NEMSIS TAC Whitepaper

NEMSIS V3 Web Services Guide

Date

February 27, 2020

Author:

Dhanya Sathyan - Software Design Engineer, NEMSIS TAC
Laurel Baeder – Software Design Engineer, NEMSIS TAC

NEMSIS V3 Web Services Guide

Background

Software applications seeking NEMSIS V3 compliance must support data exchange via Web Services (WS) as described in this guide. The standardized NEMSIS WS API provides the following advantages:

1. Fully automated electronic data exchange (including submission and retrieval) between agency, regional, state, and national systems.
2. Better transaction management: The data exchange can be easily monitored without risk of timeouts by using an asynchronous transaction model. In NEMSIS systems, XML and business validation reports can be available in real time from the same communication channel.
3. Possibility to move to real-time data exchange supporting public health surveillance.
4. Vendor independence, scalability, and wide acceptance/adoption.

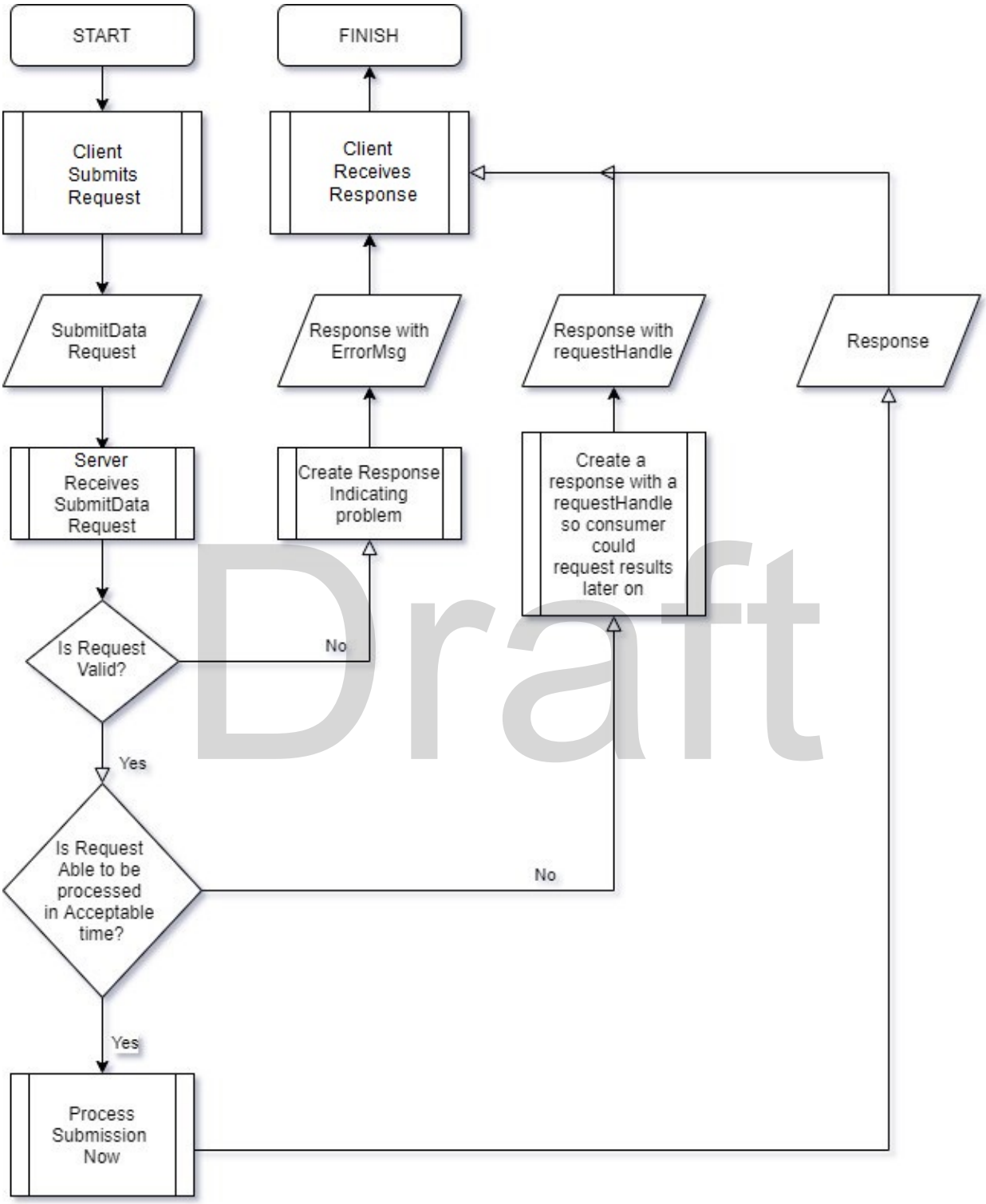
Purpose of NEMSIS V3 Web Services API

The NEMSIS V3 WS API is a unified interface that supports data exchange between distributed NEMSIS V3 systems. In a typical scenario, a client can interact with a Web Service at a given URL by using the HTTPS protocol. NEMSIS Web Services are based on SOAP.

Major Design Goals:

1. Flexibility: Allows applications to communicate over the Web in a platform-neutral, language-independent environment.
2. Efficiency: Instant response and quick turn-around.
3. Unity: Minimize confusion over information exchange by standardizing the information exchange interface.

Work Flow Diagram (Submission Request Only)



General Requirements

NEMSIS "Collect Data" systems must implement the NEMSIS Web services requirements as clients. "Collect Data" systems must be able to send NEMSIS EMSDataSet and DEMDataSet data to compliant "Receive and Process" systems and process the messages returned by "Receive and Process" systems.

NEMSIS "Receive and Process" systems must implement the NEMSIS Web services requirements as both clients and servers. As clients, "Receive and Process" systems must be able to send NEMSIS EMSDataSet, DEMDataSet, and StateDataSet data to the national EMS database and process the messages returned by the national EMS database. "Receive and Process" systems must also implement all operations, data types, and codes defined by the NEMSIS WSDL to receive NEMSIS EMSDataSet and DEMDataSet data from other systems and return messages regarding validation and processing results.

Detailed Requirements

Protocol

Simple Object Access Protocol (SOAP) is required. SOAP is a W3C standard protocol for sending and receiving requests and responses over the internet. SOAP messages can be sent back and forth between clients and servers in SOAP envelopes.

Security

Hyper Text Transport Protocol Secure (HTTPS) is the required communication channel. HTTPS is used for secure communication over public networks. In HTTPS, the transmission of information is encrypted using Transport Layer Security (TLS). Systems should support TLS version 1.3, must support version 1.2, and must not support versions 1.0 and 1.1.

Web Service messages might include sensitive information such as login credentials and protected health information (PHI). These types of data must be protected at the transport level.

WSDL

NEMSIS Web services are defined using Web Services Description Language (WSDL) A WSDL document describes how to access a web service and what operations are supported.

The NEMSIS standard reference WSDL is at https://nemsis.org/media/nemsis_v3/master/WSDL/NEMSIS_V3_core.wsdl. All NEMSIS web services must implement the operations and data types defined in the reference WSDL. Services may implement additional operations and data types or values.

Services should publish a WSDL. For example, the NEMSIS Validator WSDL is available at https://validator.nemsis.org/nemsisWs.wsdl.

The WSDL specifies the address to which clients may submit requests:

```
<soap:address location="https://validator.nemsis.org/"/>
```

In the example above, operations exposed by this Web Service will be available at the URL, "https://validator.nemsis.org/".

The WSDL specifies the operations:

```
<wsdl:operation name="RetrieveStatus">
<wsdl:input message="tns:RetrieveStatusRequest" name="RetrieveStatusRequest"> </wsdl:input>
<wsdl:output message="tns:RetrieveStatusResponse" name="RetrieveStatusResponse"> </wsdl:output>
</wsdl:operation>
<wsdl:operation name="SubmitData">
<wsdl:input message="tns:SubmitDataRequest" name="SubmitDataRequest"> </wsdl:input>
<wsdl:output message="tns:SubmitDataResponse" name="SubmitDataResponse"> </wsdl:output>
</wsdl:operation>
<wsdl:operation name="QueryLimit">
<wsdl:input message="tns:QueryLimitRequest" name="QueryLimitRequest"> </wsdl:input>
<wsdl:output message="tns:QueryLimitResponse" name="QueryLimitResponse"> </wsdl:output>
</wsdl:operation>
```

This Web Service provides “RetrieveStatus”, “SubmitData” and “QueryLimit” operations. All NEMSIS-compliant web service implementations must support these operations.

Status Codes

All responses from the Web Service must have a Status Code to communicate to a client what happened with a request. All Status Codes are defined in the WSDL, and listed at the end of this document.

1. Common errors (defined by the WSDL standard) should be included in the Response Data and use the appropriate Status Code as indicators.
2. The NEMSIS WSDL defines a set of standard response codes. All WS response messages must include a status code.
    a. All error codes are negative integers.
    b. All success codes are positive integers.
    c. Integer “0” means that the requested action is not completed yet. The client should query the server later to retrieve final processing status.
    d. Systems may define custom error codes, with integers smaller than -100. Similarly, custom success codes should use integers greater than 100. The NEMSIS TAC does not test the ability to support custom codes.
3. Systems must adopt the definitions from the NEMSIS WSDL for codes between -100 and 100. For example, it is non-compliant to use “-1” as a status code for “Failed import of a file, because of the [FATAL] level Schematron rule violation” since “-1” has been defined in the NEMSIS WSDL as “Invalid username and/or password”.

Important Common Objects

DataPayload

The submitPayload object is required for submitting data. It must contain a payloadOfXmlElement, which must contain the XML document being submitted. A simple example:

payloadOfXmlElement	<submitPayload> <payloadOfXmlElement> <EMSDataSet xmlns="http://www.nemsis.org" … > … </EMSDataSet> </payloadOfXmlElement> </submitPayload>

DataSchema and schemaVersion

Compliant implementations must support the EMSDataSet and DEMDataSet schemas. Services are not required to support the StateDataSet schema, but Receive and Process systems must be able to send StateDataSet data to the national EMS database. States/vendors may define custom schema codes. The NEMSIS TAC does not test the ability to support custom codes.

Combined with DataSchema, schemaVersion helps to identify the exact schema used for the payload. For example, DataSchema = "61" and schemaVersion = "3.4.0" is a valid combination. It points to v3.4.0 of NEMSIS EMSDataSet schema. However, there is no version 2.5.6 for NEMSIS EMSDataSet. So DataSchema = "61" and schemaVersion = "2.5.6" is not a valid combination.

The national EMS database supports the following schema and version combinations:

Dataset	requestDataSchema	schemaVersion
EMSDataSet	61	3.3.4, 3.4.0, 3.5.0
DEMDataSet	62	3.3.4, 3.4.0, 3.5.0
StateDataSet	65	3.5.0

SubmitDataReport

SubmitDataReport is for the report of data submission processing. It should include at least one XmlValidationErrorReport element for the XML Schema (XSD) validation report. The format implementation is as follows:

1. If the submission fails XSD validation, SubmitDataReport is required to contain one XmlValidationErrorReport. Since the submission is rejected at that step, SchematronReport must not be included.
2. If the submission passes XSD validation but fails Schematron, SubmitDataReport must contain
    a. XmlValidationErrorReport, with XmlValidationErrorReport's totalErrorCount set to zero.
    b. SchematronReport, which contains completeSchematronReport in Schematron Validation Report Language (SVRL). (If a receiving system accepts all records in a transaction, then it may omit schematronReport.)
3. SubmitDataReport may include optional CustomReport elements. The NEMSIS TAC does not test the ability to support CustomReport.
4. The SchematronReport allows two deviations from the SVRL standard. More details are available in Schematron Guide.
    a. Fired-rule may occur zero times.
    b. Diagnostic-reference may contain XML elements.

XmlElementInfo

This is used to identify the offending XML element reported in XSD validation. Line/column numbers, or XPATH location, could be used to identify the position of an XML element.
Usually, DOM parsers (validators) report XPATH information, and SAX/StAX parsers (validators) report line/column numbers. Some XML processors, such as SAXON EE/PE, can report both. Services must support line/column or XPath position reporting. Services may support both but are not required to do so. In cases where an error cannot be traced to a specific location, a service may use the elementLocationUnknown element.

XmlValidationError

There are two major types of XML validation errors (for a complete list of XML validation errors, check http://svn.apache.org/viewvc/xerces/java/trunk/src/org/apache/xerces/impl/msg/). One could be pinpointed to a particular element: for example, if the element is defined as an integer but the value "ABC" is submitted. For this kind of error, use XmlElementInfo to report the offending element. Another type of error is not focused on one element: for example, if a CSV file is submitted for XML validation. In this case, use XmlGeneralErrorList to include a list of error messages.

Main Operations

Communication in Web Services is defined by the request message and response message. There are three required operations: SubmitData, RetrieveStatus, and QueryLimit. The data structures for the request and response messages corresponding to these operations are defined in the NEMSIS WSDL.

Username, password, and organization are always required for any WS request. For the three defined operations, values for element requestType are predefined. Systems may support additional operations with a custom value for "requestType". The NEMSIS TAC does not test the ability to support custom operations. All operations also include an element of "additionalInfo" to allow for custom input.

NEMSIS WS response messages all include a status code (discussed above) and an echoing "requestType", set to the same value as in the request. Except for the function of QueryLimit, they also include a unique identifier "requestHandle", a system-assigned unique identifier at the server side for the transaction that takes place as a result of the request. In multiple query response situations, the "requestHandle" is used as a common point of reference.

SubmitData

To submit data, the client needs to specify the data payload, data schema, and schema version. As discussed above, the combination of data schema and version will decide the standard for the data submitted. After the payload is submitted, it is subject to XSD and Schematron validation. The submission is to be rejected if:

1. XSD validation fails
2. National Schematron [FATAL]s or [ERROR]s are generated
3. State Schematron [FATAL]s or [ERROR]s are generated
4. Other critical business rules are violated
5. The size of SOAP message exceeds the limit set by the server

The response for data submission could be synchronous or asynchronous: in the synchronous situation, an object of "SubmitDataReport" is included in the response, together with status code, handle, and echoing requestType. In the asynchronous situation, the server is not able to process the submitted data in time. Then "SubmitDataReport" is not included in the response. The client should use the assigned requestHandle in the response message to query the server later.

RetrieveStatus

The RetrieveStatus operation is used to retrieve results from a previous submission. This is most important for an asynchronous response from the submission. However, even with a synchronous response, the server should save the submission processing status result for a limited time period. In this case, if the client needs to retrieve the result later, it can use the returned requestHandle.
Possible responses to RetrieveStatus operation:

1. If the process is still pending, the server should return the same requestHandle and status code of 0.
2. If the process is completed and status is available, the server should return the same requestHandle, proper status code, and a SubmitDataReport.
3. The server should return an appropriate code if the status is not available because the status has expired or been deleted, or the requestHandle is not a valid identifier.

QueryLimit

Different web servers and WS implementations could apply a unique constraint on the size of the whole Web Service message. WS clients can use this interface to query WS server's configuration for this limit. The response to QueryLimit could be:

1. A positive integer to indicate the size limit on data payload, expressed in KB (1024 bytes).
2. A negative integer and an error status code.

Use Cases for NEMSIS Web Services

Case 1 – Submit Data Request, Synchronous scenario

Step 1: Agency "Elmo" wants to send an EMS record to state "Sesame Street". A Submit Request is sent with "Request Type" = "SubmitData". The SOAP message looks like this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws.nemsis.org/">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:SubmitDataRequest>
         <ws:username>emonster</ws:username>
         <ws:password>ABC123</ws:password>
         <ws:organization>ElmoAgency</ws:organization>
         <ws:requestType>SubmitData</ws:requestType>
         <ws:submitPayload>
            <ws:payloadOfXmlElement>
               <!—NEMSIS EMSDataSet -->
            </ws:payloadOfXmlElement>
         </ws:submitPayload>
         <ws:requestDataSchema>61</ws:requestDataSchema>
         <ws:schemaVersion>3.5.0</ws:schemaVersion>
         <ws:additionalInfo></ws:additionalInfo>
      </ws:SubmitDataRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Step 2: The State receives the message and successfully processes it within a reasonable amount of time. It replies with a "requestHandle", a system-assigned unique identifier for the transaction of handling Elmo's WS request. Then the response message looks like this:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <SubmitDataResponse xmlns="http://ws.nemsis.org/">
         <requestType>SubmitData</requestType>
         <requestHandle>23456789</requestHandle>
         <statusCode>1</statusCode>
         <reports>
            <xmlValidationErrorReport>
               <totalErrorCount>0</totalErrorCount>
            </xmlValidationErrorReport>
            <schematronReport>
               <completeSchematronReport>
                  <completeReport>
                     <payloadOfXmlElement>
                         <!-- attached Schematron report -->
                     </payloadOfXmlElement>
                  </completeReport>
               </completeSchematronReport>
            </schematronReport>
         </reports>
      </SubmitDataResponse>
   </S:Body>
</S:Envelope>
```

Case 2 – SubmitData Request, Asynchronous Scenario

Step 1: Same as Case 1, Agency "Elmo" sends XML to state "Sesame Street"

Step 2: The state receives the XML file. Because the system is too busy handling other agencies' requests, the response message looks like this:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <SubmitDataResponse xmlns="http://ws.nemsis.org/">
         <requestType>SubmitData</requestType>
         <requestHandle>12345678</requestHandle>
         <statusCode>0</statusCode>
      </SubmitDataResponse>
   </S:Body>
</S:Envelope>
```

Step 3: After 5 minutes, as a responsible agent, Elmo thinks it has given the state system enough time to process the data. Using "requestHandle", Elmo can query the state system to get the result of its previous WS request. The SOAP message for this kind of "RetrieveStatus" request looks like this:

```
 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ws="http://ws.nemsis.org/">
    <soapenv:Header/>
    <soapenv:Body>
       <ws:RetrieveStatusRequest>
          <ws:username>emonster</ws:username>
          <ws:password>ABC123</ws:password>
          <ws:organization>ElmoAgency</ws:organization>
          <ws:requestType>RetrieveStatus</ws:requestType>
          <ws:requestHandle>12345678</ws:requestHandle>
               <!--Optional:-->
          <ws:originalRequestType>SubmitData</ws:originalRequestType>
          <ws:additionalInfo></ws:additionalInfo>
       </ws:RetrieveStatusRequest>
    </soapenv:Body>

</soapenv:Envelope>
```


Step 4: At this stage, Elmo Agency could receive one of two different responses.

1. If the state has processed the file, they will receive a response similar to Case 1, Step 2.
2. If the state has not processed the file, they will receive the response in Case 2, Step 2 again. Elmo will need to send another retrieveStatus request later to determine the status of their submission.

Note: If for any reason Elmo forgets the status of his submission, even after he has received notification of a successful submission, Elmo should be able to send a "RetrieveStatus" request to the state again. The state should respond back with a status as seen in Case 1, Step 2. If the requestHandle is for an old submission, the state may tell Elmo the status has expired. For example, the national registry maintained by the NEMSIS TAC does not guarantee responses on submissions more than 6 months old.

Authentication/Security Implementation

Some vendors might prefer to utilize the security element in the SOAP message's header. Web Service Security Specification, published by OASIS (http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf), provides a set of mechanisms to enforce message integrity and confidentiality. However, due to the differences among EMS vendors/deployments, the NEMSIS TAC will not specify a particular SOAP security header structure. A vendor may choose to include the authentication information in the SOAP header. In such instances, both ends of the WS communication must agree to the security mechanism (and ignore the username/password parameters in the proposed object nemsisV3WsRequest). For example, they can choose to use (a)symmetric keys to encrypt username/password.

Passwords should never be sent as clear-text over a non-secured channel. This requirement is always met in NEMSIS-compliant web services, because NEMSIS requires the use of the HTTPS protocol.

Web Service Status Codes

The following is a summary of all status codes used in the NEMSIS V3 WSDL

Note: If any discrepancy is found between this document and the latest NEMSIS Version 3 WSDL release, the WSDL will take precedence.

PrivilegeErrorCodes: Error codes of authenticaition/authorization for an attempted web service operation	-1	Invalid username and/or password
	-2	Permission denied to the client for the operation
	-3	Permission denied to the client for that organization
ParameterErrorCodes: Generic error codes fan an attempted web service operation	-4	Invalid parameter value
	-5	Invalid parameter combination
ServerErrorCodes: Error codes for web service server	-20	Generic server error
	-21	Server error, because of database connection/operation issue
	-22	Server error, because of file system/network/IO issue
SubmitDataProcessCodes: Codes to describe return codes for an attempted data submission web service operation	-11	Failed import of a file, because the same file is already on the server
	-12	Failed import of a file, because of failing XML validation
	-13	Failed import of a file, because of [FATAL] level Schematron rule violation
	-14	Failed Import of a file, because of [ERROR] level Schematron rule violation
	-15	Failed Import of a file, because of critical ETL rule violation
	-16	Failed import of a file, because of critical Business Intelligence rule violation
	-30	Failed import of a file, because the size of SOAP message exceeds the limit
	1	Successful import of a file
	2	Successful import of a file, with [ERROR] level Schematron rule violation reported
	3	Successful import of a file, with [WARNING] level Schematron rule violation reported

	4	Successful import of a file, with ETL rule warning
	5	Successful import of a file, with Business Intelligence warning
	6	Partially successful import of a file, with [ERROR] level Schematron rule violation reported
	10	File has passed validation, processing is not yet complete
ResultPendingCode: Code to indicate the process is not finished on the server for an attempted web service operation	0	The expected data processing is not yet complete
RetrieveErrorCode: Code to indicate the error status for RetrieveStatus operation	-40	Status for the requested requestHandle is not available: it could be expired, or not in the correct format, or never exist, or for any other whatever reason.
	-41	Status for the requested requestHandle is not available since it expires already
	-42	Invalid value of requestHandle (for example, not formatted properly)
	-43	Never-used value of requestHandle
QueryLimitCodes: Code to indicate the status for QueryLimit operation	51	Successful operation of QueryLimit
	-50	Server is too busy. The client should query later
	-51	Failed operation of QueryLimit
CustomErrorCodes: State or Vendor specific error returning code for web service request. It should be smaller than -100	Smaller than -100	
CustomSuccessCodes: State or Vendor specific success returning code for web service request. It should be greater than 100	Greater than 100	

Using SoapUI

SoapUI is a popular open-source application for testing web services. See the following resources for using SoapUI as a web service testing tool:

- Download SoapUI: https://www.soapui.org/downloads/soapui.html
- Official Getting Started Guide for SoapUI: https://www.soapui.org/getting-started/introduction.html

Other applications, both free and paid, can also be used to test web services.

References

1. WS-SecurityPolicy 1.2. OASIS Standard, 1 July 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf
2. Web Services Security: SOAP Message Security 1.1. OASIS Standard Specification, 1 February 2006. http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
3. Web Services Security: UsernameToken Profile 1.1. OASIS Standard Specification, 1 February 2006. http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf
4. SOAP Message Size Performance Considerations. C. Rayns. T. Clarke, et al., 27 August 2007. http://www.redbooks.ibm.com/abstracts/redp4344.html
5. The Transport Layer Security (TLS) Protocol Version 1.3. E. Rescorla, Mozilla, August 2018. https://tools.ietf.org/html/rfc8446