RED ELÉCTRICA DE ESPAÑA

Grupo Red Eléctrica

#SERCLIENTES

IEC 62325-504

Test Cases

Dirección de **Servicios para la Operación**

Versión 1.0. Marzo 2016

Departamento de **Medidas**

It's very simple

you read the protocol and write the code.

Bill Joy

Version control

Version	Date	Changes
0.1	17/09/2015	First version. List, get and query data operations.
0.2	20/01/2016	Editorial changes.
1.0	01/03/2016	Editorial changes. Put operation. More error messages.

INDEX

1 Introduction

This document develops a basic test cases guide for the IEC 62325-504 web services protocol.

For server testing, the use of a general web service test client application[1] is recommended. So that all the possible use cases could be performed (even those that implies the use of incorrect requests). Note that this kind of clients usually does not honour SSL constraints)

For client testing, depending on the nature of the client, some of the use cases could not apply. For instance, an interactive client that allows a user to change freely the filters will be able to perform mostly of the use cases. However, clients that use a fixed set of values to create the request will not be able to be adapted to some cases present in this guide. In order to ensure full test cases coverage, the client should connect to a server "rigged" which will (for instance) send responses with an invalid signature.

Soap request examples are provided for each test case when possible.

Test cases concerning the underlying business that is implemented are outside the scope of this document.

References:

[1] Hypertext Transfer Protocol status definitions. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[2] IEC TS 62325-504:2015 Utilization of web services for electronic data interchanges on the European energy Market for electricity.

[3] IEC 61968-100:2013 System interfaces for distribution management – Implementation profiles

[4] IEC 62325-451-5 Problem statement and status request business processes

[1] Smart Bear SoapUI, WebInject, etc.

2 Security

General notes:

- If there are other environments available besides production (i.e. test, sqa, etc.) The certificate used for document signing in production must be different to the rest. If the server uses the same certificate for document signing in other environments, a user can claim that certain document (acknowledgement document, market result, etc.) obtained from a non-production environment was generated by the production one.
- Where signature applies, besides the signature validation, the application (client and server) must check that the signature certificate is valid, trusted and has relationship with the request certificate. In a basic scenario signature and secure channel (https) use the same certificate but it is possible to create a request that is signed with a certificate "B" whereas the SSL context is created with certificate "A". Note that this situation could be valid if user "A" represent user "B" (one company acts on behalf other company)
- Global schema validation will not detect errors on "any" elements ("any" = any document). That is, signature and xml payload must be validate against schema separately. When it comes to perform the signature validation there are some casuistic that must be taken into consideration:
    - Invalid xml (not valid against its schema): Signature cannot be validated. Even if the signature is valid against its schema, it would be possible that the application is unable to validate the signature because:
        - Canonicalization / signature / transform / hash algorithms are incorrect or unsupported.
        - Certification serial number is incorrect.
        - The certificate value (PEM encoded) is not valid. That is, it is impossible to build a public key with the provided value.
    - Signatures with invalid reference: Signature validation will fail because the provided hash will not be the same as the calculated. Bear in mind the according to [2] signature covers the whole request document, not the soap.
    - Signatures created with expired certificates or untrusted certificates must be rejected even if the signature is valid.
    - Signatures which certificate has no relationship with the certificate used to create the SSL context must also be rejected.
    - Requests in which there are no relationship between the SSL certificate, signature certificate and payload. When the XML payload has an element to identify the sender (in the European context "`sender_MarketParticipant.mRID`") the application must check that identity has relationship with the sender certificate. That is, the application must check that the request is not trying to send information on behalf other user (stated in the payload) where there is no relationship between them (between security certificates and payload identification).

- [2] Security is based on the use of digital certificates for client and server (2 way SSL). However it is possible a “relaxed” implementation with only server certificates provided that:
    - No confidentiality rules apply in the system.
    - The server shall reject all put requests. If the application needs user’s entries, these must be submitted using other web service entry point (a different URL) that shall be protected with 2 way SSL.
    - There are no queries (if the operation QueryData is implemented) that could return confidential information.
    - This relaxed configuration must be well known and accepted by the application’s users.
- A configuration where the server has no certificate is not acceptable. This would imply the use of http instead of https and no possibility of document signing.
- Two way SSL must be implemented (configured) at application server or web server level, not at application level. Otherwise, the responsibility of taking the decision about whether reject a request must be implemented in the application and so it will consume resources (execution threads, memory, etc.) in order to take such decision. This leads in a waste of resources that could be used by an attacker to create a DOS situation.
- Client and server must honor SSL security constrains. Web service client must abort the connection if the received certificate is not trusted or if there is a mismatch between the received server credential and the URL used (i.e. client connects using an IP instead the server name)
- Physical layer elements (i.e. firewalls, routers, proxy, etc.) and media (i.e. Internet, intranet, dedicated line, etc.) are out of the scope of [2] and thus are not covered by this document.
- The security configuration should meet the latest Internet security recommendations (e.g. TLS 1.2 connection, SHA-256 algorithms, etc.).

Test ID: SE-01

Objective: Verify that the server handles 2-way SSL properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client has no certificate.

Action: Use a web service client to send a list request by code 0.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
	<soap:Header/>
	<soap:Body>
		<mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
			<mes:Header>
				<mes:Verb>get</mes:Verb>
				<mes:Noun>MessageList</mes:Noun>
			</mes:Header>
			<mes:Request>
				<mes:Option>
					<mes:name>Code</mes:name>
					<mes:value>0</mes:value>
				</mes:Option>
			</mes:Request>
		</mes:RequestMessage>
	</soap:Body>
</soap:Envelope>
```

Result: Server returns a 403 (Forbidden) error. Client must notify this error. Depending on server configuration, a general “Hand shake error” could be returned instead of an http error.

This behavior should also be tested with operations put, get and query (if applies)

Notes: Any other result (i.e. soap fault) would mean that the rejection is made at application level; consequently, it is not acceptable. This does not guarantees that the rejection is made at server level, application could be rejecting the request returning an http 403 error.

Test ID: SE-02

Objective: Verify that server handles 2-way SSL properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client uses a certificate that is not trusted by server (i.e. self-signed certificate).

Action: Use a web service client to send a list request by code 0. (See SE-01 request sample)

Result: Server returns a 403 (Forbidden) error. Client must notify this error. Depending on server configuration, a general “Hand shake error” could be returned instead of an http error.

Notes: This behavior should also be tested with operations put, get and query (if applies)

Test ID: SE-03

Objective: Verify that client handles 2-way SSL properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. Client trust store does not contain the server's certificate CA. Note that this test is intended for clients only; it makes no sense to use a generic SOAP test tool to perform this test.

Action: Use a web service client to send a list request by code 0. (See SE-01 request sample)

Result: Client aborts the connection with a "handshake" error. (HAND-013 error code can also be used)

Notes: The exact error text depends on the client implementation's platform. For instance, for java the exception text is:

```
javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to find
valid certification path to requested target
```

The connection to a server which certificate is expired (and so that, is not trusted) is similar. In this case, the root error reason (in java) contains the expiration date:

```
Caused by: java.security.cert.CertificateExpiredException: NotAfter: Tue
Oct 08 09:45:38 CEST 2030
        at sun.security.x509.CertificateValidity.valid(Unknown Source)
        at sun.security.x509.X509CertImpl.checkValidity(Unknown Source)
…
```

Test ID: SE-04

Objective: Verify that the server handles 2-way SSL properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client uses an expired certificate that was issued by a trusted CA.

Action: Use a web service client to send a list request by code 0. (See SE-01 request sample)

Result: Server returns a 401 (Unauthorized) error. Client must notify this error.

Notes: Java clients, by default, are unable to send request with an expired (or not yet available) certificate. This behavior should also be tested with operations put, get and query (if applies)

Test ID: SE-05

Objective: Verify that the server uses different certificates for different environments.

Server Preconditions: Server is configured and running. There are more than one environment (i.e. production, testing, sqa, etc.). There must be at least one message in each environment that can be requested by the user; otherwise, the server must implement query operation.

Client Preconditions: Client has access to all environments. (Could have one certificate or a set of them, one for each environment)

Action: Use a web service client to send a list request by code 0. (See SE-01 request sample) then request (`get`) the first message on the list. Repeat the list + get process in all available environments.

Result: For each request, server returns the message and its signature. Check, that the certificate used for signature in the production environment, is different to the one used in the others environments.

Alternative action: Use a web service client to send a "`query`" timestamp request. Repeat the request in all available environments.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
	<soap:Header/>
	<soap:Body>
		<mes:RequestMessage xmlns:mes="urn:iec62325.504:messages:1:0">
			<mes:Header>
				<mes:Verb>get</mes:Verb>
				<mes:Noun>QueryData</mes:Noun>
				<mes:Timestamp>2015-06-24T07:40:24.934Z</mes:Timestamp>
			</mes:Header>
			<mes:Request>
				<mes:Option>
					<mes:name>DataType</mes:name>
					<mes:value>serverTimestamp</mes:value>
				</mes:Option>
			</mes:Request>
		</mes:RequestMessage>
	</soap:Body>
</soap:Envelope>
```

Alternative Result: For each request, server returns the message and its signature. Check that the certificate used for signature in the production environment is different to the ones used in the others environments.

Test ID: SE-06

Objective: Verify that the client aborts the connection when the server credentials do not match with the used URL.

Server Preconditions: Server is configured and running.

Client Preconditions: Client uses the server IP instead the server name in the connection or uses a server name different to the one that appears in the server's certificate (DNS alias).

Action: Use a web service client to send a list request by code 0. (See SE-01 request sample)

Result: Client aborts the connection with a "hand shake" error. Error code HAND-014 can also be used. Server administrator should check that the server has not received the request at application level.

3 General

General notes:

- General test are intended to check the correct behavior under unexpected conditions. As a rule in order to facilitate audit tasks, the application shall not lose any message. Incorrect messages, messages send by users without rights, etc. shall not be available by the web services operations (i.e. cannot be listed or retrieved)
- Malformed messages usually do not reach the application layer and are rejected by application server. Some client implementations will refuse to send malformed message.
- If the user sends no IEC 61968-100 requests (arbitrary messages) the request could also be rejected at application server level, usually with fault message with the text "*unable to find a matching operation for …*"
- For security reasons, message validation (typically against its schema) must be performed after user validations. Otherwise, a malicious (and unauthorized) user could try to cause DOS attacks by sending huge messages to the server.

Proposed error codes:

- **HAND-001:** *Unable to retrieve remote user from the https context [IP=?].*
  Internal code used to send a forbidden error code.

- **HAND-002:** *Message is not valid against schema. Details: ?.*
  Received message (by client or server) is not valid against schema. Server could send bad request error code.

- **HAND-003**: *User has no proper role for current message type.*
  Internal code used to send an unauthorized error code.

- **HAND-004**: *Unable to read soap body.*
  Internal code used to send a bad request error code.

- **HAND-005:** *Unsupported combination: [verb=?][noun=?]*
  The verb and noun combination used in the request is not supported (do not correspond to any operation defined in [2]). Check the request and send the message.

- **HAND-006:** *Invalid message format*
  Internal code used to send a bad request error code. The provided message format is invalid (i.e. a Binary format has been specified for a system that only admits XML)

- **HAND-007:** *Invalid signature*
  Received message has invalid signature (invalid certificate, invalid reference, etc.). Check additional error message for details.

- **HAND-008:** *Signature syntax error.*
  The provided signature document is not valid against schema or there is an element with invalid value (invalid certificate chain, invalid certificate serial number, etc.) Check additional error message for details.

- **HAND-009:** *Unable to sign message.*
  System cannot sign the provided message. Check if the message is a valid xml.

- **HAND-010:** *Server returns FAULT message.*
  Server has returned a FAULT message instead of a Response Message. Check error details.

- **HAND-011:** *Response message has no payload or payload is empty.*
  The response message has no payload or the given one is empty.

- **HAND-012:** *Response has invalid header elements: [verb=?][noun=?] expected: [verb=?][noun=?]*
  The server response has invalid header elements that do not correspond with the expected ones. Thus, response is rejected.

- **HAND-013:** *Client do not trust server identity. Check client's trust store, check URL (do not use IP).*

  The credentials of the certification authorization used by the server do not match any of the existent entries available in the trust store. Check that the URL used is the one you want. Add to your trust store the server's credentials.

- **HAND-014:** *The identity of the server certificate does not match the configured URL.*
  Server has presented an identity "www.examplez.com" that do not match the configured (used) URL "www.example.com". Thus, client does not trust server and communication is aborted.

- **HAND-015:** *The configured URL is unknown and cannot connected to it. Check URL, check DNS.*
  The client cannot reach the configured URL. Check that there is no mistakes in the configured URL. Check your system's DNS configuration.

- **HAND-016:** *The requested service does not exist on the server. Check configured URL. (HTTP=404, Not found).*
  You have requested a web service URI that does not exists in the server. Check the URL.

- **HAND-017:** *You do not have permission to access the web services on the server. (HTTP=403, Forbidden).*
  Your system can reach the server, but your request has been rejected because your credentials (certificate). Check that you are using the correct certificate. Check that you are invoking the correct service.

- **HAND-018:** *You are unauthorized to perform such request. (HTTP=401, Unauthorized).*
  You are not authorized to send the request. Check your certificate.

- **HAND-019:** *The server has received your request, but it has not sent a response in return (HTTP=200, Accepted).*
  Your message was received by the server, but the server has returned an empty (or no [2]) response with status accepted. Ask server administrator for further details.

- **HAND-020:** *The server has refused the connection. Check connectivity settings, check firewall settings.*
  The server exists (see HAND-016) but has refuse your connection. There must be a connectivity problem that prevents the server accepts the connection. Check firewalls settings, check internet settings. It also be possible that the server is shut down.

Test ID: GN-01

Objective: Verify that the server handles a request by an invalid (but well formed) IEC 61968-100 message.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized and valid certificate.

Action 1: Use a web service client to send an invalid IEC 61968-100 message.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
                <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                        <mes:Header/>
                        <mes:Request/>
                </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns a 400 (Bad request) error message. Fault error message with code HAND-002 can also be used, note that in this case the http status should not be 500. System administrator should access to the received request message or soap message.

Repeat the test changing the request message (i.e. add additional mandatory elements, remove others, change the element's order, repeat elements, etc.)

Test ID: GN-02

Objective: Verify that the server handles a request by an invalid (but well formed) IEC 61968-100 message and unauthorized user. The server should perform security validations before message's validations.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an unauthorized certificate.

Action: Repeat GN-01 actions.

Result: Same result as test SE-02.

Test ID: GN-03

Objective: Verify that the server handles a request by an unspecified noun and verb combination according to [2].

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate.

Action 1: Use a web service client to send a request by an unspecified noun and verb combination.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
                <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                        <mes:Header>
                                <mes:Verb>get</mes:Verb>
                                <mes:Noun>Piski</mes:Noun> <!-- example -->
                        </mes:Header>
                        <mes:Request>
                                <mes:Option>
                                        <mes:name>Piski_ID</mes:name>
                                        <mes:value>42</mes:value>
                                </mes:Option>
                        </mes:Request>
                </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns a 400 (Bad request) error message. Fault error message with error HAND-005 can also be used. System administrator should access to the received request message or soap message.

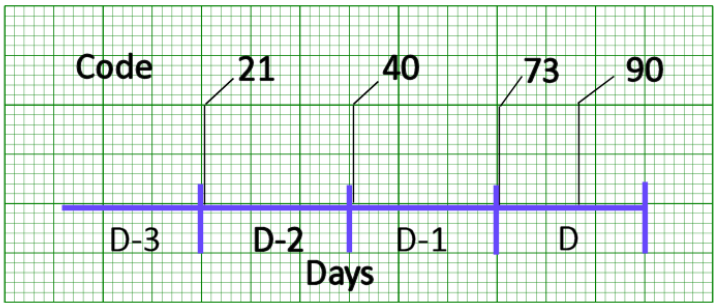
Repeat the test several times changing the noun and verb values.

Note: According to [3] the possible values for Verb are: `cancel`, `canceled`, `change`, `changed`, `create`, `created`, `close`, `closed`, `delete`, `deleted`, `get`, `reply`, `execute`, `executed`. You must use one of these values in this test, otherwise you are performing test GN-01 instead. Test for invalid combinations for verbs "`create`" and "`modify`" will be covered in put operation chapter.

4 List

General notes:

- All interchanged messages should be valid against schema (check that the returned `MessageList` has all the mandatory elements according to [2] 5.1.3 "*Service Response"*)
- List operation with filter code 0 does not mean that all messages available in the server should be included in the response list message. According to [2] "*for optimization purposes, only messages available since the 00:00 of D-1 are guaranteed to be included in the response list"*. (See also [2] 5.4.4 "*Functional requirements"*, `NumberOfDaysForLowCodeInListResponse`). In this case, a list operation with filter code 0 could return no messages (see clarification below).
- List operation with filter code will not break the `MaxNumMessageInListResponse` limit. If necessary, the server will returns the maximum allowed number of messages that have the greatest code value.

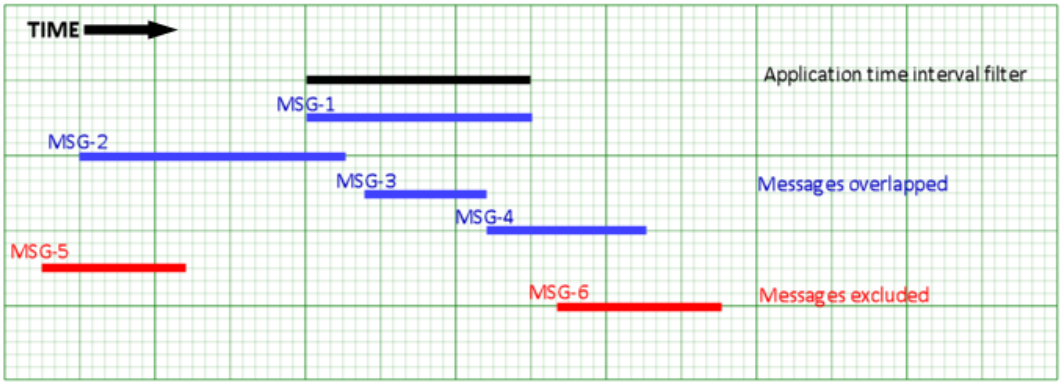


The image shows the evolution of the message code along the days. The highest message code (the latest) is 90.

  - If a user requests a list filter by code 0 and the server implements the optimization with D-1, the request will be internally translated to request by code=40.
  - If the latest available message for the current user has code=23 (there are more messages, but they are not available according to confidentiality rules) and the user requests a list filter by code 0, the response message will be an empty list (40 > 23)
  - If a user is entitled to retrieve all messages and requests a list filter by code 0 and the server implements the optimization with D-1 and has a limit for the maximum number of messages in the list response set to 10, then the request will contain the latest 10 messages (codes from 80 to 90). Note that messages from 40 to 79 will not be included in the list.

- List operation request could be rejected if the time filter spans more than a certain amount of days. (See [2] 5.4.4 Functional requirements, `MaxApplicationTimeIntervalInDaysInLisRequest`, `MaxServerTimeIntervalInDaysInLisRequest`). In this case, you should repeat the test reducing the amount of days that the filter spans.
- List operation request could also be rejected because other limits: `MaxListRequestPerMinute`.
- Additional test cases must be added if the server has additional limits for the list operation.

- List operation for server timestamp interval type will return messages that their server timestamp are spanned by the provided time interval filter. When application interval time type is used, the returned list will contain messages that have an application time interval overlapped with the provided time interval filter.



- List operation with date filter (Server timestamp or application date) can break the `MaxNumMessageInListResponse` limit. In this case, it is necessary to add additional filters (i.e. `MsgType`) or reduce the amount of time that the used filter spans.
- The owner filter applies only in systems where a user is allowed to access messages from other users (for instance if the server implements a "root" user). Otherwise, the `Owner` element will have always the same value. Tests where the use of Owner is required are intended to be performed by the server's administrator because it is necessary to know certain information (user's codes, reference data of third party messages, etc.) that should not be accessible by clients.
- List operation must honor confidentiality rules. If a user has no rights in the system, he / she will receive an empty list message in any case. (see [2] 5.1.4 "*Functional requirements"*)
- If the server receives a repeated set of filters (two different codes, two different messages types, etc.) the request must be rejected stating what filter is repeated. Otherwise, if the request is performed, the user will not be able to know which filter was used to build the response and which was ignored.
- If the server receives an unspecified filter for the operation (i.e. `MessageSubGroup`, or any random string passed as parameter) the server shall reject the request stating which filter is unknown according to the operation. Otherwise, if the request is performed, the user will not be able to know whether the parameter was used or not.
- According to [2] the request for list operation is not signed. The signature in the request will be ignored even if it is invalid.
- Because the date format elements are part of the underlying message schema, tests concerning invalid date format are covered by GN-01.

Proposed error codes:

- **LST-001:** *Invalid parameters. Code must be a positive integer value.*
  The provided code is a negative value. Code must be a positive integer number.

- **LST-002:** *Invalid operation parameters. Code must be an integer value.*
  The provided code is not a number. Code must be a positive integer number.

- **LST-003:** *Invalid operation parameters. EndTime cannot precede StartTime.*
  The provided time interval is incorrect, `EndTime` precede `StartTime`.

- **LST-004:** *Invalid operation parameters. Time interval cannot span more than ? days.*
  Server has a limit for the number of days that the provided time interval can span. Reduce the number of days and repeat the request.

- **LST-005:** *Invalid operation parameters. You must provide either Code or StartTime and EndTime time interval values*
  Check that you have provided a code or time interval filter but not both. If you have provided a time interval filter check that both values `StartTime` and `EndTime` were provided.

- **LST-006:** *Database read failed.*
  Server is unable to read the database. Contact the system administrator.

- **LST-007:** *The operation returns more than ? messages. Please, use a smaller time interval value or add more filters.*
  Server has a limit for the number messages that a response can have and the number of available messages that fulfill the provided filter is greater than such limit. Reduce the number of days that the provided time interval has or add additional filters in order to reduce the number of messages.

- **LST-008:** *Unable to create list response.*
  Server is unable to create a list response because an unexpected condition. Contact the system administrator.

- **LST-009:** *Invalid operation parameters. IntervalType must be one of Application, Server*.
  The IntervalType value provided is invalid. Check the possible values in the error message.

- **LST-010:** *Invalid operation parameters. ?*
  The provided set of parameters is invalid (repeated parameters, invalid date formats, etc.) Check error message for details.

- **LST-011:** *Unknown parameter for list operation: ?*
  The provided parameter is not valid (unspecified according to [2]) for list operation. Remove the parameter and send the request.

- **LST-012:** *Received message list has an invalid entry with no message code.*
  The client has received an invalid list response where at least one of the entries has no message code. The client could also reject the whole response because it is not valid against schema (message code is mandatory)

- **LST-013:** *Received message list has an invalid list entry with no message identification. Message info: [code=?].*
  The client has received an invalid list response where at least one of the entries has no message identification. The list entry's code is shown in the error. The client could also reject the whole response because it is not valid against schema (message's identification is mandatory)

- **LST-014:** *Received message list has an invalid list entry with no message type. Message info: [code=?][id=?].*
  The client has received an invalid list response where at least one of the entries has no message type. The list entry's code and identification are shown in the error. The client could also reject the whole response because it is not valid against schema (message type is mandatory)

- **LST-015:** *Received message list has an invalid list entry with no start application time interval. Message info: [code=?][id=?][type=?]*
  The client has received an invalid list response where at least one of the entries has no application time interval. The list entry's code, identification and message type are shown in the error. The client could also reject the whole response because it is not valid against schema (application time interval is mandatory)

- **LST-016:** *Received message list has an invalid list entry with no server timestamp. Message info: [code=?][id=?][type=?].*
  The client has received an invalid list response where at least one of the entries has no server timestamp. The list entry's code, identification and message type are shown in the error. The client could also reject the whole response because it is not valid against schema (server timestamp is mandatory)

- **LST-017:** *Received message list has an invalid list entry with no owner. Message info: [code=?][id=?][type=?].*
  The client has received an invalid list response where at least one of the entries has no owner element. The list entry's code, identification and message type are shown in the error. The client could also reject the whole response because it is not valid against schema (message's owner is mandatory)

- **LST-018:** *Received message list is invalid. ?*
  The client has received an invalid list response so that it is unable to process the elements. Further information about the error nature is included. Client can use this error instead errors LST-012 to LST-017.

- **LST-019:** *Unable to get message list from the received payload.*
  The client has received an invalid response and is unable to create a proper message list (there is no payload or the payload contains a message that is not a list). Check received message. Contact server's administrator.

- **LST-020:** *User has exceeded list operation limits. User is temporarily blocked.*
  User has exceeded the allowed number of list operation requests per minute (`MaxListRequestPerMinute`) further list request will be rejected.

Test ID: LS-01

Objective: Verify that the server / client handles the list operation by code filter properly.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by filter code.

Action 1: Use a web service client to send a list request by code 0.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>0</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with the available list of messages. Client processes the received list.

Action 2: Repeat the request using the highest code number received in the `MessageList`

Result 2: Server returns a `MessageList` with documents which code is greater than the one specified. If there are no new messages in the server (the most common situation), the `MessageList` must be empty. Client processes the new list.

Action 3: Let X the highest code number received in the `MessageList` (X>0) and let Y=X-1. Repeat the request using Y as code.

Result 3: Server returns a `MessageList` with the document with code X. The server could also return more messages (besides the one with code "X") if they are newer. Check that all received messages have a code > Y. Client processes the new list.

Test ID: LS-02

Objective: Verify that the server / client handles the list operation by code with optional filter `MsgType`.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. The test LS-01 was performed successfully and there is at least one message available for testing. Client supports request by `MsgType` filter.

Action 1: Use a web service client to send a list request by code 0 and `MsgType` set to one of the message types listed in LS-01.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>0</mes:value>
                    </mes:Option>
                    <mes:Option>
                        <mes:name>MsgType</mes:name>
                        <mes:value>ExampleType</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages of the given type. Client processes the received list.

Action 2: Repeat the request using a different code value. Let X the used code.

Result 2: Server returns a `MessageList` document with only messages of the given type which code is greater than X. Client processes the received list.

Action 3: Repeat the request using an inexistent message type.

Result 3: Server returns an empty `MessageList`.

Test ID: LS-03

Objective: Verify that the server / client handles the list operation by code with optional filter owner.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. The test LS-01 was performed successfully and there is at least one message available for testing. Client supports request by owner filter.

Action 1: Use a web service client to send a list request by code 0 and owner filter set to one of the owners listed in LS-01.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>0</mes:value>
                    </mes:Option>
                    <mes:Option>
                        <mes:name>Owner</mes:name>
                        <mes:value>ExampleOwner</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages for the given owner.

Action 2: Repeat the request using an inexistent owner.

Result 2: Server returns an empty `MessageList`.

Action 3: Repeat the request using the identification of other (existent) user whose messages should not be accessible for the current user.

Result 3: Server returns an empty `MessageList`.

Test ID: LS-04

Objective: Verify that the server / client handles the list operation by code with optional filter `MessageIdentification`.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. The test LS-01 was performed successfully and there is at least one message available for testing. Client supports request by `MessageIdentification` filter.

Action 1: Use a web service client to send a list request with code 0 and `MessageIdentification` filter set to one of the message's identification listed in LS-01.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>0</mes:value>
                    </mes:Option>
                    <mes:Option>
                        <mes:name>MessageIdentification</mes:name>
                        <mes:value>ExampleID</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages which identification is the one provided. Client processes the received list.

Action 2: Repeat the request removing some characters at the end of the identification value and add a wildcard "*" at the end (`Example*`).

Result 2: Server returns a `MessageList` document with all the available messages which identification starts with the given pattern. Client processes the received list.

Action 3: Repeat the request removing some characters at the beginning of the identification value and add a wildcard "*" (`*mple`). Clients which message identification is not editable cannot perform this test.

Result 3: Server returns a `MessageList` document with all the available messages which identification ends with the given pattern. Client processes the received list

Action 4: Repeat the request replacing some characters in the middle of the identification by a wildcard "*" (`Ex*le`).

Result 4: Server returns a `MessageList` document with all the available messages whose identification starts and ends with the given pattern.

Action 5: Repeat the request replacing several characters by a wildcard "*" (`E*x*le`).

Result 5: Server returns a `MessageList` document with all the available messages whose identification starts and ends with the given pattern.

Action 6: Repeat test 1-5 with a different code.

Result 6: Check results 1-5. Check that all the received messages have a code greater than the one provided.

Notes: The server must implement the wildcard feature in `MessageIdentification` filter (It is mandatory according to [2]). The message version (if applies in the system) is not covered by this filter.

Test ID: LS-05

Objective: Verify that the server / client handles the list operation by server timestamp filter properly.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client. The amount of available messages will not break the limit `MaxNumMessagesInListResponse` if this is implemented.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by server timestamp. There are no mistakes in the filter (i.e. correct time format, `EndTime` > `StartTime`, etc.)

Action 1: Use a web service client to send a list request with server timestamp filter.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>IntervalType</mes:name>
                        <mes:value>Server</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with the available list of messages that fulfill the filter. All received messages fulfill the given time filter. Client processes the received list.

Action 2: Repeat the test with different (valid) time interval.

Result 2: Check that all received messages fulfill the given time filter.

Test ID: LS-06

Objective: Verify that the server / client handles the list operation by server timestamp with optional filter `MsgType`.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by server timestamp and optional filter `MsgType`. The test LS-05 was performed successfully and there is at least one message available for testing.

Action 1: Use a web service client to send a list request by server timestamp filter and `MsgType` set to one of the message types listed in LS-05.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                            <mes:name>IntervalType</mes:name>
                            <mes:value>Server</mes:value>
                    </mes:Option>
                    <mes:Option>
                            <mes:name>MsgType</mes:name>
                            <mes:value>ExampleType</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages of the given type. Check that all received messages fulfill the given time filter. Client processes the list.

Action 2: Repeat the request using a different timestamp interval (There must be messages available for such time interval)

Result 2: Server returns `MessageList` document with only messages of the given type. Check that all received messages fulfill the given time filter. Client processes the list.

Action 3: Repeat the request using an inexistent message type.

Result 3: Server returns an empty `MessageList`.

Test ID: LS-07

Objective: Verify that the server / client handles the list operation by server timestamp with optional filter `Owner`.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by server timestamp and optional filter owner. The test LS-05 was performed successfully and there is at least one message available for testing. For action 3 at least two messages associated with two different owners are needed. Action 3 is intended only for systems where confidentiality rules applies.

Action 1: Use a web service client to send a list request by server timestamp and owner filter set to one of the message's owner listed in LS-05.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                        <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                        <mes:Option>
                            <mes:name>IntervalType</mes:name>
                            <mes:value>Server</mes:value>
                        </mes:Option>
                        <mes:Option>
                            <mes:name>Owner</mes:name>
                            <mes:value>ExampleOwner</mes:value> <!-- example -->
                        </mes:Option>
                </mes:Request>
        </mes:RequestMessage>
  </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages for the given owner. Check that all received messages fulfill the given time filter. Client processes the list.

Action 2: Repeat the request using an inexistent owner.

Result 2: Server returns an empty `MessageList`.

Action 3: Repeat the request using the identification of other (existent) owner code whose messages should not be accessible for the current user. The server administrator must provide this identification.

Result 3: Server returns an empty `MessageList`.

Test ID: LS-08

Objective: Verify that the server handles the list operation by server timestamp with optional filter `MessageIdentification`.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by server timestamp and optional filter `MessageIdentification`. The test LS-05 was performed successfully and there is at least one message available for testing.

Action 1: Use a web service client to send a list request with server timestamp and `MessageIdentification` filter set to one of the message's identification listed in LS-05.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>IntervalType</mes:name>
                        <mes:value>Server</mes:value>
                    </mes:Option>
                    <mes:Option>
                        <mes:name>MessageIdentification</mes:name>
                        <mes:value>ExampleID</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages whose identification is the one provided. All received messages fulfill the given time filter. Client processes the list.

Action 2: Repeat the request removing some characters at the end of the identification value and add a wildcard "*" at the end (`Example*`).

Result 2: Server returns a `MessageList` document with all the available messages whose identification starts with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 3: Repeat the request removing some characters at the beginning of the identification value and add a wildcard "*" (`*mple`).

Result 3: Server returns a `MessageList` document with all the available messages whose identification ends with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 4: Repeat the request replacing some characters in the middle of the identification by a wildcard "*" (`Ex*le`).

Result 4: Server returns a `MessageList` document with all the available messages whose identification starts and ends with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 5: Repeat the request replacing several characters by a wildcard "*" (`E*x*le`).

Result 5: Server returns a `MessageList` document with all the available messages whose identification starts and ends with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 6: Repeat test 1-5 with a different timestamp value.

Result 6: Check results 1-5

Notes: The system must implement the "wildcard" feature in the `MessageIdentification` filter (It is mandatory according to [2]). The message version (if applies) is not covered by this filter.

Test ID: LS-09

Objective: Verify that the server / client handles the list operation by application date filter properly.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client. The amount of available messages will not break the limit `MaxNumMessagesInListResponse` if this is implemented.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by application date. There are no mistakes in the filter (i.e. correct time format, `EndTime` > `StartTime`, etc.)

Action 1: Use a web service client to send a list request by application time interval filter.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>IntervalType</mes:name>
                        <mes:value>Application</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with the available list of messages that fulfill the given time filter. Client processes the list.

Action 2: Repeat the test with different (valid) time interval

Result 2: Check that all received messages fulfill the given time filter. Client processes the list.

Action 3: Repeat the test removing `IntervalType` option. See notes.

Result 3: Check that the received `MessageList` remains the same without this parameter. Client processes the list.

Notes: According to [2] 5.1.2 Service Request, the default value for date time interval is filtering by application date, so that `IntervalType` parameter can be omitted.

Test ID: LS-10

Objective: Verify that the server / client handles the list operation by application date with optional filter `MsgType`.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by application date and `MsgType` filters. The test LS-09 was performed successfully and there is at least one message available for testing.

Action 1: Use a web service client to send a list request by application date and `MsgType` set to one of the message types listed in LS-09.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                            <mes:name>MsgType</mes:name>
                            <mes:value>ExampleType</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages of the given type. Check that all received messages fulfill the given time filter. Client processes the list.

Action 2: Repeat the request using a different timestamp interval (There must be available messages for such time interval)

Result 2: Server returns `MessageList` document with only messages of the given type. Check that all received messages fulfill the given time filter. Client processes the list.

Action 3: Repeat the request using an inexistent message type

Result 3: Server returns an empty `MessageList`.

Test ID: LS-11

Objective: Verify that the server handles the list operation by application date with optional filter owner.

Server Preconditions: Server is configured and running. Client supports request by application date and owner filters. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. The test LS-09 was performed successfully and there is at least one message available for testing.

Action 1: Use a web service client to send a list request by application date and owner. Set the owner filter to one of the message's owner listed in LS-09.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>Owner</mes:name>
                        <mes:value>ExampleOwner</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
        </mes:RequestMessage>
  </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages for the given owner. Check that all received messages fulfill the given time filter. Client processes the list.

Action 2: Repeat the request using an inexistent owner

Result 2: Server returns an empty `MessageList`.

Action 3: Repeat the request using the identification of other (existent) user whose messages should not be accessible for the current user.

Result 3: Server returns an empty `MessageList`.

Test ID: LS-12

Objective: Verify that the server / client handles the list operation by application date with optional filter `MessageIdentification`.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by application date and `MessageIdentification` filters. The test LS-09 was performed successfully and there is at least one message available for testing.

Action 1: Use a web service client to send a list request with application time interval filter and `MessageIdentification` filter set to one of the message's identification listed in LS-09.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>MessageIdentification</mes:name>
                        <mes:value>ExampleID</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `MessageList` document with only messages which identification is the one provided. All received messages fulfill the given time filter. Client processes the list.

Action 2: Repeat the request removing some characters at the end of the identification value and add a wildcard "*" at the end (`Example*`).

Result 2: Server returns a `MessageList` document with all the available messages which identification starts with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 3: Repeat the request removing some characters at the beginning of the identification value and add a wildcard "*" (`*mple`).

Result 3: Server returns a MessageList document with all the available messages which identification ends with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 4: Repeat the request replacing some characters in the middle of the identification by a wildcard "*" (`Ex*le`).

Result 4: Server returns a `MessageList` document with all the available messages whose identification starts and ends with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 5: Repeat the request replacing several characters by a wildcard "*" (`E*x*le`).

Result 5: Server returns a `MessageList` document with all the available messages whose identification starts and ends with the given pattern. All received messages fulfill the given time filter. Client processes the list.

Action 6: Repeat test 1-5 with a different application date value.

Result 6: Check results 1-5

Test ID: LS-13

Objective: Verify that the server handles the list operation by mutually exclusive filters properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Use a web service client to send a list request by time interval and code filter.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-12T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-14T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>0</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that both filters are mutually exclusive (error code LST-005).

Test ID: LS-14

Objective: Verify that the server handles the list operation without filters.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action 1: Use a web service client to send a list request without filters.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that there is no request (error code LST-010).

Action 2: Repeat the request adding an element “Request” to the message:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                 <mes:Request/>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 2: Server returns `FaultMessage` stating that there is no filter (error code LST-005).

Test ID: LS-15

Objective: Verify that the server handles the list operation by invalid code properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action 1: Use a web service client to send a list request with a negative code filter.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>-10</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that code value must be positive (error code LST-001).

Action 2: Replace the code numeric value by a random (no numeric) string.

Result 2: Server returns `FaultMessage` stating that code value must be an integer (error code LST-002)

Test ID: LS-16

Objective: Verify that the server handles the list operation by invalid time interval filter properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action 1: Use a web service client to send a list request with server timestamp filter where `EndTime` is preceding `StarTime`

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-14T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2015-06-12T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>IntervalType</mes:name>
                        <mes:value>Server</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that EndTime cannot precede StartTime (error code LST-003).

Action 2: Remove `IntervalType` option in the request. The request is now by application date.
Result 2: Server returns FaultMessage stating that `EndTime` cannot precede `StartTime` (error code LST-003).

Action 3: Use the Action 1 Request sample and remove `StartTime` element.
Result 3: Server returns `FaultMessage` stating that filter is incomplete or invalid (error code LST-005).

Action 4: Use the Action 1 Request sample and remove `EndTime` element.
Result 4: Server returns `FaultMessage` stating that filter is incomplete or invalid (error code LST-005).

Action 5: Use the Action 1 Request removing `StartTime` and `IntervalType` option.
Result 5: Server returns `FaultMessage` stating that filter is incomplete or invalid (error code LST-005).

Action 6: Use the Action 1 Request removing `EndTime` and `IntervalType` option.
Result 6: Server returns `FaultMessage` stating that filter is incomplete or invalid (error code LST-005).

Test ID: LS-17

Objective: Verify that the server handles the list operation by an invalid `IntervalType` properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Use a web service client to send a list request with an invalid `IntervalType` value.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-11T22:00:00Z</mes:StartTime>
                    <mes:EndTime>2015-06-12T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>IntervalType</mes:name>
                        <mes:value>Sample String</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that the provided `IntervalType` is not recognized (error code LST-009).

Test ID: LS-18

Objective: Verify that the server handles the list operation by invalid time interval filters (according to the server limits) properly.

Server Preconditions: Server is configured and running. Server has a limit for the number of days that a time interval filter can span (`MaxApplicationTimeIntervalInDaysInListRequest`, `MaxServerTimeIntervalInDaysInListRequest`).

Client Preconditions: Client is configured with an authorized certificate. Client supports request by time interval and is able to set the interval's value.

Action 1: Use a web service client to send a list request by server timestamp where the number of days that the interval spans is greater than the specified limit.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-14T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2018-06-12T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>IntervalType</mes:name>
                        <mes:value>Server</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that the interval time cannot span more days that the one specified by the server limits (error LST-004).

Action 2: Repeat the test using "application date" `IntervalType` (remove `IntervalType` option in the request or choose such option in your client settings).

Result 2: Server returns FaultMessage stating that the interval time cannot span more days that the one specified by the server limits (error LST-004)

Notes: Server timestamp and application date limits are individual. The limits values can be different or inexistent.

Test ID: LS-19

Objective: Verify that the server rejects a list operation request that would return more messages than allowed.

Server Preconditions: Server is configured and running. Server has a limit for the number of messages that a list response can have (`MaxNumMessagesInListResponse`). There are enough messages in the server to perform the test.

Client Preconditions: Client is configured with an authorized certificate. Client support request by time interval.

Action: Use a web service client to send a list request by server timestamp filter (or application time interval, depending on the available messages) spanning the maximum days allowed.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:StartTime>2015-06-14T22:00:00Z</mes:StartTime> <!-- example -->
                    <mes:EndTime>2018-06-12T22:00:00Z</mes:EndTime>
                    <mes:Option>
                        <mes:name>IntervalType</mes:name>
                        <mes:value>Server</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that the number of available messages that fulfill the filter are greater than the declared limit (error code LST-007)

Test ID: LS-20

Objective: Verify that the server rejects new list request when the user has already executed a certain amount of list request per minute.

Server Preconditions: Server is configured and running. Server has a limit for the number of list request per minute than a user can execute (`MaxListRequestPerMinute`).

Client Preconditions: Client is configured with an authorized certificate. Client must be able to execute more requests per minute than the specified limit.

Action: Use a web service client to send more list request per minute than the specified limit.

Result: When the number of executed list operations reach the limit, the server will return a `FaultMessage` stating that the user has reached the maximum number of list request per minute instead of the message list (error code LST-020)

Test ID: LS-21

Objective: Verify that the server applies confidentiality rules excluding in the response messages that do not apply to the user according to said confidentiality rules.

Server Preconditions: Server is configured and running. Confidentiality rules apply in the application. There must be a set of messages available; at least one of them must not be accessible by the test user.

Client Preconditions: Client is configured with an authorized certificate. For test purposes, the client must know the reference data of the test message (i.e. code, timestamp and application date). Client is able to setup the code filter value.

Action 1: Use a web service client to send a list request by `Code` for the test message (use n-1 value where n is the test message's code).

Result 1: Server returns `MessageList` without the test message.

Action 2: Use a web service client to send a list request by application interval time (test message's application date must be spanned by the used application time interval).

Result 2: Same as Result 1.

Action 3: Use a web service client to send a list request by server timestamp (test message's server timestamp must be spanned by the used timestamp time interval).

Result 3: Same as Result 1.

Test ID: LS-22

Objective: Verify that the server handles the use of repeated filter elements properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action 1: Use a web service client to send a list request with two codes values.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>0</mes:value>
                    </mes:Option>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>100</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that the Code filter option was repeated, a generic “invalid filter message” is also acceptable (error code LST-010).

Action 2: Repeat the request using the others available filter twice or more times.

Result 2: All the requests are rejected as Result 1. If the error message indicates the parameter name, check that it matches the used repeated parameter.

Test ID: LS-23

Objective: Verify that the server handles the use of unknown filter elements properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Use a web service client to send a list request with an unspecified (according to [2]) parameter.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>MessageList</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>0</mes:value>
                    </mes:Option>
                    <mes:Option>
                        <mes:name>MessageSubType</mes:name> <!-- example -->
                        <mes:value>ATC</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that there is an unknown filter parameter indicating the name of said parameter a generic “invalid filter” is also acceptable (error code LST-011).

Test ID: LS-24

Objective: Verify that the server is not affected by SQL injection values passed as parameters. Because date format is validated by the underlying schema (GN-01), code validation was covered by LS-15 and `IntervalType` was covered by LS-17, we are going to use the other parameters to perform the test.

Server Preconditions: Server is configured and running. List with code = 0 returns at least one message.

Client Preconditions: Client is configured with an authorized certificate. Client is able to setup the filters values.

Action 1: Use a web service client to send a list request with a SQL injection expression in the element `MessageIdentification`

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header/>
    <soap:Body>
        <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
            <mes:Header>
                <mes:Verb>get</mes:Verb>
                <mes:Noun>MessageList</mes:Noun>
            </mes:Header>
            <mes:Request >
                <mes:Option>
                    <mes:name>Code</mes:name>
                    <mes:value>0</mes:value>
                </mes:Option>
                <mes:Option>
                    <mes:name>MessageIdentification</mes:name>
                    <mes:value>%' or 1=1 --'</mes:value> <!-- example -->
                </mes:Option>
            </mes:Request>
        </mes:RequestMessage>
    </soap:Body>
</soap:Envelope>
```

Result 1: Server returns an empty message list. Any other result is not acceptable (i.e. a FaultMessage or a non-empty list)

Action 2: Repeat the request using the parameter `MsgType`.

Result 2: Same result as Result 1.

Action 3: Repeat the request using the parameter `Owner`.

Result 3: Same result as Result 1.

Note: Different SQL injection expressions must be used. System administrator should check the correct application behavior. An empty list response could be created in case of database error even if this is not the desirable behavior.

5 Get

General notes:

- All interchanged messages should be valid against schema (i.e. check that the returned `ResponseMessage` has all the mandatory elements according to [2] 5.2.3 "`Service Response`") and the received signature must be valid.
- The validity of the received message (xml or binary) is out of the scope of this document.
- According to [3] "6.4 *Payload structures*" the server could return a XML message as binary if exceeds a predefined size. In this case, the XML is compressed using gzip algorithm and according to [2] the element `Format` could be omitted.
- If the system returns xml documents in either text or binary format depending on the document size, such size should be stated as a parameter limit (see [2] 5.4.4 "*`parameterLimits`*"). The parameter name `GetXmlAsBinaryThresholdInKb` described as "*The threshold size in Kilobytes above which an XML document is send as binary*" is recommended for this purpose.
- According to [2] "6.2. *Service mapping*" for binary payloads `Noun` element must be set to "`Compressed`".
- [2] Makes no assumption of mime type for binary messages. The mime type and any other aspects (i.e. compression algorithm, if used) are application specific.
- Get operation request could be rejected because of the limit `MaxGetRequestPerMinute`.
- In order to avoid bandwidth waste a system can limit the number of times that a user can download a message. The parameter name `MaxGetRequestsPerMessage` described as "*The maximum number of times that a message can be requested by user*" is recommended. Once the user has reached the number of allowed downloads, the message will not be available neither by the get operation nor list operation. This limit does not apply to "administrators" users.
- Additional test cases must be added if the server applies additional limits (not present in this document) for the get operation.
- In this document `Get` request with identification always uses the `MessageVersion` element. [2] States that such element is optional so that, test cases must be adapted to a specific (non-versioned) system removing said parameter in the request.
- If the server receives a repeated set of filters (two different codes, two different messages identifications, etc.) the request must be rejected stating what parameter name is repeated. Otherwise, if the request is performed, the user will not be able to know what parameter value was used for the request and what was ignored.
- If the server receives an unspecified parameter for the operation (i.e. `MessageClassification` or any random string passed as parameter), the server shall reject the request stating what parameter is unknown according to the operation. Otherwise, if the request if performed, the user will not be able to know whether the parameter was used.

- Systems that implement the optional parameter `Queue` should be tested from only one terminal. Otherwise, the test could be set as failed if two (or more) users execute the request in a short time (each user will receive a different message, consuming it from the queue).
- Because all the retrieved messages are signed, checksum values are not necessary.
- If the server returns the timestamp in each response ([2] recommendation) the signature value shall be different in each response even if the payload remains the same.
- According to [2] the request for get operation is not signed. The signature in the request will be ignored even if it is invalid.

Proposed error codes:

- **GET-001:** *Invalid parameters. Code must be a positive integer value.*
  The provided code is a negative value. Code must be a positive integer number.

- **GET-002:** *Invalid operation parameters. Code must be an integer value.*
  The provided code is not a number. Code must be a positive integer number.

- **GET-003:** *Invalid invocation parameters. You must provide either Code or MessageIdentification and MessageVersion values.*
  Your request contains both code and identification values. Use only one kind of filter.

- **GET-004:** *Invalid invocation parameters. You must provide Code or MessageIdentification and MessageVersion values.*
  Your request has no filter (code or identification).

- **GET-005:** *QUEUE filter is not supported.*
  The server does not implement the queue get operation.

- **GET-006:** *The requested message doesn't exist.*
  There is no message that fulfils the provided set of filters or, if it exists, the user is not entitled to retrieve such message.

- **GET-007:** *Database read failed.*
  The system is unable to read from its database because an unexpected condition. Contact system administrator.

- **GET-008:** Unable to create get response.
  Server is unable to create a get response because an unexpected condition. Contact the system administrator.

- **GET-009:** *Unable to filter the message payload.*
  Server is unable to provide a proper message according to confidentiality rules. Contact the system administrator.

- **GET-010:** *User has exceeded get operation limits. User is temporarily blocked.*
  User has exceeded the allowed number of get operation requests per minute (`MaxGetRequestPerMinute`) further get request will be rejected.

- **GET-011:** *Invalid operation parameter. ?*
  The provided set of parameters is invalid (repeated parameters, etc.) Check error message for details.

- **GET-012:** *Unknown parameter for get operation: ?*
  The provided parameter is not valid (unspecified according to [2]) for get operation. Remove the parameter and send the request.

- **GET-013:** *File read failed*
  The system is unable to read from its file system because an unexpected condition. Contact system administrator.

- **GET-014:** *The received message format is not supported [?]. Valid formats are [?]*
  Server has returned a message which message format is not supported by [2]. Contact server's system administrator.

- **GET-015:** *Unable to unzip received binary XML.*
  Server has returned a compressed xml payload but cannot be decompressed. Contact server's system administrator.

- **GET-016:** *Unable to read payload element.*
  Client is unable to read the payload element from the received response. Check response is valid and has a payload element.

- **GET-017:** *Queue value must be "?" not "?"*
  Given value for queue is not supported by [2].

- **GET-018:** *Server returns a binary message but did not provide file name.*
  Server has return a binary message but it did not provide a file name that is mandatory according to [2].

- **GET-019:** *MessageVersion must be a positive integer.*
  The provided MessageVersion value is not a positive integer.

- **GET-020:** *Server requires MessageVersion when MessageIdentification is provided.*
  In the context of the business, the server requires a MessageVersion when MessageIdentification is provided.

- **GET-021:** *The received message's size ? is greater that the maximun allowed ?*
  There is an agreement regarding the maximum message's size (see `MaxPayloadSizeInMBInGetResponse`) but the server has send a message which its size is greater than the one agreed.

Test ID: GT-01

Objective: Verify that the server /client handles the get operation by code parameter properly.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by code. Client has information about the message to be retrieved (i.e. message's code)

Action: Use a web service client to send a get request with a proper code.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>16890595</mes:value> <!-- example  -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns a `ResponseMessage` document with the requested message in the Payload element.

A) If the requested message is a binary file, the header noun is set to "`Compressed`". The message's content appears in the element `Compressed` inside the `Payload` Base 64 encoded. Element Format is set to "`BINARY`". The file name appears in the Reply element under the sub-element ID.

B) If the request message is an XML document, the header noun is set to the local name of the document's root element. The message's content appears in the element "any" inside the Payload (first child of Payload)

C) If the request message is an XML document which size is greater than a specific limit (`GetXmlAsBinaryThresholdInKb`), then the message can be received as a binary document. The header noun is set to "`Compressed`". The message's content appears in the element Compressed inside the Payload Base 64 encoded and compressed with gzip algorithm. Element Format is set to XML or could be not specified (according to [2] default value is XML). The document name appears in the Reply element under the sub-element ID.

In any case, the client shall be able to process the message (save to file, send to an external application, event creation, etc.)

Test ID: GT-02

Objective: Verify that the server handles the use of invalid code filter properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action 1: Use a web service client to send a get request with an invalid (negative) code value:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>-10</mes:value> <!-- example  -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that code value must be positive (error code GET-001).

Action 2: Replace the code numeric value by a non-numeric string.

Result 2: Server returns `FaultMessage` stating that code value must be an integer (error code GET-002).

Test ID: GT-03

Objective: Verify that the server /client handles the get operation by message identification and version properly.

Server Preconditions: Server is configured and running. There must be at least one message that could be retrieved by the client.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by identification and version. Client has information about the message to be retrieved (i.e. message's identification and version)

Action: Use a web service client to send a get request with a proper identification and version.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage  xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>MessageIdentification</mes:name>
                        <mes:value>TotalImbalanceVol_2015050302</mes:value> <!-- example -->
                    </mes:Option>
                    <mes:Option>
                        <mes:name>MessageVersion</mes:name>
                        <mes:value>1</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Same result as GT-01.

Test ID: GT-04

Objective: Verify that the server / client handles the get operation by `Queue` properly.

Server Preconditions: Server is configured and running. There must be at least two messages that could be retrieved by the client. The server implements the optional get parameter "`Queue`". See also GT-06.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by queue. Client has executed previously a list by code request in order to know the messages' order.

Action 1: Use a web service client to send a get request by queue.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Queue</mes:name>
                        <mes:value>NEXT</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Client gets the first message in the queue.

Action 2: Repeat action 1.

Result 2: Client gets the second message in the queue.

Test ID: GT-05

Objective: Verify that the server handles the get operation by queue with an invalid value properly.

Server Preconditions: Server is configured and running. The server implements the optional get parameter `Queue`.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Use a web service client to send a get request by queue using and invalid (random) parameter value.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Queue</mes:name>
                        <mes:value>OTHER</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that the given parameter is invalid (error code GET-017).

Test ID: GT-06

Objective: Verify that the server returns a proper response if it does not support the get operation by queue.

Server Preconditions: Server is configured and running. The server does not implement the optional parameter `Queue`.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by queue.

Action: Same as GT-04.

Result: Server returns `FaultMessage` stating that the queue filter is not supported (error code GET-005)

Test ID: GT-07

Objective: Verify that the server handles the use of invalid parameter properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test. Client has information about the message to be retrieved (i.e. message's code)

Action 1: Send a get request using both code and `MessageIdentification` parameters. Set both filters to a known message values.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage  xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>16638282</mes:value>     <!-- example -->
                    </mes:Option>
                    <mes:Option>
                        <mes:name>MessageIdentification</mes:name>
                        <mes:value>TotalImbalanceVol_2014080902</mes:value> <!-- example -->
                    </mes:Option>
                    <mes:Option>
                        <mes:name>MessageVersion</mes:name>
                        <mes:value>1</mes:value>  <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that that just one kind of filter must be used (error code GET-003).

Action 2: Repeat the request removing `Code` filter and adding `Queue` filter.

Result 2: The same result as 1.

Action 3: Repeat the request from Action 1 removing identification and version filter and adding `Queue` filter.

Result 3: The same result as 1.

Test ID: GT-08

Objective: Verify that the server handles the use of invalid filter (repeated elements) properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action 1: Send a get request using two different code values.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>16638282</mes:value> <!-- example -->
                    </mes:Option>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>16638283</mes:value>  <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that the `Code` filter option was repeated. A generic “invalid filter message” is also acceptable (error code GET-011)

Action 2: Repeat the request using the others available filters twice or more times.

Result 2: All the requests are rejected like Result 1. If the error message indicates the parameter name, check that it matches the repeated parameter name.

Test ID: GT-09

Objective: Verify that the server handles the use of invalid filter (unknown elements) properly.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Use a web service client to send a “get” request with an unspecified (according to [2]) parameter.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request>
                    <mes:Option>
                        <mes:name>RandomParameter</mes:name> <!-- example -->
                        <mes:value>RandomString</mes:value>
                    </mes:Option>
                    <mes:Option>
                        <mes:name>Code</mes:name>
                        <mes:value>11</mes:value>
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that there is an unknown filter parameter indicating the name of said parameter. A generic “invalid parameter” is also acceptable (error code GET-012).

Test ID: GT-10

Objective: Verify that the server /clients returns a proper message if the user ask for an inexistent message.

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate.

Action: Use a web service client to send a get request for an inexistent message (use a random message code or identification).

Result: Server returns FaultMessage stating that the requested message does not exists (error code GET-006). Client notifies the error.

Test ID: GT-11

Objective: Verify that the server applies confidentiality rules rejecting get request that applies to messages that cannot be retrieved by the user according to said confidentiality rules.

Server Preconditions: Server is configured and running. Confidentiality rules apply in the application. There must be a set of messages available; at least one of them must not be accessible by the testing user.

Client Preconditions: Client is configured with an authorized certificate. For test purposes, the client must know the reference data of the test message (i.e. code and message identification). Client should be able to choose the message to be retrieved without using the list operation.

Action 1: Use a web service client to send a get request by code for the test message.

Result 1: Same result as GT-10. For security reasons and according to [2] 5.2.4 "*Functional requirements*" this is the only admitted behavior (i.e. a message stating that the user is not entitled to retrieve the message is not acceptable). Client notifies the error.

Action 2: Use a web service client to send a get request by `MessageIdentification` and `MessageVersion` for the test message (use `MessageVersion` only if this element applies).

Result 2: Same result as result 1.

Test ID: GT-12

Objective: Verify that the server applies confidentiality rules returning to the client only the parts of the requested message that applies to the user according to said rules.

Server Preconditions: Server is configured and running. Confidentiality rules apply in the application. There must be at least one message which its whole content does not apply for the testing user (i.e. the user will receive only parts of the document). Server has only one copy of the message for all users (filter applies). If the server publishes several messages, one for each entity (instead of apply a filter) this test does not apply.

Client Preconditions: Client has available a set of authorized certificates. Each certificate represent a different company or entity.

Action 1: Use a web service client to send a get request for the test message.

Result 1: The user receives a partial document only with the elements that apply to the user.

Action 2: Repeat the test with a user who other parts of the document apply.

Result 2: The user receives a partial document only with the elements that apply to the user. This document is different for the one retrieved in Result 1, even if the code (or identification) of the message is the same.

Action 3: Repeat the test with an administrator user (if this role applies in the application).

Result 3: The user receives the whole message.

Test ID: GT-13

Objective: Verify that the server is not affected by SQL injection values passed as parameters. Because code validation was covered by GT-02 and invalid queue was covered by GT-05, we are going to use `MessageIdentification` for testing

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. Client supports request by identification and version.

Action: Use a web service client to send a get request with a SQL injection expression in the element `MessageIdentification`

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <mes:RequestMessage xmlns:mes="http://iec.ch/TC57/2011/schema/message">
                <mes:Header>
                    <mes:Verb>get</mes:Verb>
                    <mes:Noun>Any</mes:Noun>
                </mes:Header>
                <mes:Request >
                    <mes:Option>
                        <mes:name>MessageIdentification</mes:name>
                        <mes:value>%' or 1=1 --'</mes:value> <!-- example -->
                    </mes:Option>
                </mes:Request>
            </mes:RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that the requested message does not exists. Any other result is not acceptable (i.e. a `FaultMessage` with a database error message)

Note: Different SQL injection expressions must be used. System administrator should check the correct application behavior. A `FaultMessage` stating that the message does not exist could be covering a database error.

Test ID: GT-14

Objective: Verify that the client rejects responses with invalid signature.

Server Preconditions: Server is configured and running. Server is modified in order to create invalid signatures (mock server).

Client Preconditions: Client is configured with an authorized certificate.

Action: Use a web service client to send a valid get request.

Result: Server returns a response with the requested message but with an invalid signature. Client rejects the response with a message error.

Notes: Signature validation is not a straightforward task. You should test, at least, the following scenarios:

- Signature created by an untrusted certificate.
- Signature created by an expired (or not yet valid) certificate.
- Signature with invalid hash (the provided hash value is different to the calculated one).
- Signature with invalid key material (the certificate PEM encoded value is invalid)
- Signature with invalid algorithms (e.g. SHA-1 instead of SHA-256)

Test ID: GT-15

Objective: Verify that the server rejects new get request when the user has already executed a certain amount of get requests per minute.

Server Preconditions: Server is configured and running. Server has a limit for the number of get requests per minute than a user can execute (`MaxGetRequestPerMinute`). There are messages available to the client.

Client Preconditions: Client is configured with an authorized certificate. Client must be able to execute more requests per minute than the specified limit.

Action: Use a web service client to send more get requests per minute than the specified limit.

Result: When the number of executed get operations reach the limit, the server will return a `FaultMessage` stating that the user has reached the maximum number of get requests per minute instead of returning the requested message.

Note: In order to avoid DOS condition, the count of get request must be increased even if the user request an inexistent message.

Test ID: GT-16

Objective: Verify that the server hides (=makes it unavailable for list and get operations) a message when the user has already executed a certain amount of get requests for such message.

Server Preconditions: Server is configured and running. Server has a limit for the number of get request per message than a user can execute (`MaxGetRequestsPerMessage`). There is at least one message available to the client.

Client Preconditions: Client is configured with an authorized certificate. Client must be able to request the same message several times. The certificate has not “admin” or “root” roles assigned.

Action: Use a web service client to send more get request for a certain message than the specified limit.

Result: When the number of executed get operations reach the limit, the server will return a `FaultMessage` stating that the message does not exist.

Test ID: GT-17

Objective: Verify that the client rejects messages, which its size is greater than certain agreed size (`MaxPayloadSizeInMBInGetResponse`). See also PT-10

Server Preconditions: Server is configured and running. Server has at least one message available for the test client, which its size is greater than the agreed size.

Client Preconditions: Client is configured with an authorized certificate. Client has a limit regarding the maximum size that a received payload message can have.

Action: Client request the test message.

Result: Message is rejected by the client (error code GET-021). Client process the error.

6 QueryData

General notes:

- According to [2] `QueryData` is an optional operation so that, this chapter applies only in systems that implement this operation.
- Additional test cases must be added if the application implements more queries than the ones described in [2].
- All interchanged messages should be valid against schema (i.e. check that the returned `ResponseMessage` has all the mandatory elements according to [2] 5.4.3 "*Service Response*") and the received signature must be valid.
- If the server returns the timestamp in each response (As [2] recommends) the signature value will be different in each response even if the payload remains the same.
- All parameters used in the request shall be present in the response ([2] 5.4.3 "*Service response*"
- If the server receives a repeated set of filters names, the request must be rejected stating what filter name is repeated. Otherwise, if the request is performed, the user will not be able to know what filter value was used for the request and what was ignored.
- If the server receives an unspecified filter name for an operation and query, the server shall reject the request stating what filter is unknown according to the operation and query. Otherwise, if the request if performed, the user will not be able to know whether the parameter was used.
- According to [2] the request for `QueryData` is not signed. The signature in the request will be ignored even if it is invalid.
- Server can use [4] instead of `QueryData` to implement query functionality. Note that QueryData response contains the used parameters, the result data and the servers' signature. Thus, the response contains all necessary elements to meet the requirement of non-repudiation. [4] Requires the use of messages whereas `QueryData` performs dynamic queries.

Proposed error codes:

- **QRY-001:** *Invalid parameters. DataType value must be provided.*
  The provided set of parameters does not contain the mandatory parameter `DataType`.

- **QRY-002:** *Invalid parameters. Provided DataType value is not recognized.*
  The system has no query with the provided identification.

- **QRY-003:** *Invalid parameters. EndTime cannot precede StartTime.*
  The provided time interval is incorrect, EndTime precede StartTime.

- **QRY-004:** *Unable to create QueryData response*.
  Server is unable to create a query data response because an unexpected condition. Contact the system administrator.

- **QRY-005:** *Invalid parameters. Provided value ? for parameter ? is not recognized.*
  For the given parameter the provided value is not recognized (is not valid).

- **QRY-006:** *Database read failed.*
  Server is unable to read the database. Contact the system administrator.

- **QRY-007**: *Invalid parameters. Provided value ? for parameter ? must be a positive number.*
  For the given parameter the provided value must be a positive number.

- **QRY-008:** *The given parameters (? and ?) are mutually exclusive.*
  The given parameters are mutually exclusive. Repeat the request using either one of them.

- **QRY-009:** *Provided date parameter ? has invalid format: ? .*
  The provided date value for the given parameter has an invalid format.

- **QRY-010:** *Invalid operation parameters. ?*
  The provided set of parameters is invalid (repeated parameters, invalid date formats, etc.) Check error message for details.

- **QRY-011:** *Unknown parameter for query DataType ?: ?*
  The provided parameter is not valid for the given data type. Remove the parameter and send the request.

- **QRY-012:** *Cannot process response: ?*
  Client is unable to process the received payload. Check error detail.

- **QRY-013:** *User has exceeded Query operation limits. User is temporarily blocked.*
  User has exceeded the allowed number of query operation requests per minute (`MaxQueryRequestPerMinute`) further query requests will be rejected.

Test ID: QD-01

Objective: Verify Server Timestamp query.

Server Preconditions: Server supports `QueryData` optional operation.

Client Preconditions: Client supports `QueryData` optional operation.

Action: Send `QueryData` server timestamp query

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
                <Header>
                    <Verb>get</Verb>
                    <Noun>QueryData</Noun>
                </Header>
                <Request>
                    <Option>
                        <name>DataType</name>
                        <value>serverTimestamp</value>
                    </Option>
                </Request>
            </RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns a signed response with the timestamp. Note that the timestamp value appears in the header element. Client processes the response.

Test ID: QD-02

Objective: Verify list of data types query (`listOfDataTypes`)

Server Preconditions: Server supports `QueryData` optional operation.

Client Preconditions: Client supports `QueryData` optional operation.

Action: Send `QueryData` list of data types query

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
                <Header>
                    <Verb>get</Verb>
                    <Noun>QueryData</Noun>
                </Header>
                <Request>
                    <Option>
                        <name>DataType</name>
                        <value>listOfDataTypes</value>
                    </Option>
                </Request>
            </RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns a signed response with the list of supported data types. Client processes the response (typically displays it)

Test ID: QD-03

Objective: Verify optional parameter limits query (`parameterLimits`)

Server Preconditions: Server supports `QueryData` optional operation and the optional query `parameterLimits`

Client Preconditions: Client supports `QueryData` optional operation and the optional query `parameterLimits`

Action: Send `QueryData` Parameter Limits query

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
                <Header>
                    <Verb>get</Verb>
                    <Noun>QueryData</Noun>
                </Header>
                <Request>
                    <Option>
                        <name>DataType</name>
                        <value>parameterLimits</value>
                    </Option>
                </Request>
            </RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns a signed response with the list of parameter limits. Client processes the response.

Test ID: QD-04

Objective: Verify sever behavior when an invalid `DataType` is provided.

Server Preconditions: Server supports `QueryData` optional operation.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Send `QueryData` query with a random value for `DataType` (an inexistent query identification)

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
                <Header>
                    <Verb>get</Verb>
                    <Noun>QueryData</Noun>
                </Header>
                <Request>
                    <Option>
                        <name>DataType</name>
                        <value>random-string</value> <!-- example -->
                    </Option>
                </Request>
            </RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that the provided value is not valid (error code QRY-002).

Test ID: QD-05

Objective: Verify sever behavior when `DataType` is not provided.

Server Preconditions: Server supports `QueryData` optional operation.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action 1: Send `QueryData` query with no value for `DataType`.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
            <RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
                <Header>
                    <Verb>get</Verb>
                    <Noun>QueryData</Noun>
                </Header>
                <Request>
                    <Option>
                        <name>DataType</name>
                    </Option>
                </Request>
            </RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result 1: Server returns `FaultMessage` stating that `DataType` must be provided (error code QRY-001).

Action 2: Repeat the request removing the Option block.

Result 2: Server returns `FaultMessage` stating that `DataType` must be provided (error code QRY-001).

Test ID: QD-06

Objective: Verify that the server handles the use of repeated elements properly.

Server Preconditions: Server supports `QueryData` optional operation.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Send `QueryData` query with two values for `DataType`.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
	<soap:Header/>
	<soap:Body>
		<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
			<Header>
				<Verb>get</Verb>
				<Noun>QueryData</Noun>
			</Header>
			<Request>
				<Option>
					<name>DataType</name>
					<value>listOfDataTypes</value> <!-- example -->
				</Option>
				<Option>
					<name>DataType</name>
					<value>serverTimestamp</value> <!-- example -->
				</Option>
			</Request>
		</RequestMessage>
	</soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that the `DataType` parameter was repeated. A generic “invalid filter message” is also acceptable (error code QRY-010)

Test ID: QD-07

Objective: Verify that the server handles the use of unknown elements properly.

Server Preconditions: Server implements `QueryData` optional operation.

Client Preconditions: Client is configured with an authorized certificate. An ordinary client (not intended for testing) shall not be able to perform this test.

Action: Send `QueryData` query with an unspecified (according to [2]) parameter.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
                <RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
                        <Header>
                                <Verb>get</Verb>
                                <Noun>QueryData</Noun>
                        </Header>
                        <Request>
                                <Option>
                                        <name>DataType</name>
                                        <value>listOfDataTypes</value> <!-- example -->
                                </Option>
                                <Option>
                                        <name>Parameter</name>
                                        <value>Value</value> <!-- example -->
                                </Option>
                        </Request>
                </RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Server returns `FaultMessage` stating that there is an unknown filter parameter indicating the name of said parameter. A generic “invalid filter” is also acceptable (error code QRY-011).

Note that the set of allowed parameters could be different for different queries.

Test ID: QD-08

Objective: Verify that the server is not affected by SQL injection values passed as parameters.

Server Preconditions: Server implements `QueryData` optional operation and additional queries that requires database use. [2] basic set of queries (server timestamp, list of queries and limits) do not use database.

Client Preconditions: Client is configured with an authorized certificate.

Action: Send `QueryData` query with a SQL injection expression.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
        <soap:Header/>
        <soap:Body>
                <RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
                        <Header>
                                <Verb>get</Verb>
                                <Noun>QueryData</Noun>
                        </Header>
                        <Request>
                                <Option>
                                        <name>DataType</name>
                                        <value>alert</value> <!-- example -->
                                </Option>
                                <Option>
                                        <name>AlertNumber</name>
                                        <value>%' or 1=1 --'</value> <!-- example -->
                                </Option>
                        </Request>
                </RequestMessage>
        </soap:Body>
</soap:Envelope>
```

Result: Depending on the system, there are two possible responses:

- A `MessageFault` stating that the parameter value is not valid.
- An empty `Query` response (only with the request parameters) because there is no data available for the given parameter.

Note: Different SQL injection expressions must be used. System administrator should check the correct application behavior. An empty query response could be created in case of database error even if this is not the desirable behavior.

Test ID: QD-09

Objective: Verify that the client reject responses with invalid signature.

Server Preconditions: Server implements `QueryData` optional operation. Server is modified in order to create invalid signatures.

Client Preconditions: Client is configured with an authorized certificate.

Action: Send a valid `QueryData` request.

Result: Server returns the query response with an invalid signature. Client rejects the response with a message error.

Notes: See GT-14 notes.

Test ID: QD-10

Objective: Verify that the server rejects new `QueryData` request when the user has already executed a certain amount of `QueryData` request per minute.

Server Preconditions: Server implements `QueryData` optional operation. Server is configured and running. Server has a limit for the number of `QueryData` request per minute than a user can execute (`MaxQueryRequestPerMinute`).

Client Preconditions: Client is configured with an authorized certificate. Client must be able to execute more `QueryData` requests per minute than the specified limit.

Action: Use a web service client to send more `QueryData` request per minute than the specified limit.

Result: When the number of executed `QueryData` operations reach the limit, the server will return a FaultMessage stating that the user has reached the maximum number of `QueryData` request per minute instead of the `QueryData` response (error code QRY-013)

7 Put

General notes:

- Tests of the operation `Put` are closely related to the business model that is implemented, which is out of the scope of this document. In order to perform the tests, there must have a valid set of messages that the server can handle.
- The following technical validations are not covered by this document:
    - Version handling. If document version applies (`version`), the server should check that:
        - The minimum version is 1.
        - A message with version "X" will be rejected if there is other previously processed message with the same identification (`mRID`) and version "Y" having $X \leq Y$.
    - Document structure. Xml payloads should be valid against their schema.
    - Time handling.
        - If the payload have an element for creation date (`createdDateTime`) and the current time precedes the given value, the message should be rejected. (This means that the received message will be created in the future)
        - The request must be also rejected if the message is received after a closure gate.
- Unlike the rest of the test cases covered by this document in this case, we cannot provide a complete (and usable) SOAP request because the signature and the message must be valid to the server.
- Put implementation must be aware of the different users identities that this operation handles:
    - Http certificate used to open the secure channel.
    - Certificate used for document signing.
    - Identity used in the payload (In European Market profile `sender_marketParticipant.mRID`)

  For instance, if a message's signature is valid but it is performed by a certificate different to the one used for transmission (https identity) and there is no relationship between them, the request must be rejected.

  Note that a hierarchical model can be implemented where a user (or company) is able to send messages on behalf other user (or company) in this case the server must know this relationship in order to avoid message rejection.
- According to [2] the implementation of the `Put` operation is mandatory. Nevertheless, it is possible to have a system that creates messages using internal sources. In this case the operation `Put` shall return a security error (forbidden, not authorized) for any request.
- Put operation does not use Request's `Option` elements. If sender uses these elements, the server shall ignore them.

- According to [2] communication must follow a synchronous pattern. Nevertheless, it is possible to build asynchronous systems. In this case, the server will return an empty payload document in response. According to [2] the header's noun element must be set to the local name of document that is inside the payload: empty in this case:

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
    <Header>
        <Verb>reply</Verb>
        <Noun/>
        <Timestamp>2016-01-05T08:33:44.467Z</Timestamp>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
            ....
        </Signature>
    </Header>
    <Reply>
        <Result>OK</Result>
    </Reply>
    <Payload/>
</ResponseMessage>
```

  This kind of response is also useful in case the received document has no possible acknowledgement (e.g. the input document was already an acknowledgement).

- As a general rule, all the received messages will be available for list and get operations with the following exceptions:
  - Messages received by users with invalid credentials or without credentials. The message does not reach the application level and is rejected before it could be stored.
  - Messages send by users that had broken the limit `MaxPutRequestPerMinute`
  - Messages with empty payload.
  - Circumstances where it makes impossible to extract the elements that conforms the message (such as message type or identification)
    - Messages that are not valid according to the [3] schema.
    - Messages which payload is not valid according to its schema.
    - Invalid xml-compressed payloads that cannot be decompressed.
  - Messages that according to [2] do not conform the "put" operation (e.g. list or get operation requests are not stored)
  - Messages sent by a user that is not entitled to send such message.
  - Messages with invalid signature.
  - Messages sent on behalf a third party where the sender has no relationship with such party.
  - Messages where the Noun does not correspond with the local name of the root element that is inside the payload.

  Despite the message will not be available for list and get operations, in order to ease audit work this must be stored in a separate location. This do not apply to the three first bullets of the previous list.

- [2] allows binary and xml payloads, but this does not mean the server should accept both. If according to the business rules only xml payload applies, binary request will be rejected (and vice versa). Note that xml-compressed payload must be treat as an xml payload.

- Server must store the message's signature along with the payload, even if the server has already validate that it is valid. To fulfil with non-repudiation, signature must be always present.
- Because [2] allows the use of binary messages, in case the documents' size make them unmanageable, it is possible to send them split into different messages (i.e. gzip, b2z compressed). So that, server and client must create as many parts as needed taking into account the maximum message size (`MaxPayloadSizeInMBInPutRequest`, `MaxPayloadSizeInMBInGetResponse`)

Proposed error codes:

- **PUT-001:** *Remote system is unable to process your message [?] and cannot give a detailed (human readable) reason why. Please ask system administrator.*
  Message processing has ended with an unexpected error. The administrator must provide details about the reasons.

- **PUT-002:** *System is currently processing a message for the same message type and application date. Please wait until the system provides a proper acknowledgement.*
  User sent a new message version when the previous one is still under processing. Do not send a new version until the previous one is processed.

- **PUT-003:** *Malformed request: [provided noun=?] [expected noun=?]*
  There is a mismatch between the provided noun and the message payload (expected noun).

- **PUT-004:** *Unable to process message [?] there is no message handler for [noun=?]. Check your client's URL.*
  The system does not handle messages for the given type. Check if your client is pointing at the correct system.

- **PUT-005:** *You have no rights on this message type [MsgType=?][Rights=?]*
  The system understands the request but you do not have the necessary role (permission) to send it.

- **PUT-006:** *Invalid signature. Details: ?*
  The system has rejected the request because the received signature is incorrect. Details about the rejection are provided in the error.

- **PUT-007***: The provided signature document is incorrect and cannot be validated. Details: ?*
  The system has rejected the request because the received signature is not valid against its schema.

- **PUT-008:** *There is no relationship among the identities [Sender=?][Signer=?][Owner=?]*
  There is a mismatch among the identities e.g. sender and signer certificates are different but there is no relationship between them.

- **PUT-009:** *Unable to store the message. Please, send a new document version. If the message persists, ask system administrator.*
  An unexpected error has prevented the system to store the message. If the message persist, ask the system administrator.

- **PUT-010:** *Invalid message format: ?*
  The request message has an invalid value for element “Format” e.g. an XML document with binary format or a format not supported according to [2]

- **PUT-011:** *Invalid binary payload.*
  The provided binary payload is invalid, e.g. a gzip payload that cannot be decompressed.

- **PUT-012:** *Message was rejected due to technical validations: ?*
  The request was rejected due to technical validations such as schema validation, date format, etc.

- **PUT-013:** *Received message has no payload or it is empty.*
  The received message (request or response) has no payload or there is no document inside it.

- **PUT-014:** *Cannot create request: ?*
  Client cannot create a proper request. Check client’s logs.

- **PUT-015:** *Cannot process response: ?*
  Client is unable to process the received payload. Check error details.

- **PUT-016:** *User has exceeded put operation limits. User is temporarily blocked*
  User has exceeded the allowed number of put operation requests per minute (`MaxPutRequestPerMinute`) further put requests will be rejected.

- **PUT-017:** *Invalid operation parameters. ?*
  The provided set of parameters is invalid. Check error message for details.

- **PUT-018:** *The received message's size ? is greater that the maximun allowed ?*
  There is an agreement regarding the maximum message’s size (see `MaxPayloadSizeInMBInPutRequest`) but the client has send a message which its size is greater than the one agreed.

Test ID: PT-01

Objective: Verify that the server / client handles the put operation properly.

Server Preconditions: Server is configured and running. Server implements put operation for at least one message type.

Client Preconditions: Client is configured with an authorized certificate. Client role (if applies) allows the user to send messages.

Action: Client sends a message to the server.

Result: Server returns:

A) If the server can process the message, it will return a response that follows the acknowledgement schema (for European Electricity Market context IEC 62325-451-1 must be used, other implementations can return other response according to their rules):

   a. Noun=Acknowledgement_MarketDocument.

   b. Result must be OK or FAILED according to the rejection / acceptance of the message.

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>Acknowledgement_MarketDocument</Noun>
    <Timestamp>2015-12-14T13:28:09.819Z</Timestamp>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    […]
    </Signature>
  </Header>
  <Reply>
    <Result>OK</Result>
  </Reply>
    <Payload>
      <ns0:Acknowledgement_MarketDocument [....]
```

B) In the same context of A), the server could return a XML message as binary if it exceeds a predefined size. In this case, the XML is compressed using gzip algorithm and according to [2] the element `Format` could be omitted.

   a. Noun=Compressed.

   b. Result must be OK or FAILED according to the rejection / acceptance of the message.

C) Where a proper Acknowledgement cannot be created due to technical validations (i.e. invalid signature) a FaultMsg will be returned. Note that FaultMessage always correspond to a “FAILED” status.

Test ID: PT-02

Objective: Verify that the server / client supports XML compressed payloads.

Server Preconditions: Server is configured and running. Server implements put operation for at least one message type. Server supports XML payloads.

Client Preconditions: Client is configured with an authorized certificate. Client role (if applies) can send messages to the server. Client is able to send compressed payloads.

Action 1: Client sends a XML compressed message. Format element must be unset or must be set to "XML".

Result 1: Same result as PT-01.

Action 2: Client sends a XML compressed message. Format element has any value but "BINARY".

Result 2: Server returns a 400 (Bad request) error message. Fault error message with code PUT-010 can also be used. The message is not available for list and get operations.

Action 3: Client sends an invalid XML compressed (the binary B64 string do not correspond to a valid gzip file). Format element is not set or is set to "XML".

Result 3: Server returns a 400 (Bad request) error message. Fault error message with code PUT-011 can also be used. The message is not available for list and get operations.

Test ID: PT-03

Objective: Verify that the server supports different roles per message.

Server Preconditions: Server is configured and running. Server implements put operation for at least two message types, each one associated with a different role.

Client Preconditions: Client is configured with an authorized certificate. Client role can only send messages of a certain type.

Action 1: Client sends a message, which he / she is not entitled to send. For this test, a message that is expected to be received by other entity must be used.

Result 1: Server returns an unauthorized message (401). The message is not available for list and get operations.

Action 2: Client sends a message, which he / she is entitled to retrieve (get, list) but he / she is not entitled to send (e.g. a publication or an acknowledgement)

Result 2: Same as Result 1

Test ID: PT-04

Objective: Verify that the server verifies the request's signature.

Server Preconditions: Server is configured and running. Server implements put operation for at least one message type. Server supports different roles per message.

Client Preconditions: Client is configured with an authorized certificate. Client role can send messages to the server. Client is able to create an incorrect signature.

Action: Client sends a message with an invalid signature.

Result: Server rejected the request returning a FaultMessage stating that the signature is not valid (error code PUT-006). The message is not available for list and get operations

Notes: See GT-14 notes.

Test ID: PT-05

Objective: Server checks the relationship among identities. There are three identities involved:

- Client certificate for 2-way https connection.
- Client certificate for document signing.
- Document sender in payload (sender_marketParticipant.mRID)

Server Preconditions: Server is configured and running. Server implements put operation for at least one message type.

Client Preconditions: Client is configured with an authorized certificate. Client role can send messages to the server. Client is able to use several certificates for signing and sending. Client can create messages for third parties (sender_marketParticipant.mRID different to its own user's code). For this test, client should have an authorized and existent third party certificate.

Action 1: Client sends a message which sender (sender_marketParticipant.mRID) is an existing entity but with no relationship with the client that is performing the test. Client uses its own certificate for signature and sending.

Result 1: Server returns a fault stating that there is no relationship among the entities (error code PUT-008). The message is not available for list and get operations.

Action 2: Client sends a message using his/her identity (sender_marketParticipant.mRID). Client uses different certificates for signing and sending. Each certificate belongs to different (with no relationship) entity.

Result 2: Same as Result 1.

Action 3: Repeat action 2 interchanging certificate usage (use now for signature the one you have used for sending and vice versa)

Result 3: Same as Result 1.

Test ID: PT-06

Objective: Verify that the server rejects new `Put` request when the user has already executed a certain amount of `Put` request per minute.

Server Preconditions: Server is configured and running. Server has a limit for the number of `Put` request per minute than a user can execute (`MaxPutRequestPerMinute`).

Client Preconditions: Client is configured with an authorized certificate. Client must be able to execute more `Put` requests per minute than the specified limit.

Action: Use a web service client to send more `Put` request per minute than the specified limit.

Result: When the number of executed `Put` operations reach the limit, the server will return a FaultMessage stating that the user has reached the maximum number of `Put` request per minute instead of an acknowledgement response (error code PUT-016)

Note: Requests rejected by this limit should not be stored at all because this could lead into a DOS situation.

Test ID: PT-07

Objective: Verify that the server does not make available for list and get operations invalid messages.

Server Preconditions: Server is configured and running. Server accepts messages.

Client Preconditions: Client is configured with an authorized certificate. Client can send invalid messages.

Action: Use a web service client to send a document that is not valid against its schema.

Result: Message is rejected and it is not available for list and get operations.

Note: Others situations where the input message must not be available for get and put has been already tested.

Test ID: PT-08

Objective: Verify that the client / server supports empty payloads.

Server Preconditions: Server is configured and running. Server accepts acknowledgement messages (Any kind of message that, for its nature, has no response in return) or has an asynchronous message processing.

Client Preconditions: Client is configured with an authorized certificate.

Action: Use a web service client to send a document that

Result: Message is processed / accepted and the server sends an empty response is return. The response message is not stored in server (it is not available for list and get operations). Client ends the communication without error.

Test ID: PT-09

Objective: Verify that the server rejects request with invalid Noun according to [2].

Server Preconditions: Server is configured and running.

Client Preconditions: Client is configured with an authorized certificate. Client can create invalid request, which its noun has no relationship with the payload (Noun must be set to the local name of the first element inside the payload)

Action: Use a web service client to send a document that has no relationship between the Noun and Payload

Result: Message is rejected and it is not available for list and get operations.

Test ID: PT-10

Objective: Verify that the server rejects messages which its size is greater than certain agreed size (`MaxPayloadSizeInMBInPutRequest`). See also GT-17

Server Preconditions: Server is configured and running. Server has a limit regarding the maximum size that a payload message can have.

Client Preconditions: Client is configured with an authorized certificate. Client can sent messages, which its size is greater than the maximum agreed size.

Action: Use a web service client to send a document that has no relationship between the Noun and Payload

Result: Message is rejected and it is not available for list and get operations (error code PUT-018). Client process the error.

8 Score tables

Server environment

Connectivity

- Public URL:
- Available SSL / TSL protocols:
- Bandwidth:
- Server certificate (SSL/TSL)
    - Subject:
    - Valid from __/__/____ to __/__/____
    - Signature algorithm:
    - Public key (algorithm and key size):
- Signature certificate
    - Subject:
    - Valid from __/__/____ to __/__/____
    - Signature algorithm:
    - Public key (algorithm and key size):

Web layer

- Operative system (name, version, patch level):
- Number of servers in the farm:
- Web Server application (name, version):

Application layer

- Operative system (name, version):
- Number of servers in the farm:
- Application server (name, version):
- Programming language (name, version):

Database layer

- Operative system (name, version, patch level):
- Number of serves in the farm:
- Database server (name, version):

Other

Please, write down any other relevant information concerning the environment.

Application limits

Fill in the set of application's limits. Leave blank if the limit does not apply. Add on the list any additional limit that the application implements.

Limit name	Value	
MaxNumMessagesInListResponse		IEC 62325-504
NumberOfDaysForLowCodeInListResponse		
MaxApplicationTimeIntervalInDaysInListRequest		
MaxServerTimeIntervalInDaysInListRequest		
MaxPayloadSizeInMBInPutRequest		
MaxGetRequestPerMinute		
MaxPutRequestPerMinute		
MaxListRequestPerMinute		
MaxQueryRequestPerMinute		
MaxMessageAgeInDays		
MaxDiffServerTimestampInSeconds		
GetXmlAsBinaryThresholdInKb		Additional
MaxGetRequestsPerMessage		
MaxPayloadSizeInMBInGetResponse		

Test results

Leave blank if the test does not apply in your system (e.g. QueryData) or scope (server, client)

Test id	Description	#	☑	☒
SE-01	2-way SSL without client certificate			
SE-02	2-way SSL with unauthorized client			
SE-03	2-way SSL with unauthorized server			
SE-04	2-way SSL with client expired certificate			
SE-05	Uniqueness server certificate			
SE-06	Client aborts mismatched SSL connections			
GN-01	Request by invalid request	1		
		2		
		3		
		4		
GN-02	Request by invalid request and unauthorized client			
GN-03	Request by unsupported noun + verb	1		
		2		
		3		
		4		
LS-01	List by code	1		
		2		
		3		
LS-02	List by code and message type	1		
		2		
		3		
LS-03	List by code and owner	1		
		2		
		3		
LS-04	List by code and message identification	1		
		2		
		3		
		4		
		5		
		6		
LS-05	List by server timestamp	1		
		2		

LS-06	List by server timestamp and message type	1		
		2		
		3		
LS-07	List by server timestamp and owner	1		
		2		
		3		
LS-08	List by server timestamp and message identification	1		
		2		
		3		
		4		
		5		
		6		
LS-09	List by application time interval	1		
		2		
		3		
LS-10	List by application time interval and message type	1		
		2		
		3		
LS-11	List by application time interval and owner	1		
		2		
		3		
LS-12	List by application time interval and message identification	1		
		2		
		3		
		4		
		5		
		6		
LS-13	List by code and interval time			
LS-14	List by no filter	1		
		2		
LS-15	List by invalid code	1		
		2		
LS-16	List by invalid time interval (end precedes start)	1		
		2		
		3		
		4		
		5		

		6		
LS-17	List by unsupported IntervalType			
LS-18	List by invalid time interval (server limits)	1		
		2		
LS-19	List by time interval returns more messages than allowed			
LS-20	List by a greedy client			
LS-21	List honors confidentiality	1		
		2		
		3		
LS-22	List by repeated filter elements	1		
		2		
LS-23	List by unknown filter			
LS-24	List by SQL injection parameter	1		
		2		
		3		
GT-01	Get by code			
GT-02	Get by invalid code	1		
		2		
GT-03	Get by message identification			
GT-04	Get by queue	1		
		2		
GT-05	Get by invalid queue			
GT-06	Get by queue (queue not supported)			
GT-07	Get by code, message identification and queue	1		
		2		
		3		
GT-08	Get by repeated filter elements	1		
		2		
GT-09	Get by unsupported filter			
GT-10	Get an inexistent message			
GT-11	Get honors confidentiality (do not return other users' messages)	1		
		2		
GT-12	Get honors confidentiality (message filtering)	1		
		2		
		3		
GT-13	Get by SQL injection parameter			
GT-14	Client rejects invalid signature			

GT-15	Get by a greedy client			
GT-16	Get hides a message after certain number of requests			
GT-17	Get client rejects huge payloads			
QD-01	Query server timestamp			
QD-02	Query list of data types			
QD-03	Query parameter limits			
QD-04	Query by invalid data type			
QD-05	Query by no data type	1		
		2		
QD-06	Query by repeated filter elements			
QD-07	Query by unknown filter			
QD-08	Get by SQL injection parameter			
QD-09	Client rejects invalid signature			
QD-10	Get by a greedy client			
PT-01	Put supports XML / binary documents			
PT-02	Put supports XML compressed documents	1		
		2		
		3		
PT-03	Put supports different roles	1		
		2		
PT-04	Put checks signature			
PT-05	Put checks identities relationship	1		
		2		
		3		
PT-06	Put by greedy user			
PT-07	Put does not store invalid request			
PT-08	Put supports empty payloads			
PT-09	Puts rejects invalid nouns			
PT-10	Put rejects huge payloads			

Performance

Client and server performance will be computed separately.

Client performance is computed as the sum of the time need for creating and sending a request plus the time needed to process the server response. Note that the communication time and server response time is not included.

Server performance is measured as the time consumed since the request is received until the response is emitted.

Each test shall be repeated at least ten times. Best and worse marks will be ignored. The score is calculated as the average of the remaining marks.

Note that the number of messages (list) or the message's size (get, put) are also part of the performance mark. An empty system will always get better marks that a system that have to deal with thousands of messages. Get operation must be performed on available messages. Put operation must be performed with valid messages with different versions. In put test, mark whether the client / server uses xml-compressed.

To help on this purpose, it is strongly recommend that the application provides performance values along with the log entries.

Test id	Description	Mark	#Messages / Size
PR-01	List by code=0		
PR-02	List by code=0 and MessageIdentification (no wildcards)		
PR-03	List by code=0 and MessageIdentification (starting with wildcard)		
PR-04	List by server time interval (1 day)		
PR-05	List by server time interval (1 day) and MessageIdentification (no wildcards)		
PR-06	List by server time interval (1 day) and MessageIdentification (starting with wildcard)		
PR-07	List by server time interval (max. allowed number of days)		
PR-08	List by server time interval (max. allowed number of days) and MessageIdentification (no wildcards)		
PR-09	List by server time interval (max. allowed number of days) and MessageIdentification (starting with wildcard)		
PR-10	List by application date (1 day)		
PR-11	List by application date (1 day) and MessageIdentification (no wildcards)		
PR-12	List by application date (1 day) and MessageIdentification (starting with wildcard)		
PR-13	List by application date (max. allowed number of days)		
PR-14	List by application date (max. allowed number of days) and MessageIdentification (no wildcards)		

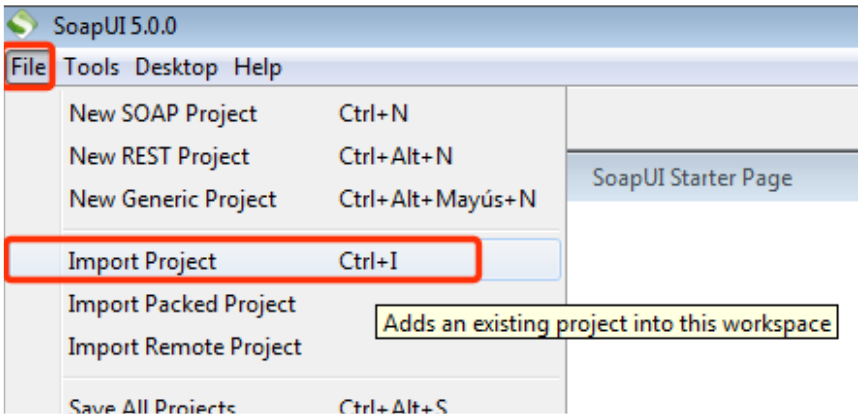
PR-15	List by application date (max. allowed number of days) and MessageIdentification (starting with wildcard)		
PR-16	Get by code (same code in each iteration)		
PR-17	Get by code (different message code in each iteration)		
PR-18	Get by message identification (same identification)		
PR-19	Get by message identification (different identification)		
PR-20	Get by queue		
PR-21	Query server timestamp		
PR-22	Put small message (<500k) [Compressed S-N]		
PR-23	Put medium message (>500k <2M) [Compressed S-N]		
PR-24	Put huge message (>2M) [Compressed S-N]		

9 Annex A: SOAP-UI

SOAP-UI permite las tareas de test de servicios web de forma visual permitiendo ejecutar las pruebas tanto de cliente como de servidor. Dispone de versiones Community y PRO

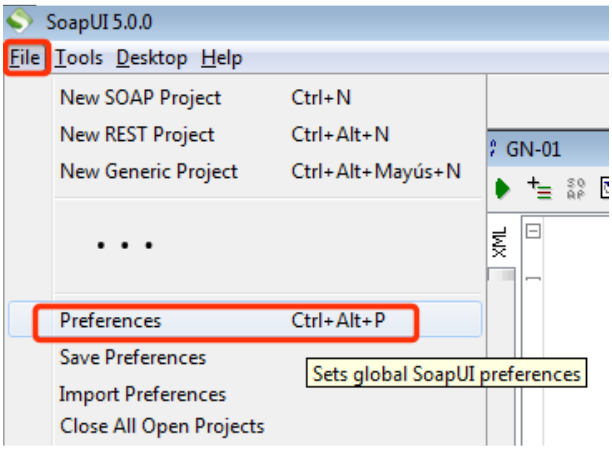
Installation:

- Download and install SOAP-UI: https://www.**soapui**.org/
- Download the test project from the bitbucket site: https://bitbucket.org/smree/eemws-core/downloads/IEC-62535-504-soapui-project.xml
- Open SoapUI and import the test file using: `File / Import Project` and choose the xml file that you have downloaded from bitbucket.

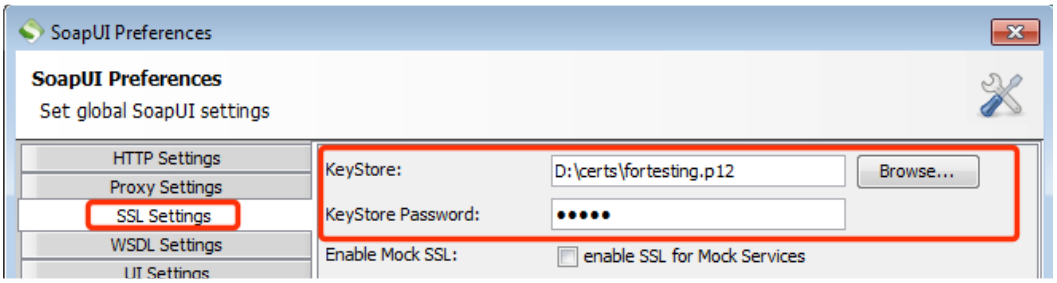


Server testing (use SoapUI as client)

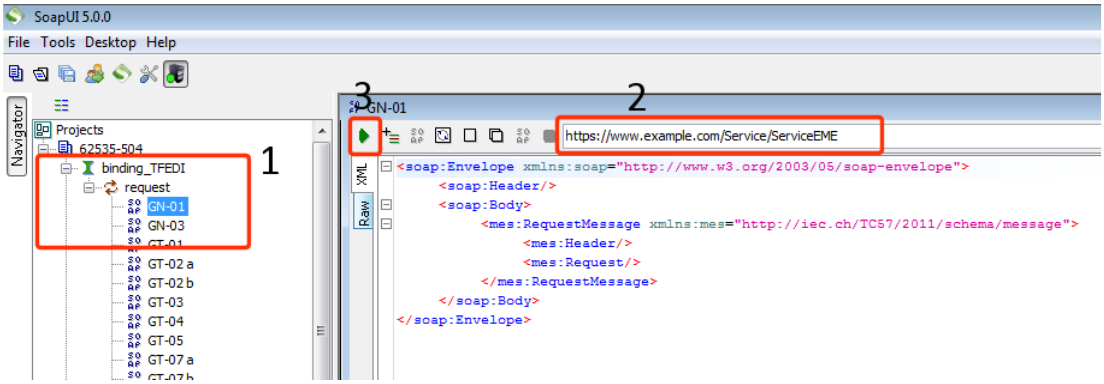
- Setup your client's certificate using: `File / Preferences:`



- In the Preferences dialog, choose the option "`SSL Settings`".



- Write the information concerning certificate file name and password (Tip: Depending on the test, you have to put here an expired certificate, no certificate (blank), an unauthorized certificate, etc.)
- Close `Preferences` dialog using "`OK`" button.
- Choose a test case from the request tree.
- Edit the test case –if needed- using the left panel (client request)
- Set the server URL (tip: you can search and replace in the xml document the text "https://www.example.com/Service/ServiceEME by the server's URL that you are testing)
- Click on the "play" button.
- The server response will appear in the right panel (server response)



Client testing (use SoapUI as server)

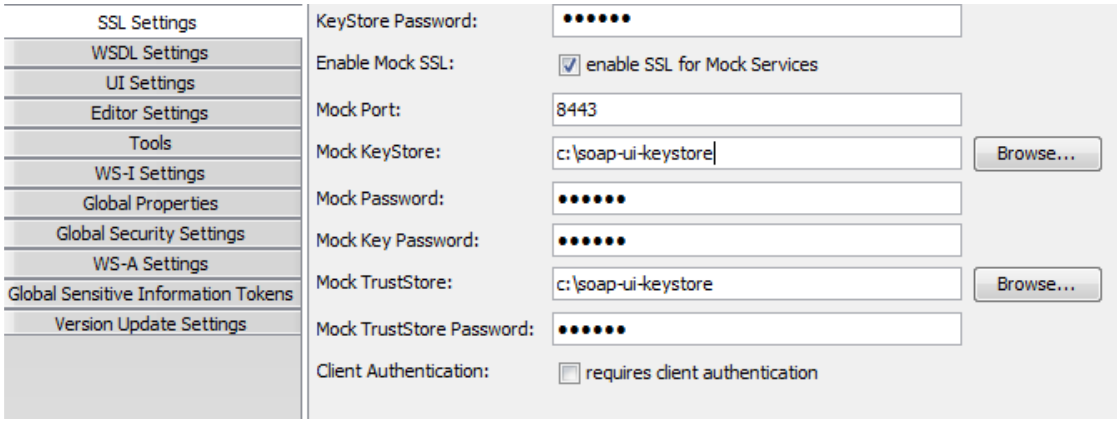
- Setup a server certificate. If you do not have one, create it as following. (Tip: your server certificate name must match the url you are going to use for testing)
    - Using keytool (provided with java developer kit aka JDK) type the following:

```
keytool -genkey -alias soapui -keyalg RSA -keystore c:\soap-ui-keystore
```

    - You will be prompted for first name, organizational unit, etc. Use "`localost`" when possible. At the end of the process, you will be asked for a password. This will be crated a certificate with the following distinguish name:

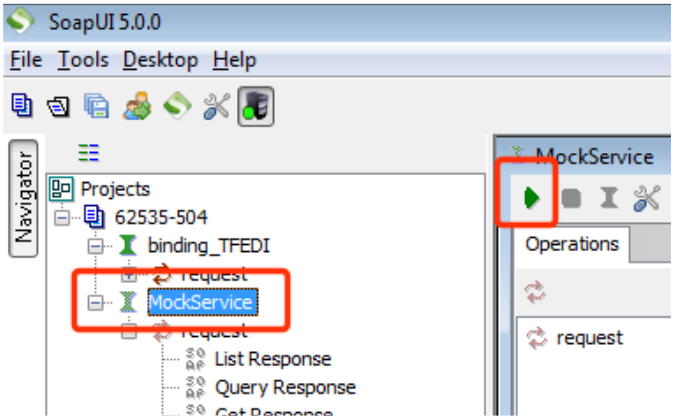
```
CN=localhost,OU=localhost,O=localhost,L=Unknown,ST=Unknown,C=es
```

- Go to the SSL settings and setup "`Mock Services`". Enable SSL for Mock Services, indicate the port and use the key store you have prepared.



Once you have setup SoapUI Mock security settings, you have to close SoapUI and start it again in order to apply the changes. Otherwise, the application will not use the security settings.

- Open the "MockService" item and click on the play button:



- In your client application setup the url: `https://localhost:8443/Service/ServiceEME`

- Use the request items under the `MockService` to check the received request as well as the provide responses. You can change the responses to test your client behavior.