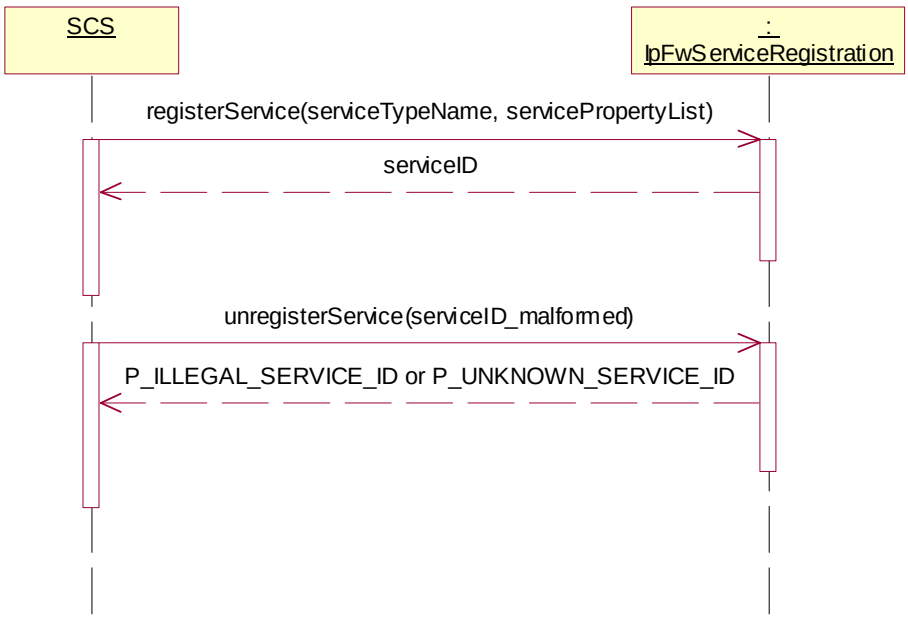
Summary: **IpFwServiceRegistration**, registerService and unregisterService methods, P_ILLEGAL_SERVICE_ID

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList
   Check: valid value of TpServiceID is returned

2. Method call **unregisterService()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID exception is returned



Test FW_FS_SR_10

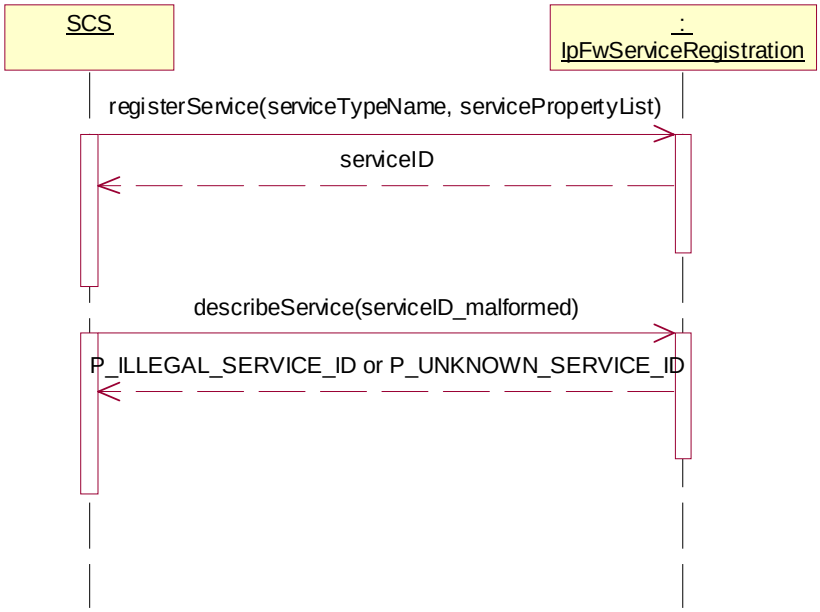
Summary: **IpFwServiceRegistration**, describeService method, P_ILLEGAL_SERVICE_ID

Reference: ES 201 915-3 [1], clause 9.3.1

Preamble: the service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **describeService()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID exception is returned



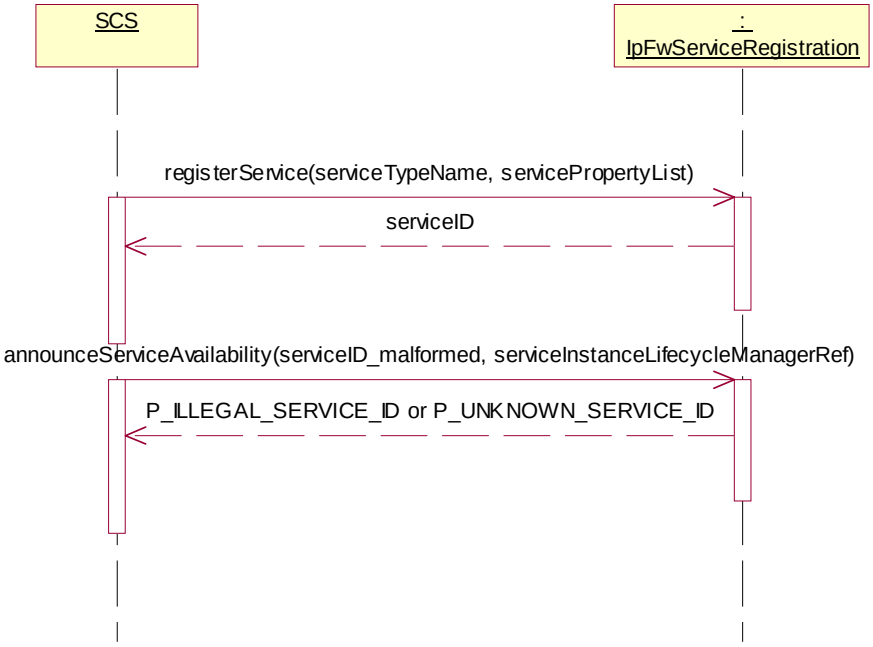
Test FW_FS_SR_11

Summary: **IpFwServiceRegistration**, announceServiceAvailability methods, P_ILLEGAL_SERVICE_ID

Reference: ES 201 915-3 [1], clause 9.3.1

Test Sequence:

1. Method call **announceServiceAvailabilty()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID exception is returned



Test FW_FS_SR_12

Summary: **IpFwServiceRegistration**, announceServiceAvailability and unannounceService methods, P_ILLEGAL_SERVICE_ID

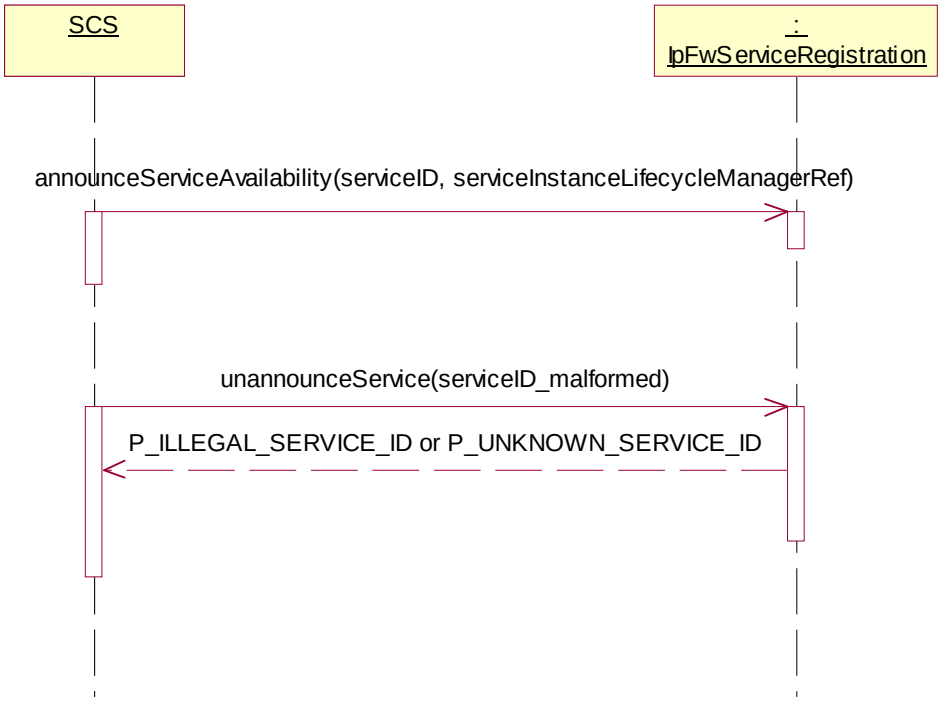
Reference: ES 201 915-3 [1], clause 9.3.1

Preamble: the service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **announceService()**
   Parameters: serviceID as returned by the registerService method,
   serviceInstanceLifeCycleManagerRef as returned by the createServiceManager method
   Check: no exception is returned.

2. Method call **unannounceService()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID exception is returned



5.4.3.2 Service Instance Lifecycle Management (SILM)

Test FW_FS_SILM_01

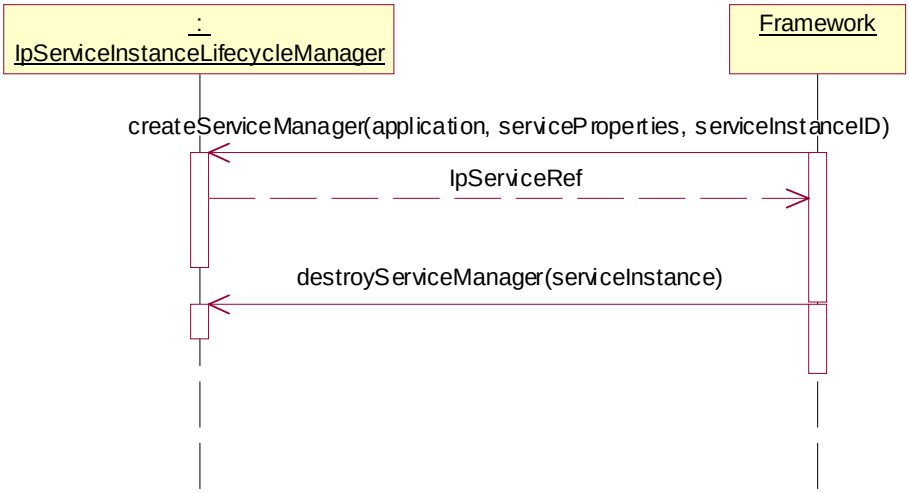
Summary: **IpServiceInstanceLifecycleManager**, createServiceManager and destroyServiceManager methods, successful

Reference: ES 201 915-3 [1], clause 9.3.2

Test Sequence:

1. Triggered action: cause IUT to call **createServiceManager()** method on the tester's (Service) **IpServiceInstanceLifecycleManager** interface.
   Parameters: application, serviceProperties, serviceInstanceID
   Check: valid value of IpServiceRef is returned

2. Triggered action: cause IUT to call **destroyServiceManager** method on the tester's (Service) **IpServiceInstanceLifecycleManager** interface. **()**
   Parameters: serviceInstanceID (same value as used in 1..
   Check: no exception is returned



5.4.3.3 Service Discovery (SD)

Test FW_FS_SD_01

Summary: **IpFwServiceDiscovery** all methods, successful

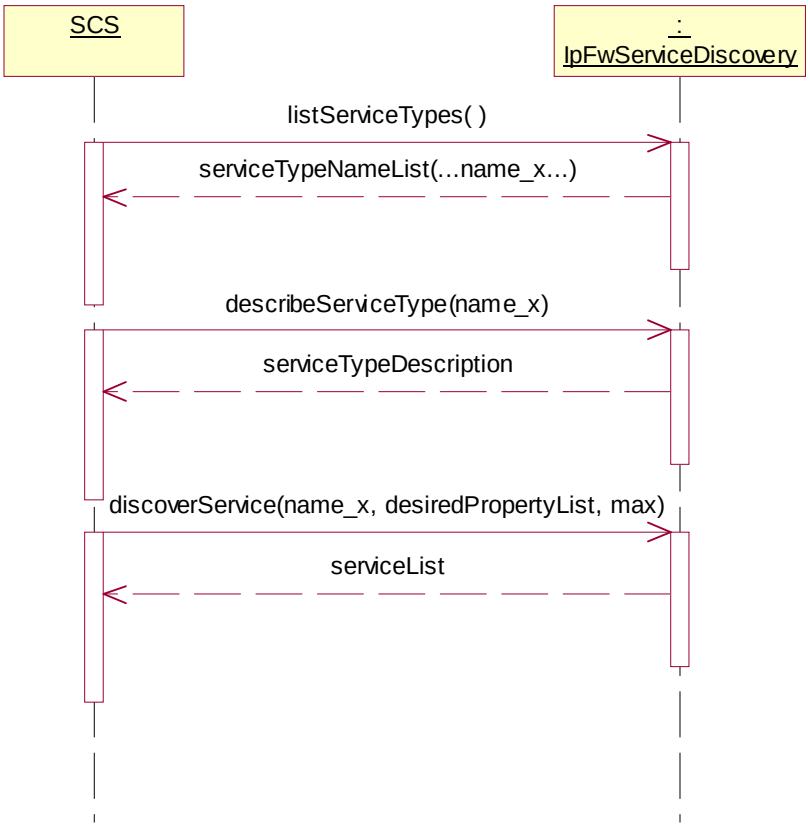
Reference: ES 201 915-3 [1], clause 9.3.3

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

3. Method call **describeServiceType()**
   Parameters: serviceTypeName from the list returned in 1.
   Check: valid value of TpServiceTypeDescription is returned

4. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., valid desiredPropertyList, valid max
   Check: valid value of TpServiceList is returned



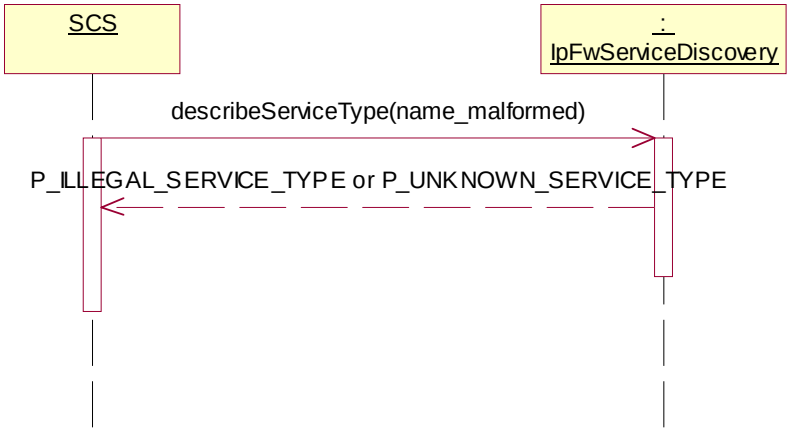
Test FW_FS_SD_02

Summary: **IpFwServiceDiscovery** describeServiceType, P_ILLEGAL_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 9.3.3

Test Sequence:

1. Method call **describeServiceType()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE exception is returned



Test FW_FS_SD_03

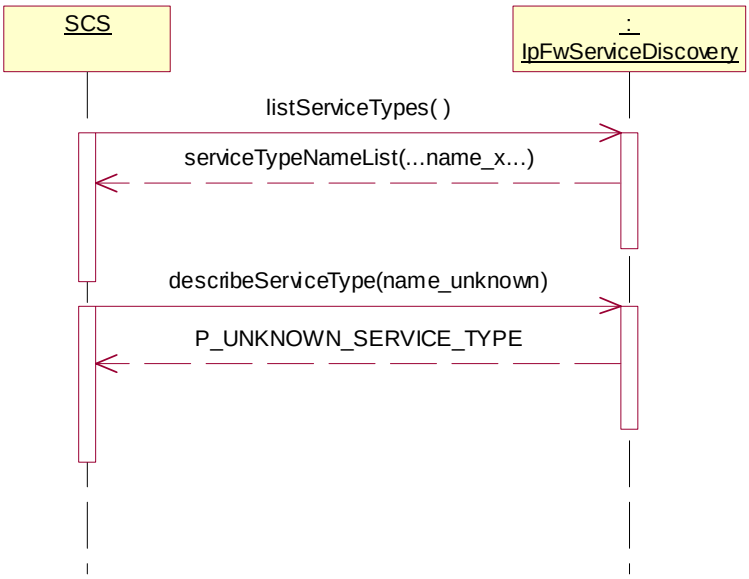
Summary: **IpFwServiceDiscovery** describeServiceType, P_UNKNOWN_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 9.3.3

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **describeServiceType()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE exception is returned



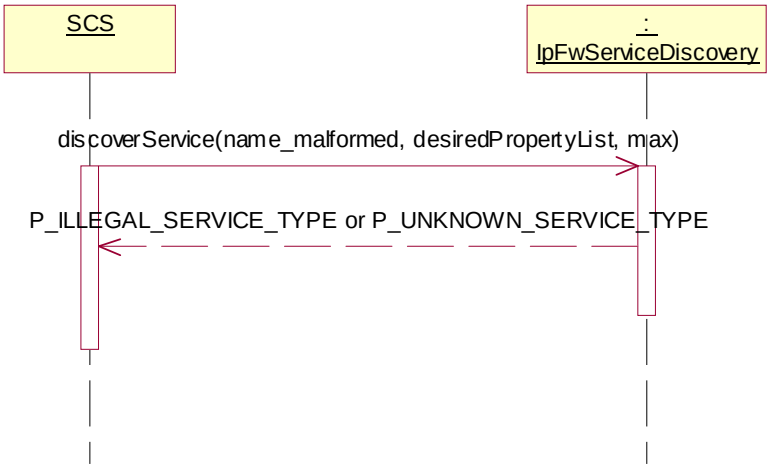
Test FW_FS_SD_04

Summary: **IpFwServiceDiscovery** discoverService, P_ILLEGAL_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 9.3.3

Test Sequence:

1. Method call **discoverService()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE exception is returned



Test FW_FS_SD_05

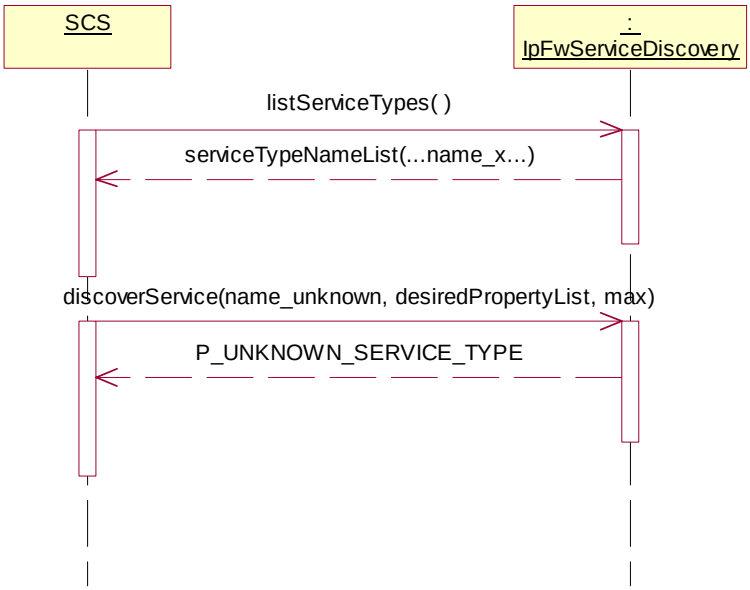
Summary: **IpFwServiceDiscovery** discoverService, P_UNKNOWN_SERVICE_TYPE

Reference: ES 201 915-3 [1], clause 9.3.3

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE exception is returned



Test FW_FS_SD_06

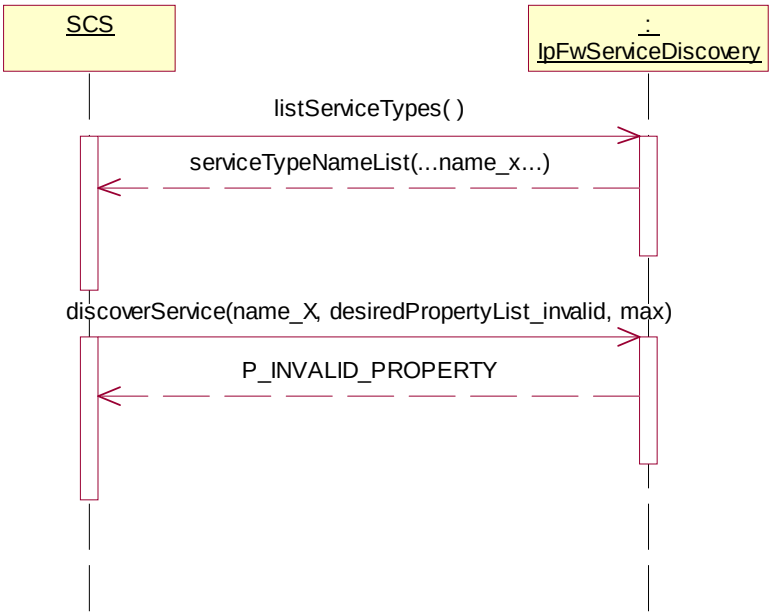
Summary: **IpFwServiceDiscovery** discoverService, P_INVALID_PROPERTY

Reference: ES 201 915-3 [1], clause 9.3.3

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., invalid desiredPropertyList, valid max
   Check: P_INVALID_PROPERTY exception is returned



Test FW_FS_SD_07

Summary: **IpFwServiceDiscovery** listRegisteredServices, successful

Reference: ES 201 915-3 [1], clause 9.3.3

Precondition: listRegisteredServices supported

Test Sequence:

1. Method call **listRegisteredServices()**
   Parameters: none
   Check: valid value of TpServiceList is returned

5.4.3.4 Integrity Management (IM)

Test FW_FS_IM_01

Summary: **IpFwHeartBeatMgmt**, all methods, successful

Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwHeartBeatMgt supported

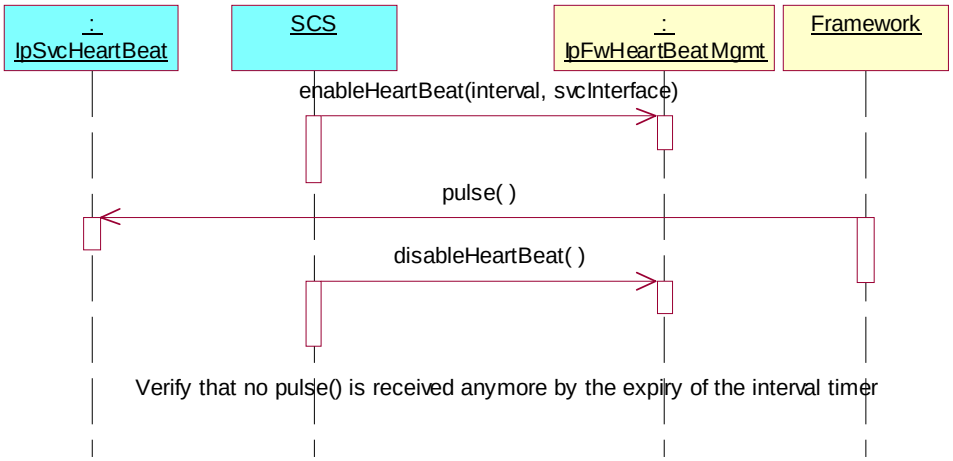
Preamble: The calling service must have a callback interface and a reference to this interface.

Test Sequence:

1. Method call **enableHeartBeat()**
   Parameters: interval, svcInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to regularly call **pulse()** method on the tester's (Service) **IpSvcHeartBeat** interface.
   Parameters: none
   Check: no exception is returned. Check also that the pulse() method is invoked at the requested interval.

3. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no **pulse()** is received anymore by the expiry of the interval timer.



Test FW_FS_IM_02

Summary: **IpFwHeartBeatMgmt**, all methods, successful

Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwHeartBeatMgmt, changeInterval supported

Preamble: The calling service must have a callback interface and a reference to this interface.

Test Sequence:

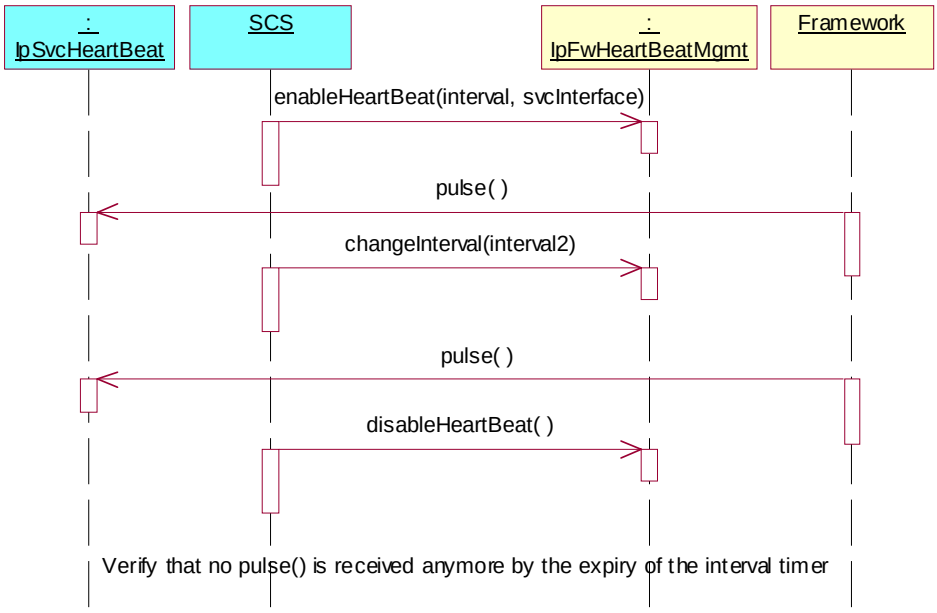
1. Method call **enableHeartBeat()**
   Parameters: interval, svcInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to call **pulse()** method regularly on the tester's (Service) **IpSvcHeartBeat** interface.
   Parameters: none
   Check: no exception is returned. Check also that the pulse() method is invoked at the requested interval.

3. Method call **changeInterval()**
   Parameters: interval
   Check: no exception is returned.

4. Triggered action: cause IUT to call **pulse()** method regularly on the tester's (Service) **IpSvcHeartBeat** interface.
   Parameters: none
   Check: the pulse() method is invoked at the new requested interval.

5. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception Verify that no **pulse()** is received anymore by the expiry of the interval timer.



Test FW_FS_IM_03

Summary: **IpFwHeartBeat**, all methods, successful

Reference: ES 201 915-3 [1], clause 9.3.4

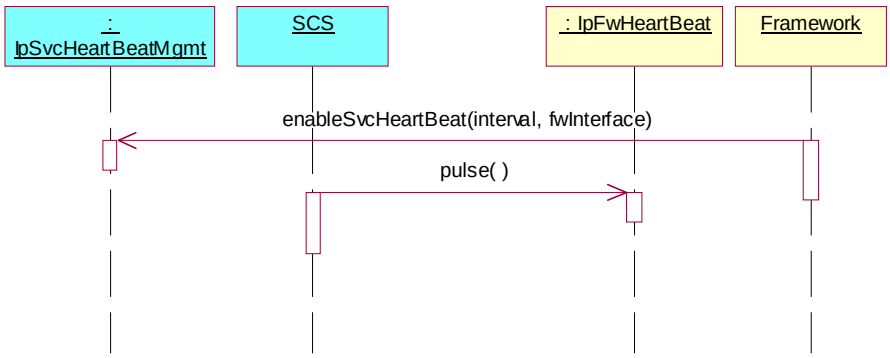
Precondition: IpFwHeartBeat is supported

Preamble: The calling service must have a callback interface and a reference to this interface.

Test Sequence:

1. Triggered action: cause IUT to call **enableHeartBeat ()** method on the tester's (Service) **IpSvcHeartBeatMgmt** interface.
   Parameters: interval, fwInterface

2. Method call **pulse()**
   Parameters: none
   Check: no exception



Test FW_FS_IM_04

Summary: **IpFwFaultManager** activityTestReq, successful

Reference: ES 201 915-3 [1], clause 9.3.4

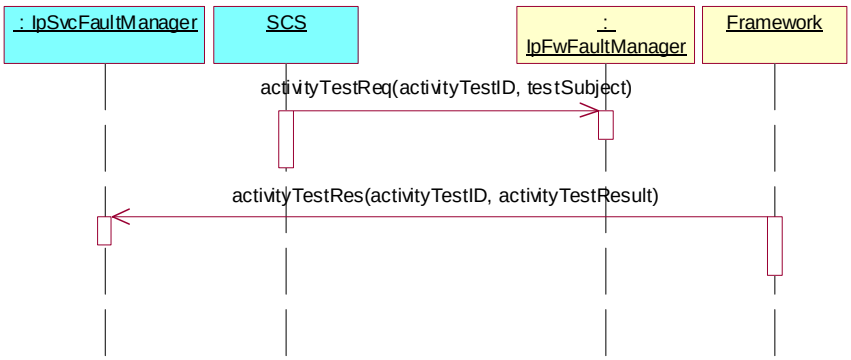
Precondition: IpFwFaultManager, activitiyTestReq supported

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: activityTestID, testSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call **activityTestRes ()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID, activityTestResult



Test FW_FS_IM_05

Summary: **IpFwFaultManager** genFaultStatsRecordReq, successful

Reference: ES 201 915-3 [1], clause 9.3.4

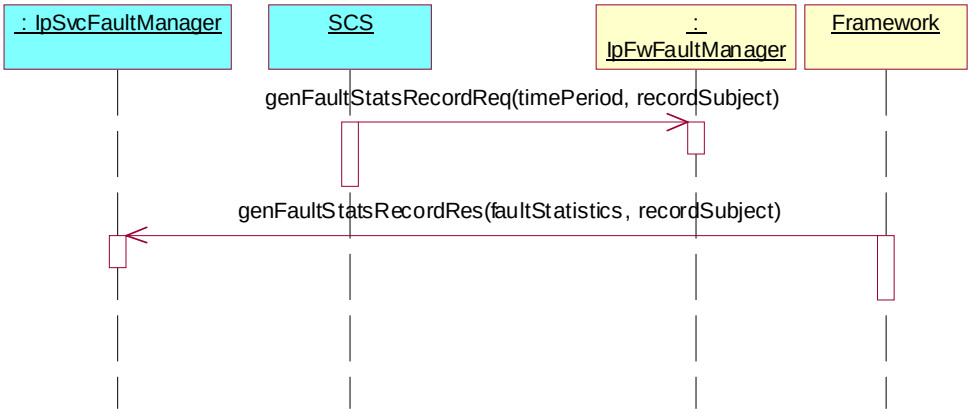
Precondition: IpFwFaultManager, genFaultStatsRecordReq supported

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **genFaultStatsRecordReq()**
   Parameters: timePeriod, recordSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call **genFaultStatsRecordRes()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: faultStatistics, recordSubject



Test FW_FS_IM_06

Summary: **IpFwFaultManager** genFaultStatsRecordReq, unsuccessful

Reference: ES 201 915-3 [1], clause 9.3.4

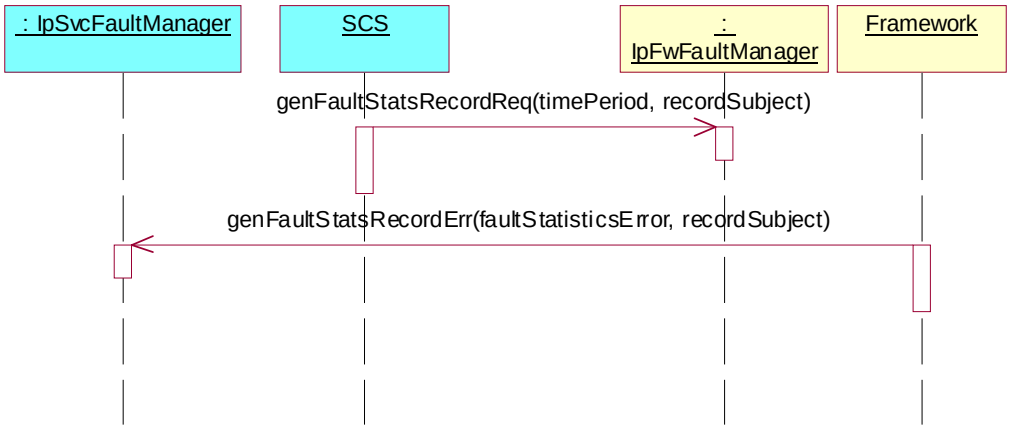
Precondition: IpFwFaultManager, and genFaultStatsRecordErr supported
FW configured to respond with **genFaultStatsRecordErr** (undefined or unavailable)

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **genFaultStatsRecordReq()**
   Parameters: timePeriod, recordSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call **genFaultStatsRecordErr()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: faultStatisticsError, recordSubject



Test FW_FS_IM_07

Summary: **IpFwFaultManager** svcUnavailableInd, successful

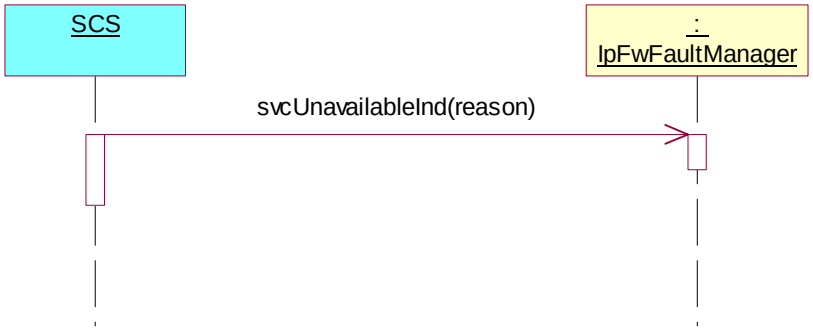
Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwFaultManager, svcUnavailableInd supported

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **svcUnavailableInd()**
   Parameters: reason
   Check: no exception is returned



Test FW_FS_IM_08

Summary: **IpFwFaultManager** svcActivityTestRes, successful

Reference: ES 201 915-3 [1], clause 9.3.4

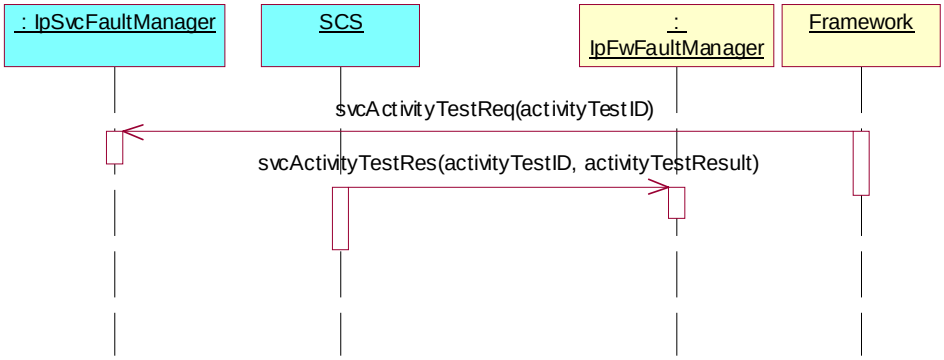
Precondition: IpFwFaultManager, svcActivityTestRes supported

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestRes()**
   Parameters: activityTestID, activityTestResult
   Check: no exception is returned



Test FW_FS_IM_09

Summary: **IpFwFaultManager** svcActivityTestRes, P_INVALID_ACTIVITY_TEST_ID Exception

Reference: ES 201 915-3 [1], clause 9.3.4

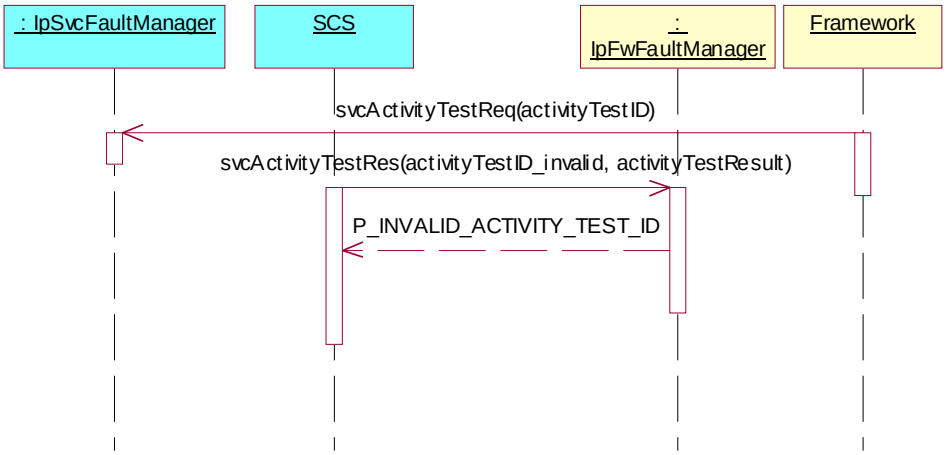
Precondition: IpFwFaultManager, svcActivityTestRes supported

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestRes()**
   Parameters: invalid activityTestID, activityTestResult
   Check: P_INVALID_ACTIVITY_TEST_ID exception is returned



Test FW_FS_IM_10

Summary: **IpFwFaultManager** svcActivityTestErr, successful

Reference: ES 201 915-3 [1], clause 9.3.4

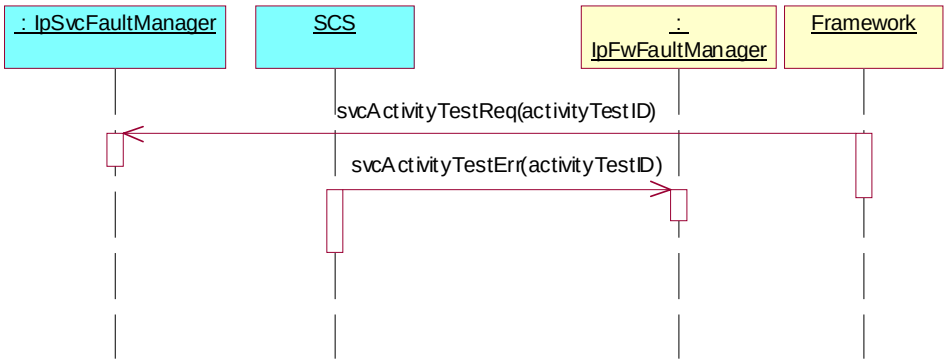
Precondition: IpFwFaultManager, svcActivityTestRes supported

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestErr()**
   Parameters: activityTestID
   Check: no exception is returned



Test FW_FS_IM_11

Summary: **IpFwFaultManager** svcActivityTestErr, P_INVALID_ACTIVITY_TEST_ID exception

Reference: ES 201 915-3 [1], clause 9.3.4

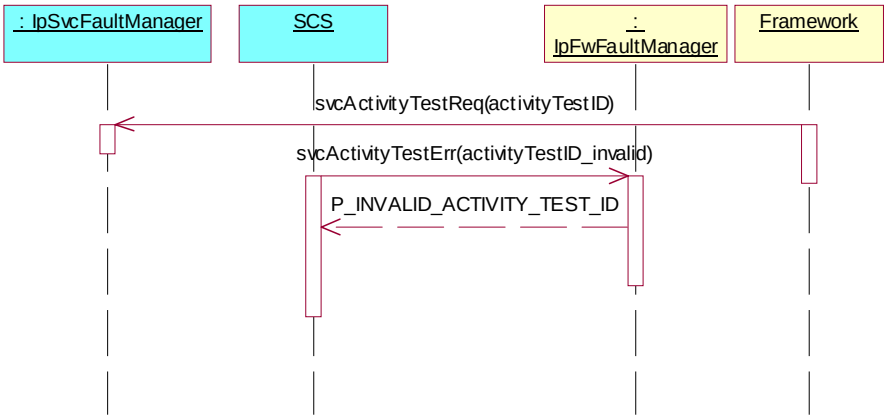
Precondition: IpFwFaultManager, svcActivityTestRes supported

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestErr()**
   Parameters: invalid activityTestID
   Check: P_INVALID_ACTIVITY_TEST_ID exception is returned



Test FW_FS_IM_12

Summary: **IpFwFaultManager** appUnavailableInd, successful

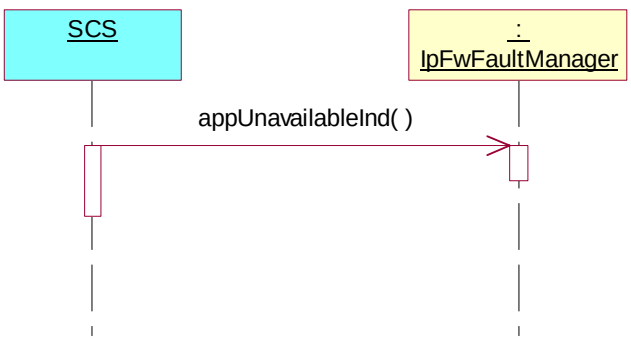
Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwFaultManager, appUnavailableInd supported

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **appUnavailableInd()**
   Parameters: none
   Check: no exception is returned



Test FW_FS_IM_13

Summary: createLoadLevelNotification and destroyLoadLevelNotification methods, successful

Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwLoadManager, createLoadLevelNotification and destroyLoadLevelNotification notifications supported.

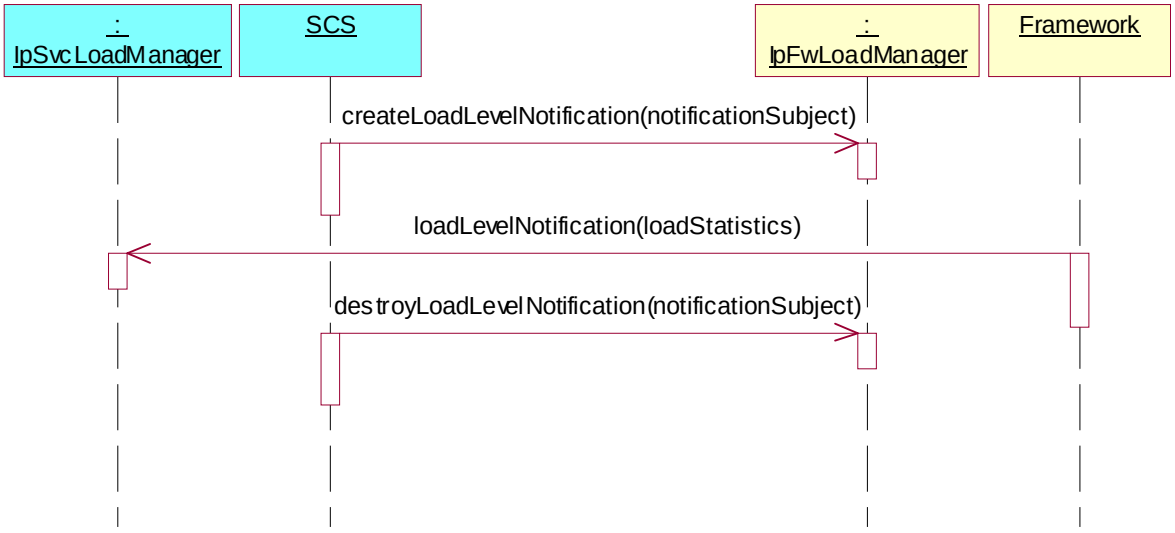
Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call **loadLevelNotification()** method on the tester's (Application) **IpSvcLoadManager** interface.
   Parameters: loadStatistics

3. Method call **destroyLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned



Test FW_FS_IM_14

Summary: **IpFwLoadManager** All methods, successful

Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwLoadManager, notifications with suspendNotification and resumeNotification supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

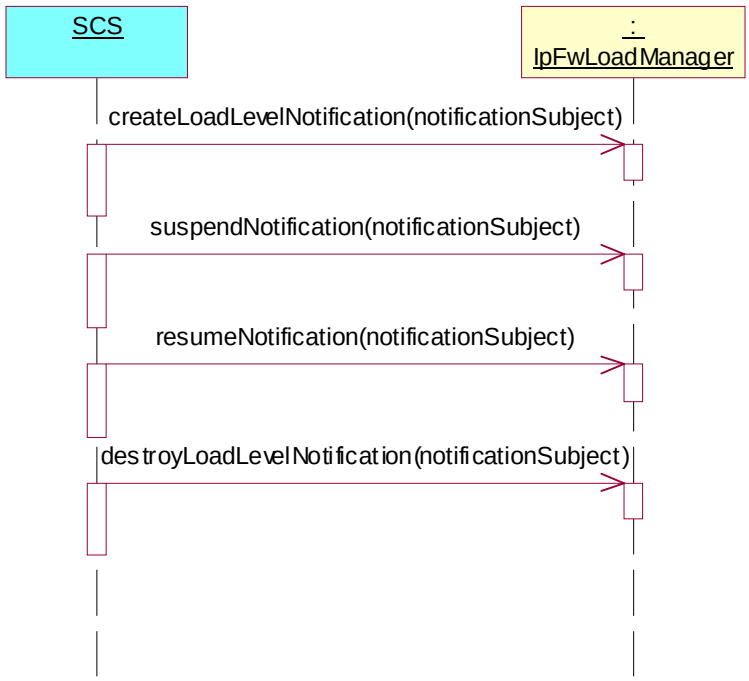
Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned

2. Method call **suspendNotification()**
   Parameters: notificationSubject
   Check: no exception is returned

3. Method call **resumeNotification()**
   Parameters: notificationSubject
   Check: no exception is returned

4. Method call **destroyLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned



Test FW_FS_IM_15

Summary: **IpFwLoadManager** reportLoad, successful

Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwLoadManager, reportLoad supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **reportLoad ()**
   Parameters: loadLevel
   Check: no exception is returned

Test FW_FS_IM_16

Summary: **IpFwLoadManager** queryLoadReq, successful

Reference: ES 201 915-3 [1], clause 9.3.4

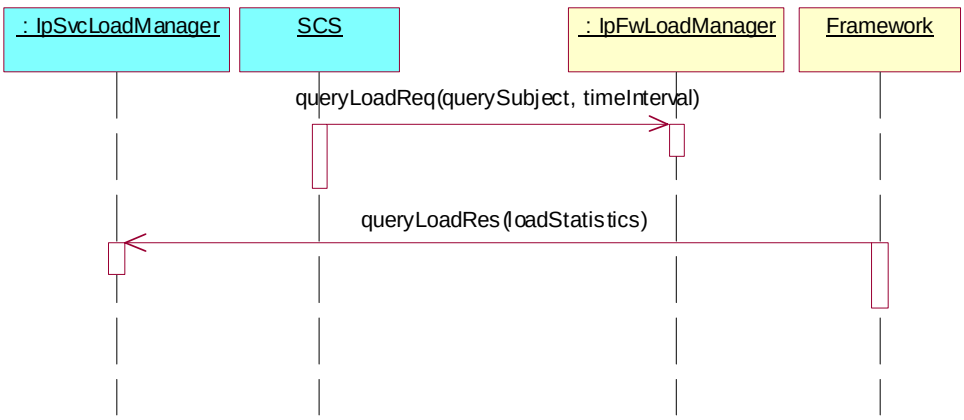
Precondition: IpFwLoadManager, queryLoadReq supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **queryLoadReq()**
   Parameters: querySubject, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **queryLoadRes ()** method on the tester's (Service) **IpSvcLoadManager** interface.
   Parameters: loadStatistics
   Check: no exception is returned.



Test FW_FS_IM_17

Summary: **IpFwLoadManager** querySvcLoadRes, successful

Reference: ES 201 915-3 [1], clause 9.3.4

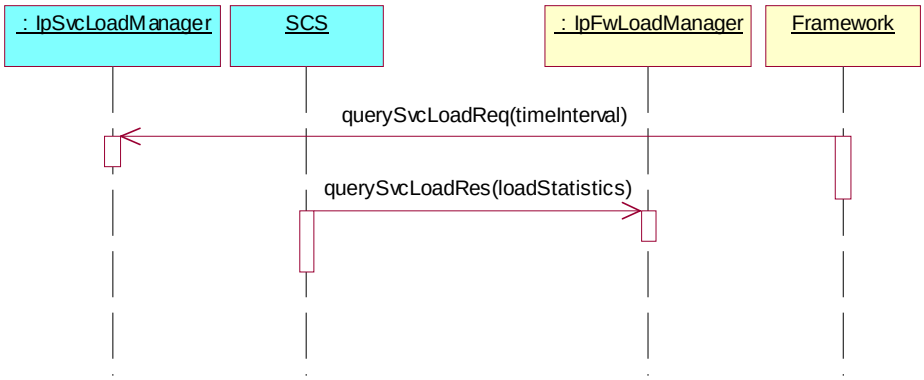
Precondition: IpFwLoadManager, querySvcLoadRes supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Triggered action: cause IUT to call **querySvcLoadReq ()** method on the tester's (Service) **IpSvcLoadManager** interface.
   Parameters: timeInterval
   Check: no exception is returned.

2. Method call **querySvcLoadRes ()**
   Parameters: loadStatistics
   Check: no exception is returned



Test FW_FS_IM_18

Summary: **IpFwLoadManager** querySvcLoadErr, successful

Reference: ES 201 915-3 [1], clause 9.3.4

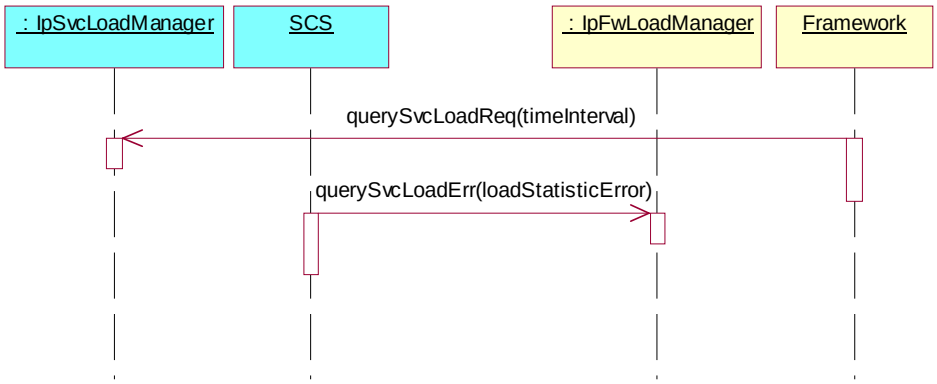
Precondition: IpFwLoadManager, querySvcLoadRes supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Triggered action: cause IUT to call **querySvcLoadReq ()** method on the tester's (Service) **IpSvcLoadManager** interface.
   Parameters: timeInterval
   Check: no exception is returned.

2. Method call **querySvcLoadErr ()**
   Parameters: loadStatisticsError
   Check: no exception is returned



Test FW_FS_IM_19

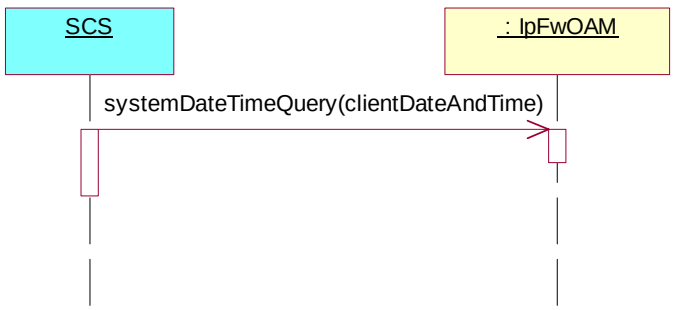
Summary: **IpFwOAM**, systemDateTimeQuery, successful

Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwOAM supported

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: clientDateAndTime
   Check: valid value of TpDateAndTime is returned



Test FW_FS_IM_20

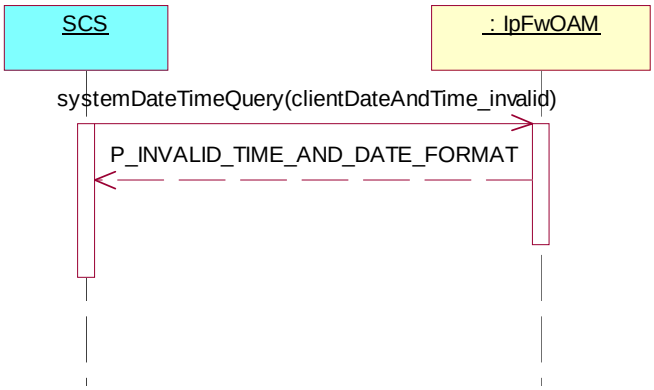
Summary: **IpFwOAM**, systemDateTimeQuery, P_INVALID_TIME_AND_DATE_FORMAT exception

Reference: ES 201 915-3 [1], clause 9.3.4

Precondition: IpFwOAM supported

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: invalid clientDateAndTime
   Check: P_INVALID_TIME_AND_DATE_FORMAT is returned



5.4.3.5 Event Notification (EN)

Test FW_FS_EN_01

Summary: **IpFwEventNotification**, create and destroy methods, successful

Reference: ES 201 915-3 [1], clause 9.3.5

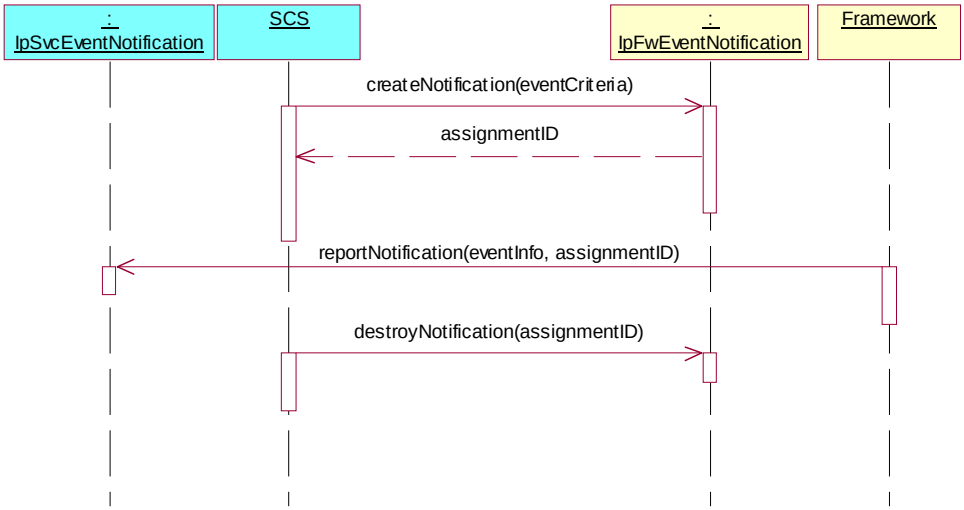
Precondition: IpFwEventNotification supported

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Service) **IpSvcEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned



Test FW_FS_EN_02

Summary: **IpFwEventNotification**, all methods, successful

Reference: ES 201 915-3 [1], clause 9.3.5

Precondition: IpFwEventNotification supported

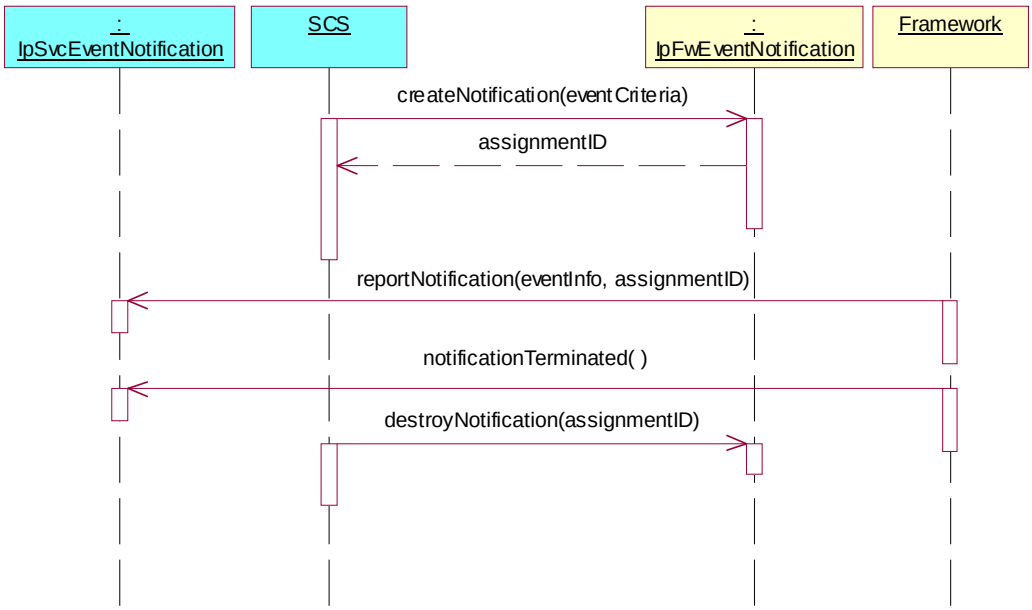
Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Service) **IpSvcEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Triggered action: cause IUT to call **notificationTerminated()** method on the tester's (Service) **IpSvcEventNotification** interface.
   Parameters: none.

4. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned



Test FW_FS_EN_03

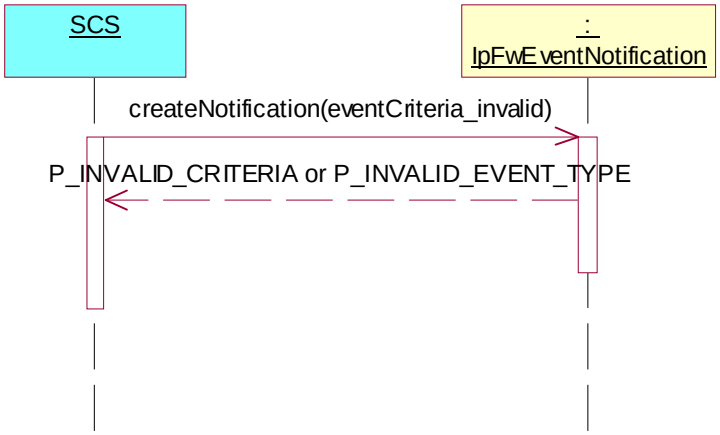
Summary: **IpFwEventNotification**, createNotification, P_INVALID_CRITERIA

Reference: ES 201 915-3 [1], clause 9.3.5

Precondition: IpFwEventNotification supported

Test Sequence:

1. Method call **createNotification()**
   Parameters: invalid eventCriteria
   Check: P_INVALID_ CRITERIA or P_INVALID_EVENT_TYPE exception is returned



Test FW_FS_EN_04

Summary: **IpFwEventNotification**, destroyNotification, P_INVALID_ ASSIGNMENT_ID

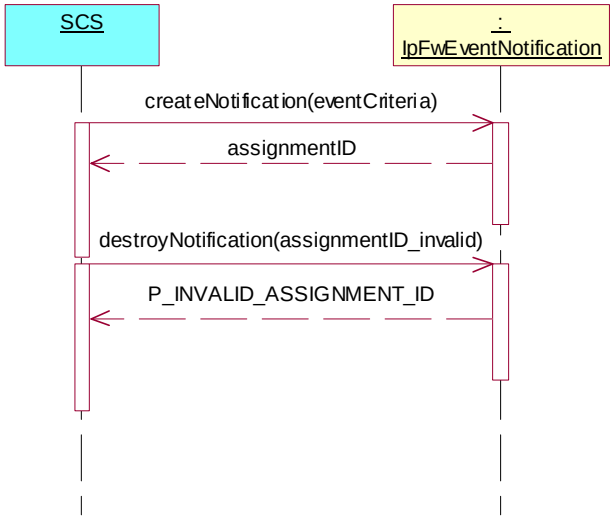
Reference: ES 201 915-3 [1], clause 9.3.5

Precondition: IpFwEventNotification supported

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Method call **destroyNotification()**
   Parameters: invalid assignmentID
   Check: P_INVALID_ ASSIGNMENT_ID exception is returned



History

Document history		
V1.1.1	June 2003	Membership Approval Procedure MV 20030801: 2003-06-03 to 2003-08-01
V1.1.1	August 2003	Publication