ICC international computing centre

HUB Web Service API

IPPC ePhyto HUB

v1.14

Public - FAO/IPPC

Table of Contents

Document Profile

Author:	UNICC
Owner:	UNICC
Client:	FAO/IPPC
Document Number:	1.15

Revision History

Date of next revision: N/A

Version:	Who:	What:	When:
1.0	UNICC	Primary Document	12/12/2016
1.1	UNICC	Revision after PTC meeting in Geneva	22/03/2017
1.2	UNICC	Iteration 2 Review	31/07/2017
1.3	UNICC	Iteration 3 review	11/09/2017
1.4	UNICC	Revision after PTC meeting in Valencia	03/10/2017
1.5	UNICC	Java client sample code added	24/10/2017
1.6	UNICC	Receiving through PUSH Sample implementation added	16/11/2017
1.7	UNICC	Reviewed HUB Admin console urls	04/01/2018
1.8	UNICC	Updates of the March 2018 Release	22/03/2018
1.9	UNICC	Updates of the April 2018 Release	07-May-2018
1.10	UNICC	Review of external links	01-Jun-2018
1.11	UNICC	July 18 release	16-July-2018
1.12	UNICC	Oct 18 release	9-Nov-2018
1.13	UNICC	Jan-2019 release	28-Jan-2019
1.14	UNICC	Q1-2020 release (Channel implementation)	27-Apr-2020

Distribution

This document has been distributed as follow, from version 1.10 is published and distributed as part of the HUB release communication and hosted under the HUB Admin Console.

Document Roadmap

Following is the planned enhancements to this document

Feature

1. Introduction

1.1 Purpose

This document describes the IPPC HUB Web Service. It should be used as guideline to implement the required client software components needed to connect to the HUB.

1.2 Intended Audience and Reading Suggestions

The audience for this document is for developers and system architects who will evaluate and release the components for connecting to the HUB. It will also be used for developing the interface between the IPPC HUB and the IPPC Generic National System (GeNS), as well as serve as prototype documentation of the HUB implementation. This document should be read in parallel with the ePhyto HUB Software Requirements Specification and it is totally related to the hub web services operations and usage.

1.3 References

- ePhyto HUB Software Requirements Specification
- https://www.ippc.int/en/ephyto/
- https://www.ippc.int/en/ephyto/ephyto-technical-information/

2. HUB On-Boarding

- The first step in this process is to send a Registration request using https://www.ephytoexchange.org/onboard
- After due validation and confirmation from the IPPC official contact point of the indicated country, the user account is automatically created (using the indicated contact email) and credentials are sent.
- After obtaining access to the admin console the HUB Administrator will contact the focal point sending the latest version of this document and supporting the NPPO for the initial setup of the UAT site, where the implementation can be tested and validate before the final release to production.

3. Technical Support

For general queries on the HUB please go to https://www.ephytoexchange.org/support

We also encourage using the collaboration tool to get quick answers and share experiences. *(Registered users only)*

If you encounter issues while testing the implementation of the components needed to connect to the HUB you can raise a technical support request from within the Administration Console (https://www.ephytoexchange.org/AdminConsole) following the link available in the menu after the successful login.

The system will send a mail to the technical team that will respond to the query.

We suggest looking first at the collaboration area of the admin console as possible source of information.

4. HUB Web Service Systems

4.1 Testing Environment (UAT)

The testing environment (UAT) is a live system with the latest release of the system that is constantly available to test the implementation of the client application connection to the HUB.

Self-Signed certificates and ad-hoc credentials can be provided in order to facilitate the activities.

4.1.1 URL for Testing

HUB UAT/Test environment can be accessed from the following URLs:

https://uat-hub.ephytoexchange.org/hub/DeliveryService?wsdl (web service WSDL)

(web service endpoint will accept only certificate authentication)

https://uat-hub.ephytoexchange.org/AdminConsole (Admin interface)

Log in credentials to the console are provided during the process of the on-boarding.

4.1.2 Certificates for Web Service client authentication

During the testing phase, web services client authentication will use self-signed certificates. Connecting entities can issue their certificates with the “keytool” command found in the Java Development Kit (JDK) or they can request that UNICC provide a sample certificate that they can use.

With the March 2018 release the administrator user can access the HUB Admin Console and update the public certificates that will be used to authenticate the client application when using the web services.

4.1.2.1 Requesting a test certificate from UNICC

The following are the information needed to create the *X.500 Distinguished Name* for the certificate:

- Common Name
- Organization Unit
- Organization Name
- Locality Name (city)
- State Name
- Two-letter country code

UNICC will send a key store file in the PKCS12 format with the certificate, and upload the public key in the HUB Admin Console.

4.1.2.2 Generating a self-signed test certificate

Certificates can be generated with the “keytool” command provided by the JDK.

Example of certificate generation for a NPPO entity located in London, UK with a validity of 10 years:

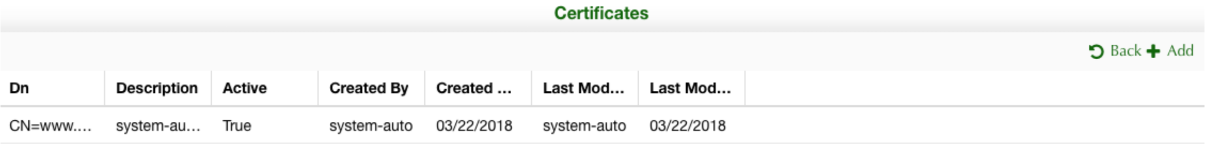
```
C:\certificates>keytool -genkey -alias nppo1 -keyalg RSA -keysize 1024 -keystore nppo.keystore -validity 3650 -keypass nppo1pass -storepass nppoStore1pass
What is your first and last name?
  [Unknown]:  www.nppo.mycountry
What is the name of your organizational unit?
  [Unknown]:  NPPO
What is the name of your organization?
  [Unknown]:  NPPO-MyCountry
What is the name of your City or Locality?
  [Unknown]:  Capital
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:  MC
Is CN=www.nppo.mycountry, OU=NPPO, O=NPPO-MyCountry, L=Capital, ST=Unknown, C=MC correct?
  [no]:  yes
```

Example of public key export from the key store:

```
C:\certificates>keytool -export -keystore nppo.keystore -alias nppo1 -file nppo1.cer -keypass nppo1pass -storepass nppoStore1pass
```

Once generated, the public key can be uploaded and configured in the HUB Admin Console, with the following steps:

1) Enter the Admin Console and navigate to the Configuration View ⚙ Configuration
2) Open the certificates screen with the link at the top right ⚙ Certificates
3) Client Certificates are listed (see screenshot below), by using the add button + Add you can upload the new certificate (The HUB Console will only need the Public Key in a form of a .cer file). Existing certificates can be only disabled by entering the record, changing the active flag and saving the information.



Certificates

Dn	Description	Active	Created By	Created ...	Last Mod...	Last Mod...
CN=www....	system-au...	True	system-auto	03/22/2018	system-auto	03/22/2018

4.2 Production Environment

HUB Production environment can be accessed from the following URLs:

https://hub.ephytoexchange.org/hub/DeliveryService?wsdl (web service WSDL)

(web service endpoint will accept only certificate authentication)

https://www.ephytoexchange.org/AdminConsole (Admin interface)

4.3 Authentication

Authentication to the web service supports TLS 1.1 and TLS 1.2 client certificates that are associated to each country accessing the HUB. X509 certificates are the client credentials. Each connected application will have a defined certificate, issued by a recognized Certificate Authority that will authenticate the client application to the HUB on HTTPS protocol.

Details of the Security implementation are outside the scope of this document but contained in the referenced HUB requirements document specification.

The HUB will only accept 'envelopes' where the 'From' field (described below) matches the TLS Certificate of the connecting system.

5. HUB XML Schemas

5.1 Schema

The HUB web service schema is composed by a large number of entities, some of them are part of the ePhyto definition, they will be described more in details in each web service operation. See below the list of the main elements:

1) Envelope Header
2) Envelope Content
3) ePhyto Envelope

The WSDL defined in this document (Section 6) has several operations; mainly supported by the following entities:

a. Envelope Header
b. Envelope = header + content
c. ePhytoEnvelope = header + SPSCertificate
d. Array of Envelope Header
e. Array of Envelope
f. HUBTrackingInfo
g. NPPO
h. ValidationResult

5.1.1 Envelope Header

The envelope header element is used to exchange information on the ePhyto certificates without viewing/processing the content of the actual certificate.

The HUB will be instrumented to verify the correct use of such codes and raise communication errors when such attributes are not complying with the standards. This will be a feature of the HUB software.

During interaction with the HUB, it is not mandatory to set all the elements within the header. However, some identified elements are required at the minimum.

The Envelope header has the following elements:

- **From:** ISO 3166-1 alpha 2 letter Country Code of the exporting country
- **To:** ISO 3166-1 alpha 2 letter Country Code of the importing country
- **CertificateType:** This is the UNECE code for certificate types. For the IPPC implementation, the HUB will check that the type code corresponds to the following two numbers only.
    - 851 for Phyto
    - 657 for Re-Export Phyto
- **CertificateStatus:** This is the UNECE code for the status of the certificate. For the IPPC implementation, the HUB will check that the status code corresponds to one of the following numbers:
    - 70:Issued
    - 39:Approved
    - 40:Withdrawn
    - 41:Rejected
- **NPPOCertificateNumber:** For its own reference, the **exporting** NPPO can insert the certificate number of the ePhyto contained within the envelope, in this field. It will allow the NPPO national system to match a certificate against the HubTrackingNumber in its own national system. Furthermore, the HUB user-interface will also display this number along with the delivery status. This element is multi-lingual; allowing the exporting NPPO to use any language of their choice. This is limited to 1000 characters.
- **HUBTrackingNumber:** This is unique identifier that will be assigned by the HUB for each envelope when it receives the envelope for the first time. The NPPO system can subsequently query the HUB against this identifier; to get delivery information on any particular certificate identified by the HUBTrackingNumber. This element size can grow up to 50 characters long.
- **HUBTrackingInfo:** This element has one of the following four status codes; indicating the delivery status of the envelope within the HUB:
    - **PendingDelivery**: implies that the envelope is still held within the HUB and has not been delivered. Also, the queue expiry period is not over; thus, the HUB still has the envelope.
    - **Delivered**: The envelope was successfully delivered by the HUB and has been deleted after delivery
    - **FailedDelivery**: The HUB has not been able to deliver the envelope and the Queue expiry period set by the exporting system was reached. Thus, the envelope was deleted from the HUB queue.
    - **EnvelopeNotExists**: For the given Tracking Number, the HUB does not have any information.
    - **DeliveredWithWarnings:** introduced with March 2018 release it is used to mark envelopes that are acknowledged from the importing country with some non-compliancy warnings that can be reported as error text and read from the sending system to fine tune the generation of a standardized ePhyto XML

- **DeliveredNotReadable:** introduced with April 2020 release it is used to mark envelopes that are received but not readable, due to an XML that is not well-formatted or that does break the reading procedure at the receiving system.
- **HUBErrorMessage:** This element will have messages for different errors that may occur during interaction with the HUB. Most of the error messages are related to Queue retention time expiration.
    - The importing country can set the warning messages in the AdvancedAcknowledge (see operations below) to indicate elements to be improved in the ePhyto XML they have received.
    - The importing country can set the error message in the AcknowledgeFailedEnvelopeReceipt (see operations below) to add the error message that prevented the correct opening of the envelope content

- **Forwardings:** From April 2020 release the sender country can specify the list of channels that the envelope must be forwarded. It is an optional Array of EnvelopeForwarding elements that is used during the Delivery operations and returning the list of forwarded channels with the relative HUB tracking info (see below the Channel forwarding option section below)

5.1.2 Envelope Content

The envelope type *inherits* the envelope header and extends it with the “Content” element that can be any type of string/xml.

The electronic phytosanitary certificate will be created by exporting client application, serialized into XML and sent to the HUB using the Content attribute of the envelope.

The HUB will not perform the validation of the certificate content and its adherence to the ISPM 12 schema. The importing client application will be responsible for opening the certificate content and ensuring it adheres to the applicable standard. At the receipt of the Envelope the importing Client Application has to acknowledge the successful receipt of the message, regardless of the certificate validation that will be performed with a separate business process.

The following reference document contains all the details on the XML mapping requirements and resources.

https://www.ephytoexchange.org/doc/mapping/Mapping_ISPM_12_to_ePhyto_standard_Export_certificate_V.2.pdf

5.1.3 Array of EnvelopeHeader

This element is used to exchange a number of envelope headers grouped together. The operations ‘GetUnderDeliveryEnvelope’ and ‘GetImportEnvelopeHeaders’ use this as described below.

5.1.4 Array of Envelope

This element contains a list of envelopes. Each envelope contains – one header and one ePhyto certificate. This entity is used in the operation ‘PULLImportEnvelope’ described in detail below.

6. Operations

6.1 Connect to the hub

Connecting to the HUB is not an operation exposed by the web service, but the internal call needed before any invoke of the remote web service operations.

In this section we show the generic code needed to open a client connection with the HUB using C# and the .Net Framework 4.6.1 and also Java 1.8 and the Apache Axis 1 framework for generating the client code from the WSDL definition.

The code will create the new client, add the certificate and the URL (depending on the environment) to be used in all the subsequent calls to the web service.

C#

```
    private static DeliveryService getClientConnection()
     {
        // the following code is use to prevent security protocol 
exceptions
        // raised by using self-signed certificates (test environment)
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls11;
        System.Net.ServicePointManager.ServerCertificateValidationCallback 
= delegate (
        Object obj, X509Certificate certificate, X509Chain chain,
        SslPolicyErrors errors)
        {
            return (true);
        };

        //This is the actual implementation of the generated proxy
        //from the given or downloaded WSDL
        DeliveryService client = new DeliveryService();

        //setting the test environment URL
        client.Url = "https://uat.ippchub.unicc.org/hub/DeliveryService";

        //adding the certificate
        X509Certificate2 cert = new 
X509Certificate2("/Users/luca/repos/IPPCHubDev/certificates/nppo-it.p12", 
"nppoITp12");
        client.ClientCertificates.Add(cert);

        //returning the client object
        return client;
    }
```

Java

```
  private static final String KEYSTORE_TRUSTED = 
"G:\\certificates\\trustedStore";
  private static final String KEYSTORE_TRUSTED_PASSWORD = "changeit";
  private static final String KEYSTORE_SERVER =
"G:\\certificates\\privateStore";
  private static final String KEYSTORE_SERVER_PASSWORD = "changeit";

  private static IDeliveryServiceProxy getClientConnection() {
    // Configure the stores with certificates
    System.setProperty("sun.security.ssl.allowUnsafeRenegotiation",
"true");  // true for self-signed certificates, false in production

    // Trusted certificates, IPPC HUB certificate should be here
    System.setProperty("javax.net.ssl.trustStore", KEYSTORE_TRUSTED);
    System.setProperty("javax.net.ssl.trustStorePassword",
KEYSTORE_TRUSTED_PASSWORD);

    // Private Key store, with NPPO certificate
    System.setProperty("javax.net.ssl.keyStore", KEYSTORE_SERVER);
    System.setProperty("javax.net.ssl.keyStorePassword",
KEYSTORE_SERVER_PASSWORD);

    // Uncomment next line to have handshake debug information
    // System.setProperty("javax.net.debug", "ssl");

    // Getting the proxy to the appropriate URL
    IDeliveryServiceProxy proxy = new IDeliveryServiceProxy("https://uat-
hub.ephytoexchange.org/hub/DeliveryService");

    return proxy;
  }
```

6.2 DeliverEnvelope

The exporting system will use this operation to send the envelope to the HUB. The Header must be filled with the following required minimum attributes:

- From,
- To,
- CertificateType,
- CertificateStatus
- NPPO Certiticate Number (is not mandatory but we suggest to use the field to be able to easily reference each transmission with the original certificate in the exporter system)
- and the 'Content' attribute is populated with the actual certificate; to complete the envelope with the XML serialized version of the generated ePhyto.

The HUB responds back with the EnvelopeHeader – which contains all the attributes populated by the exporting client application as well as the HUBTrackingNumber and the HUBTrackingInfo attributes are added by the HUB application.

In the case of 'Transit', when the certificate has to be distributed to transit countries too, the client application should send the envelope to all involved countries as separate message and each of the transmission will be tracked separately.

Client sample implementation in C# generated as .Net 2.0 standard web service client:

https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/transport-security-with-certificate-authentication

C#

```
           // initialize the client
            DeliveryService client = getClientConnection();

            // simulating an Issue certificate from Italy to United States
            Envelope env = new Envelope()
            {
                From = "IT",
                To = "US",
                CertificateType = 851,
                CertificateStatus = 70,
                NPPOCertificateNumber = "Internal NPPO Certificate Number"

            };

            //load the actual electronic certificate XML
            var ePhyto = new System.Xml.XmlDocument();
            ePhyto.LoadXml("<?xml version=\"1.0\" encoding=\"UTF-
8\"?><ephyto><contents/></ephyto>");

            //set the XML to the content element of the message
            env.Content = ePhyto.InnerXml;

            try
            {
                // send the message to the hub and get back the header
                EnvelopeHeader header = client.DeliverEnvelope(env);

                //handle internal issues
                if (header.HUBTrackingInfo == "FailedDelivery")
                {
                    //manage the exception and provide errors to the client
                    //in this case the error is due to one of the following
                    //Header Validation error (certificate, destination
country not boarded...)
                    //Internal error of the system

                    //get the error message
                    string error = header.hubDeliveryErrorMessage;
                    System.Console.WriteLine("Message failed delivery,
"+error);
                }
                else
                {
                    //get the hub tracking number...
                    string hubTrackingNumber = header.hubDeliveryNumber;
                    System.Console.Write("header delivered with tracking
number : " + hubTrackingNumber);

                    //persist the header details to record that the message
is under delivery
                }
            }catch(Exception ex){
                //manage the exception and provide errors to the client
                //in this case the error is due to one of the following
                //Header Validation error (certificate, destination country
not boarded...)
                //network
                //unavailability of the remote system
                Console.WriteLine("Failed to deiver the message to the HUB"
+ ex.Message);
            }
```

Java

```
  private static EnvelopeHeader DeliverEnvelope() throws HubClientException
{
    IDeliveryServiceProxy proxy = getClientConnection();

    DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();

    // Envelope creation, from Italy to United States
    Envelope envelope = new Envelope();
    envelope.setFrom("IT");
    envelope.setTo("US");
    envelope.setCertificateType(851);
    envelope.setCertificateStatus(70);
    envelope.setNPPOCertificateNumber("EPHYTO-IT-2017-0010277");

    try {
      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
      Document doc = dBuilder.parse("<?xml version=\"1.0\" encoding=\"UTF-
8\"?><ephyto><contents/></ephyto>");
      DOMSource domSource = new DOMSource(doc);
      StringWriter writer = new StringWriter();
      StreamResult result = new StreamResult(writer);
      TransformerFactory tf = TransformerFactory.newInstance();
      Transformer transformer = tf.newTransformer();
      transformer.transform(domSource, result);
      envelope.setContent(writer.toString());
    } catch (SAXException | IOException | ParserConfigurationException |
TransformerException e1) {
      //manage the exception and provide errors to the client
      //in this case the error is due to one of the following
      //The XML string could not be parsed
      System.out.println("Failed to load certificateinto XML document.");
      throw new HubClientException(e1); // Without certificate we cannot
continue
    }

    try {
      // send the message to the hub and get back the header
      EnvelopeHeader header = proxy.deliverEnvelope(envelope);

      // Handle internal issues
      if (header.getHUBTrackingInfo().equals("FailedDelivery")) {
        //manage the exception and provide errors to the client
        //in this case the error is due to one of the following
        //Header validation error
        String error = header.getHubDeliveryErrorMessage();
        System.out.println(String.format("Message failed delivery. %s",
error));
      } else {
        //get the hub tracking number...
        String hubTrackingNumber = header.getHubDeliveryNumber();
        System.out.println(String.format("Header delivered with tracking
number: %s", hubTrackingNumber));
      }
      return header;
    } catch (RemoteException e) {
      //manage the exception and provide errors to the client
      //in this case the error is due to one of the following
      // network
      // unavailability of the remote system
      System.out.println(String.format("Failed to deliver the message to
the HUB. ", e.getMessage()));
      throw new HubClientException(e);
    }
  }
```

If any error occurs, the HUBTrackingInfo is set to "FailedDelivery" and the details will be found in the hubDeliveryErrorMessage element of the Envelope Header returned. Possible errors detected include:

- The connected system cannot send with the specified "From" field
- There is no system connected to receive for the country specified in the "To" field
- Invalid certificate type
- Invalid certificate status

Connectivity issues such as network outages or unavailability of the system will be reported as standard HTTP protocol errors, as they are not generated by the remote application.

6.3 PULLImportEnvelope, AcknowledgeEnvelopeReceipt, AdvancedAcknowledgeEnvelopeReceipt, AcknowledgeFailedEnvelopeReceipt

The importing system configured for PULL operation will use this operation to retrieve all the envelopes that are destined for them. The authenticated client is representing the importing country and it will receive all of the envelopes (*array of envelope*) that are in the HUB's queue with the importing country in the To field. For each of these envelopes, the importing country should message back on the operation AckknowledgeEnvelopeReceipt the successful receipt of each envelope; with the HUBTrackingNumber.

Acknowledged messages will be removed from the queue and the next pull operation will fetch the remaining messages until the result is empty.

The system configuration will allow for reducing the batch of the messages of each pull in order to fine tune the communication with office using a poor connection.

The system supports an additional text message with the acknowledge operation that will set the tracking info to DeliveredWithWarnings and can provide with the error message the details of the issues found during the receiving and opening of the XML.

Please note that the warning message is limited to 200 characters, the return message may indicate such warning, the data will be truncated in case exceeds such limit.

See sample below, such messages can be extracted from a schema validation action performed by the receiver and reported back to the sender to leverage the XML standardization and codes harmonization.

Similar to the above if the envelope content is not readable the receiving system can use the AcknowledgeFailedEnvelopeReceipt to communicate that there was an error in opening the envelope and use the message of 200 characters to add the error text or reason of the failure.

Client sample implementation:

C#

```
    // initialize the client
    DeliveryService client = getClientConnection();

    //get all the envelopes pending delivery
    Envelope[] envelopesToImport = client.PULLImportEnvelope();

    foreach(Envelope env in envelopesToImport)
    {
        System.Console.WriteLine("Processing hub delivery number : "
+env.hubDeliveryNumber);

        try
        {
            //get the content containing the certificate XML
            String xmlContent = env.Content;

            //verifications in xml
            var ePhyto = new System.Xml.XmlDocument();
            ePhyto.LoadXml(xmlContent);


            //save the ePhyto to the client application
            //acknowledge the receipt back to the server (this could be
done as separate action based on user validation ??)
            client.AcknowledgeEnvelopeReceipt(env.hubDeliveryNumber);


            //perform schema/xml checks
           client.AdvancedAcknowledgeEnvelopeReceipt(env.hubDeliveryNumber,
“please indicate the date elements without milliseconds”);
        }
        catch(Exception ex)
        {
            //handle the content parsing error
            System.Console.WriteLine(String.Format("error when parsing
content of {0} {1}", env.hubDeliveryNumber,ex.Message));
        }
    }
```

Java

```
  private static void pullAcknowledge() throws HubClientException {
    IDeliveryServiceProxy proxy = getClientConnection();

    try {
      // get all the envelopes pending delivery
      Envelope[] envelopesToImport = proxy.PULLImportEnvelope();

      for (Envelope envelope : envelopesToImport) {
        System.out.println(String.format("Processing hub delivery number:
%s", envelope.getHubDeliveryNumber()));

        // get the content containing the certificate XML
        String xmlContent = envelope.getContent();

        // verifications in XML
        DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder;
        try {
          dBuilder = dbFactory.newDocumentBuilder();
          InputStream content = new
ByteArrayInputStream(envelope.getContent().getBytes(StandardCharsets.UTF_8.
name()));
          Document doc = dBuilder.parse(content);
        } catch (ParserConfigurationException | SAXException | IOException
e) {
          // The content of the envelope is not a proper XML file
          System.out.println(String.format("Error parsing content of %1$s
%2$s", envelope.getHubDeliveryNumber(), e.getMessage()));

          // This envelope won't be acknowledged

proxy.advancedAcknowledgeEnvelopeReceipt(envelope.getHubDeliveryNumber(),
"error while parsing the XML");
          continue;
        }

        //acknowledge the receipt back to the server (this could be done as
a separate action based on user validation)
        proxy.acknowledgeEnvelopeReceipt(envelope.getHubDeliveryNumber());
      }
    } catch (RemoteException e) {
      //manage the exception and provide errors to the client
      //in this case the error is due to one of the following
      // network
      // unavailability of the remote system
      System.out.println(String.format("Failed to deliver the message to
the HUB. ", e.getMessage()));
      throw new HubClientException(e);
    }
  }
```

If an error occurs in the processing of PullImportEnvelope, AckowledgeEnvelopeReceipt, AcknowledgeFailedEnvelopeReceipt or AdvancedAckowledgeEnvelopeReceipt, a standard SOAP Fault element will be sent describing the error. Errors detected in these services include:

- The system making the request is not in the system
- The acknowledge of receipt requester isn't from the entity in the "To" field of the acknowledged header

- Envelope not found, as above related to acknowledge request. When the sent number is not found in the HUB.

6.4 GetUnderDeliveryEnvelope

The operation allows the exporting system to get a list of all the envelope headers that are in the delivery process (i.e. with HUBDeliveryStatus as PendingDelivery). The authenticated client represents the exporting system. The HUB will return the list of all the envelopes that are pending delivery (array of EnvelopeHeader).

The client application can use the HUBTrackingNumber from the returned envelope headers and updates the system.

Client sample implementation.

C#

```
        DeliveryService client = getClientConnection();
        try
        {
              //get the envelopes under delivery (received by the HUB and
queued to be delivered to the destination)
              EnvelopeHeader[] headers = client.GetUnderDeliveryEnvelope();

              //cicles the records to update the client system
              foreach (var head in headers)
              {
                  //updates the client records
System.Console.WriteLine("Env:"+head.hubDeliveryNumber+",Tracking
Info:"+head.HUBTrackingInfo);
              }
        }
        catch (Exception ex)
        {
              System.Console.WriteLine(ex.Message);
        }
```

Java

```
  private static void getUnderDeliveryEnvelope() throws HubClientException
{
    IDeliveryServiceProxy proxy = getClientConnection();

    try {

      // get the envelopes under delivery
      EnvelopeHeader[] headers = proxy.getUnderDeliveryEnvelope();

      //clicles the records to update the client system
      for(EnvelopeHeader header : headers) {
        // updates client records
        System.out.println(String.format("Envelope: %1$s - Tracking info:
%2$s", header.getHubDeliveryNumber(), header.getHUBTrackingInfo()));
      }
    } catch (RemoteException e) {
      //manage the exception and provide errors to the client
      //in this case the error is due to one of the following
      // network
      // unavailability of the remote system
      System.out.println(String.format("Failed to deliver the message to 
the HUB. ", e.getMessage()));
      throw new HubClientException(e);
    }
  }
```

If an error occurs in the processing of GetUnderDeliveryEnvelope a standard SOAP Fault element will be sent describing the error. Errors detected in this service include:

- The system making the request is not in the HUB

6.5 GetImportEnvelopeHeaders & PULLSingleImportEnvelope

Similar, to the previous this operation allows the importing system to get a list of all the envelope headers that are in the delivery process. The authenticated client represents the importing country. The HUB will return the list of all the envelopes that are pending delivery (array of EnvelopeHeader).

The client application can use the HUBTrackingNumber from the returned envelope headers and pull each of them one by one. This will allow the importing country to work on the entire subset of messages to be delivered, rather than having to pull them in batches.

Additionally, by using the optional countryCode parameter the importing country can specify to retrieve only the list of headers of an exporting country and apply the required logic to the import procedures.

Client sample implementation.

C#

```
      DeliveryService client = getClientConnection();
      try
      {
            //get the envelopes under delivery (received by the HUB and 
queued to be delivered to the destination) use null or a country code to 
retrive all or only for a specific sending country
            EnvelopeHeader[] headers = client.GetImportEnvelopeHeaders();

            //cicles the records to update the client system
            foreach (var head in headers)
            {
                Envelope env = 
client.PULLSingleImportEnvelope(head.hubDeliveryNumber);
                //get the content containing the certificate XML
                String xmlContent = env.Content;

                //verifications in xml
                var ePhyto = new System.Xml.XmlDocument();
                ePhyto.LoadXml(xmlContent);
                //save the ePhyto to the client application
                //acknowledge the receipt back to the server (this could be 
done as separate action based on user validation ??)
                client.AcknowledgeEnvelopeReceipt(env.hubDeliveryNumber);


               //perform schema/xml checks

client.AdvancedAcknowledgeEnvelopeReceipt(env.hubDeliveryNumber, "please 
indicate the date elements without milliseconds");



            }
      }
      catch (Exception ex)
      {
            System.Console.WriteLine(ex.Message);
      }
```

Java

```
  private static void getImportEnvelopeHeaders() throws HubClientException 
{
    IDeliveryServiceProxy proxy = getClientConnection();

    try {

      // get the envelopes under delivery
      EnvelopeHeader[] headers = proxy.getImportEnvelopeHeaders("IT");

      //clicles the records to update the client system
      for(EnvelopeHeader header : headers) {

         // get the envelope
         Envelope env = proxy.PULLSingleImportEnvelope(header.
getHubDeliveryNumber());

        // get the content containing the certificate XML
        String xmlContent = env.getContent();

        // verifications in XML
        DocumentBuilderFactory dbFactory = 
DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder;
        try {
          dBuilder = dbFactory.newDocumentBuilder();
          InputStream content = new 
ByteArrayInputStream(env.getContent().getBytes(StandardCharsets.UTF_8.name(
)));
          Document doc = dBuilder.parse(content);
        } catch (ParserConfigurationException | SAXException | IOException 
e) {
          // The content of the envelope is not a proper XML file
          System.out.println(String.format("Error parsing content of %1$s 
%2$s", env.getHubDeliveryNumber(), e.getMessage()));

          // This envelope won't be acknowledged
proxy.advancedAcknowledgeEnvelopeReceipt(env.getHubDeliveryNumber(), "error
while parsing the XML");
     continue;
    }

    //acknowledge the receipt back to the server (this could be done as
a separate action based on user validation)
    proxy.acknowledgeEnvelopeReceipt(env.getHubDeliveryNumber());
   }
  } catch (RemoteException e) {
   //manage the exception and provide errors to the client
   //in this case the error is due to one of the following
   // network
   // unavailability of the remote system
   System.out.println(String.format("Failed to pull envelopes from the
HUB. ", e.getMessage()));
   throw new HubClientException(e);
  }
 }
```

If an error occurs in the processing of GetImportEnvelopeHeader, AcknowledgeEnvelopeReceipt and AdvanvcedAcknowledgeEnvelopeReceipt a standard SOAP Fault element will be sent describing the error. Errors detected in this service include:

- The system making the request is not in the HUB

6.6 GetEnvelopeTrackingInfo

This operation provides the HUBTrackingInfo for a given HUBTrackingNumber;none envelope at a time. The idea is that if the client application has sent the envelope and the envelope header is not listed in the pending delivery, then the system should query the hub to understand if it was delivered successfully and/or at which stage it is. In the code example below you can find a possible implementation of a recurring process that takes actions in the National System based on each of the tracking info.

C#

```
   DeliveryService client = getClientConnection();
      try
      {
         EnvelopeHeader head= client.GetEnvelopeTrackingInfo(num);

        System.Console.WriteLine(string.Format("The envelope {0}
tracking info is {1}",head.hubDeliveryNumber,head.HUBTrackingInfo ));

        switch(head.HUBTrackingInfo){
          case "Delivered":
            //perform client updates to mark the envelope
delivered
            break;
          case "DeliveredWithWarnings":
            //perform client updates to mark the envelope
delivered, capture the text and send the information to technical people
                        break;
                    case "FailedDelivery":
                        string error = head.hubDeliveryErrorMessage;
                        //update the client state with the informational
error message
                        break;
                    case "EnvelopeNotExists":
                        //the message was received by the hub but not yet
added to the queue or the number is not correct
                        //resending of the original can be applied
                        break;
                    case "PendingDelivery":
                        //still in the queue on the hub, waiting to be
pulled or pushed
                        break;
                }
            }
            catch (Exception ex)
            {
                  System.Console.WriteLine(ex.Message);
            }
```

Java

```
  private static void getEnvelopeTrackingInfo(String hubTrackingNumber)
throws HubClientException {
    IDeliveryServiceProxy proxy = getClientConnection();

    try {
      EnvelopeHeader header =
proxy.getEnvelopeTrackingInfo(hubTrackingNumber);
      System.out.println(String.format("The envelope %1$s tracking info is
%2$s", header.getHubDeliveryNumber(), header.getHUBTrackingInfo()));

      switch (header.getHUBTrackingInfo()) {
        case "Delivered":
          // perform client updates to mark the envelope as delivered
          break;
        case "DeliveredWithWarnings":
          // perform client updates to mark the envelope as delivered,
capture the error message and send it to the technical people
          break;
        case "FailedDelivery":
          String errorMessage = header.getHubDeliveryErrorMessage();
          // update the client state with the informational error message
          break;
        case "EnvelopeNotExists":
          //the message was received by the hub but not yet added to the
queue or the number is not correct
          //resending of the original can be applied
          break;
        case "PendingDelivery":
          //still in the queue on the hub, waiting to be pulled or pushed
          break;
      }
    } catch (RemoteException e) {
      //manage the exception and provide errors to the client
      //in this case the error is due to one of the following
      // network
      // unavailability of the remote system
      System.out.println(String.format("Failed to deliver the message to
the HUB. ", e.getMessage()));
      throw new HubClientException(e);
    }
  }
```

If an error occurs, the error will be returned as SOAP exception. Possible errors detected include:

- The system making the request is not in the HUB
- The requester is not authorized to use the specified “From” field

6.7 GetActiveNppos

This operation is a simple query that return all the active Country entities of the HUB, with only the Country code, the Send and Receive flags. Such flags may be used by a client application to automate the sending or receiving from the relevant country depending on their status on the HUB.

This operation can be used to automatically verify that a country is configured and ready to receive or send ephytos.

6.8 ValidatePhytoXML

This operation is exposing the functionality of the AdminConsole for validating the XML against the latest ePhyto schema (based on the UN/CEFACT version 17A).

This can be used to collect and warn issues on incoming as well as outgoing messages.

When used from SoapUI, it is suggested to wrap the XML to validate into a <![CDATA[….]]> element to be able to copy paste the source text as it is.

Code examples are not provided here as they are just a different operation call from the ones described above. The result of the operation will return an array of validation results like the following

```
<ns3:ValidatePhytoXMLResult>
        <area>MandatoryElements</area>
        <field>SPSExchangedDocument.IssueDateTime.DateTimeString</field>
        <level>SEVERE</level>
        <msg>Issue date is mandatory field</msg>
</ns3:ValidatePhytoXMLResult>
```

The field indicate the source element of the issue, the msg indicates a descriptive message of the issue.

Possible Areas are the following:

- MandatoryElements (elements that must be there as part of the document structure)
- Mapping (issues related to the mapping of the ePhyto to the schema)
- Schema (issues related to non-compliances to the XML schema)

Possible Levels are the following:

- SEVERE : lead to issues in reading and visualizing the certificate
- WARNING : not leading to issues, may need some revision on how the XML is produced
- INFO: optimization level changes may be applied

6.9 DeliverPhytoEnvelope

This operation is exposing a new entity in the HUB, the ePhytoEnvelope that is inherited from the EnvelopeHeader and instead of having a String Content element, as defined in the Envelope, has the SPSCertificate defined with the schema used by the validation tool (based on the UN/CEFACT).

The operation, similarly to the DeliverEnvelope will require all the information as defined above and a valid SPSCertificate entity defined. The operation before queuing the content for the delivery will perform a validation and if any SEVERE level issues are found, it will stop the delivery.

Usage of this operation is following exactly the same workflow as defined for DeliverEnvelope and it may leverage the client application from compiling the XML, in actually filling in the required information from the existing entities defined in the exporter application.

The sample code here is similar to the one defined above for the DeliveryEnvelope operation. Instead of setting an XML in the content, the entire SPSCertificate object should be filled, depending on the available information (such scenario may vary depending on how the system is implemented to fulfill dynamically the various phytosanitary information).

This operation is suggested for National Systems that do not have yet the transformation to XML and can immediately map the internal information to the standardized SPSCertificate that is exposed in the web service and available in the client software.

6.10 GetAvailableChannels

This operation is a simple query that return all the active Channels registered in the HUB, with the code, the name, the Send and Receive flags. Similarly, to the GetActiveNppos the flags are indicative of the capabilities of the connected system.

The channel codes can be used in the delivery operations to forward the envelope to systems that are not destination or transit country systems, see XML sample below of an envelope header with a forwarding channel:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:eph="http://ephyto.ippc.int/"
xmlns:hub="http://ephyto.ippc.int/HUB.Entities">
  <soapenv:Header/>
  <soapenv:Body>
    <eph:DeliverEnvelope>
      <eph:env>
        <hub:From>AR</hub:From>
        <hub:To>US</hub:To>
        <hub:CertificateType>851</hub:CertificateType>
        <hub:CertificateStatus>70</hub:CertificateStatus>
        <hub:NPPOCertificateNumber>…</hub:NPPOCertificateNumber>
        <hub:Forwardings>
           <hub:EnvelopeForwarding>
            <hub:Code>xCB01</hub:Code>
          </hub:EnvelopeForwarding>
        </hub:Forwardings>
        <hub:Content><![CDATA[<?xml …]]></hub:Content>
      </eph:env>
    </eph:DeliverEnvelope>
  </soapenv:Body>
</soapenv:Envelope>
```

In the example above the code xCB01 represent an existing channel.

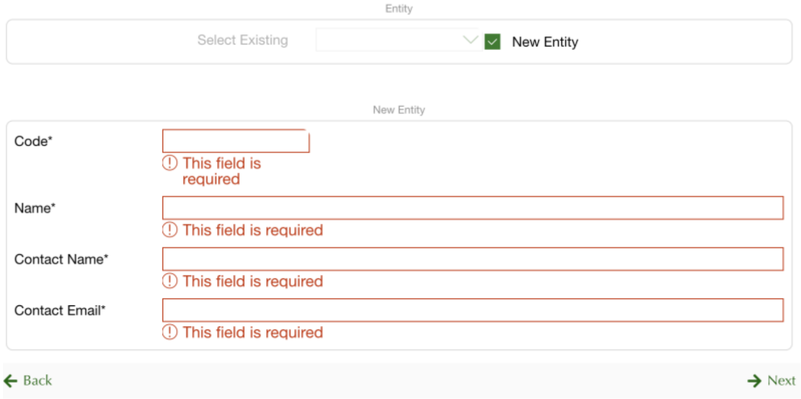
The result from the HUB will be something like the following:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:DeliverEnvelopeResponse xmlns:ns2="http://ephyto.ippc.int/" xmlns:ns3="http://ephyto.ippc.int/HUB.Entities"
xmlns:ns4="urn:un:unece:uncefact:data:standard:ReusableAggregateBusinessInformationEntity:21"
xmlns:ns5="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:21"
xmlns:ns6="urn:un:unece:uncefact:data:standard:SPSCertificate:17">
      <ns2:DeliverEnvelopeResult xsi:type="ns3:Envelope" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <ns3:From>AR</ns3:From>
        <ns3:To>US</ns3:To>
        <ns3:CertificateType>851</ns3:CertificateType>
        <ns3:CertificateStatus>70</ns3:CertificateStatus>
        <ns3:NPPOCertificateNumber>…</ns3:NPPOCertificateNumber>
        <ns3:hubDeliveryNumber>ARUSM2004270738011959481</ns3:hubDeliveryNumber>
        <ns3:Forwardings>
          <ns3:EnvelopeForwarding>
            <ns3:Code>xCB01</ns3:Code>
            <ns3:HubDeliveryNumber>ARxCB01M2004270738011955240</ns3:HubDeliveryNumber>
          </ns3:EnvelopeForwarding>
        </ns3:Forwardings>
      </ns2:DeliverEnvelopeResult>
    </ns2:DeliverEnvelopeResponse>
  </soap:Body>
</soap:Envelope>
```

The envelope will be then delivered in parallel to the system of the channel in the same way as it is delivered to the receiving country.

Following are the steps to create a Country only Channel (visibility of the channel will be restricted to the connected country system):

- Enter the HUB Console and go to the Configuration menu link
- Click on top right the channel rules button Channel Rules
- Click the add button + Add
- Check the new entity box and fill the fields



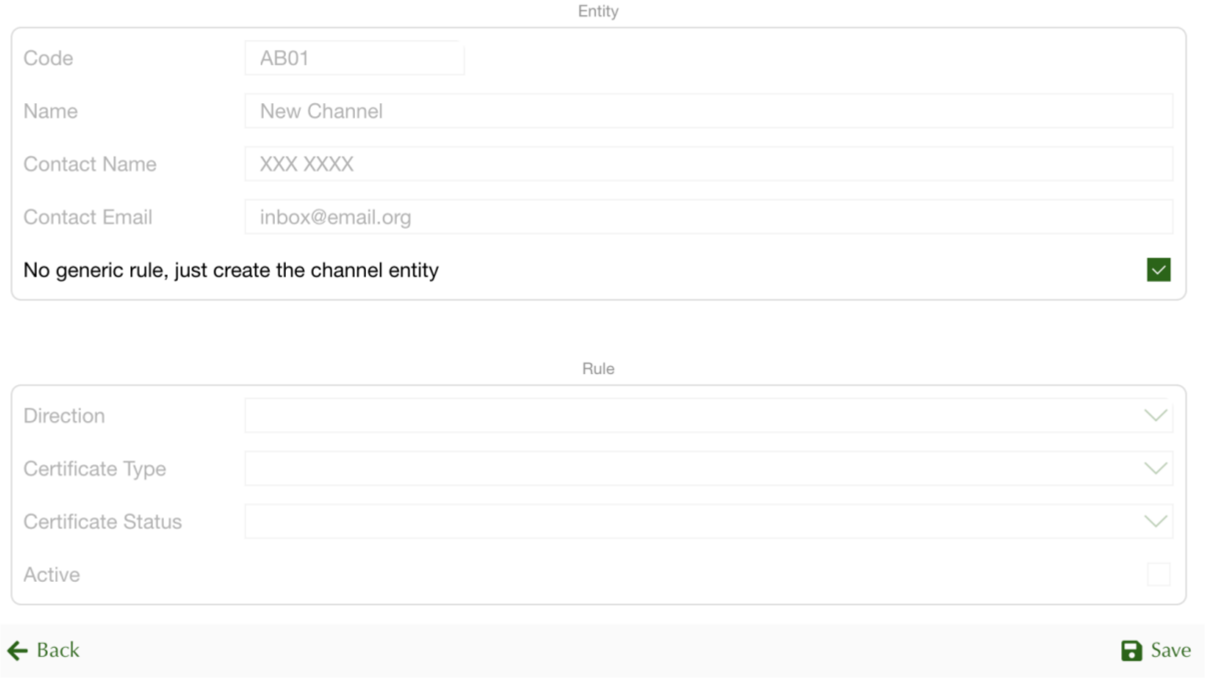
Entity

Select Existing ☑ New Entity

New Entity

Code*	ⓘ This field is required
Name*	ⓘ This field is required
Contact Name*	ⓘ This field is required
Contact Email*	ⓘ This field is required

- Click Next and verify the creation of the record



Entity

Code	AB01
Name	New Channel
Contact Name	XXX XXXX
Contact Email	inbox@email.org

No generic rule, just create the channel entity ☑

Rule

Direction	
Certificate Type	
Certificate Status	
Active	□

- Leave the default and Save.

By saving the record an account and entity profile will be created in the HUB, the contact email will receive the account credentials, the onboarding guide and this API document to start developing the integration with the HUB.

Channel Rules can be created to automatically forward Incoming or Outgoing envelopes of the country. The country can specify the type and status or automate all the envelopes to be sent to another system, in the scenario of a country system that need to receive all the envelopes that are sent or received. The combination of type and status will allow countries to specify the set of envelopes that are for import or export.

6.11 Receiving a PUSH delivery

In order to receive a PUSH delivery, the receiving system must have an endpoint ready for the HUB to connect.

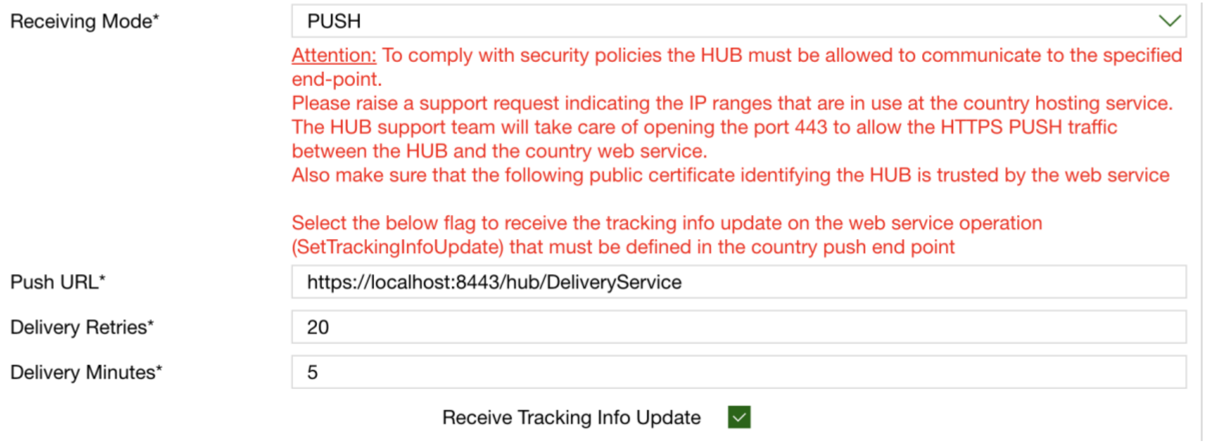
The system must use the HUB WSDL file in order to generate the needed sources, and develop the needed functionalities to receive in the DeliveryEnvelope operation the envelopes and confirm the receipt using the acknowledge HUB operations (`acknowledgeEnvelopeReceipt` and `advancedAcknowledgeEnvelopeReceipt`) described above with the corresponding PULL delivery operations.

If there is an error and the PUSH endpoint application wants to receive again the envelope the end-point result shoud have a envelope header with the tracking info set as FailedDelivery (the HUB in this case will continue to try and send the envelope until it is acknowledged or the answer from the enpoint is successful). If no FailedDelivery is responded to the HUB, the system will lock the envelope waiting for the acknowledge.

The system administrator will need to communicate (using the support request described above) the IP ranges that are hosting the PUSH web service endpoint. After the opening of the ports on the HUB the system administrator will receive a confirmation email with the SSL client certificate used by the HUB to authenticate.

Additionally, the HUB connection profile can be configured to Receive Tracking Info Update via PUSH. This will mean that the HUB will send the envelope header to the operation **SetTrackingInfoUpdate** on the changing of the tracking info, from pending delivery to any of the final delivery or failure state.

See below NPPO settings related to the PUSH receiving mode.



Receiving Mode* PUSH

Attention: To comply with security policies the HUB must be allowed to communicate to the specified end-point.
Please raise a support request indicating the IP ranges that are in use at the country hosting service.
The HUB support team will take care of opening the port 443 to allow the HTTPS PUSH traffic between the HUB and the country web service.
Also make sure that the following public certificate identifying the HUB is trusted by the web service

Select the below flag to receive the tracking info update on the web service operation (SetTrackingInfoUpdate) that must be defined in the country push end point

Push URL* https://localhost:8443/hub/DeliveryService

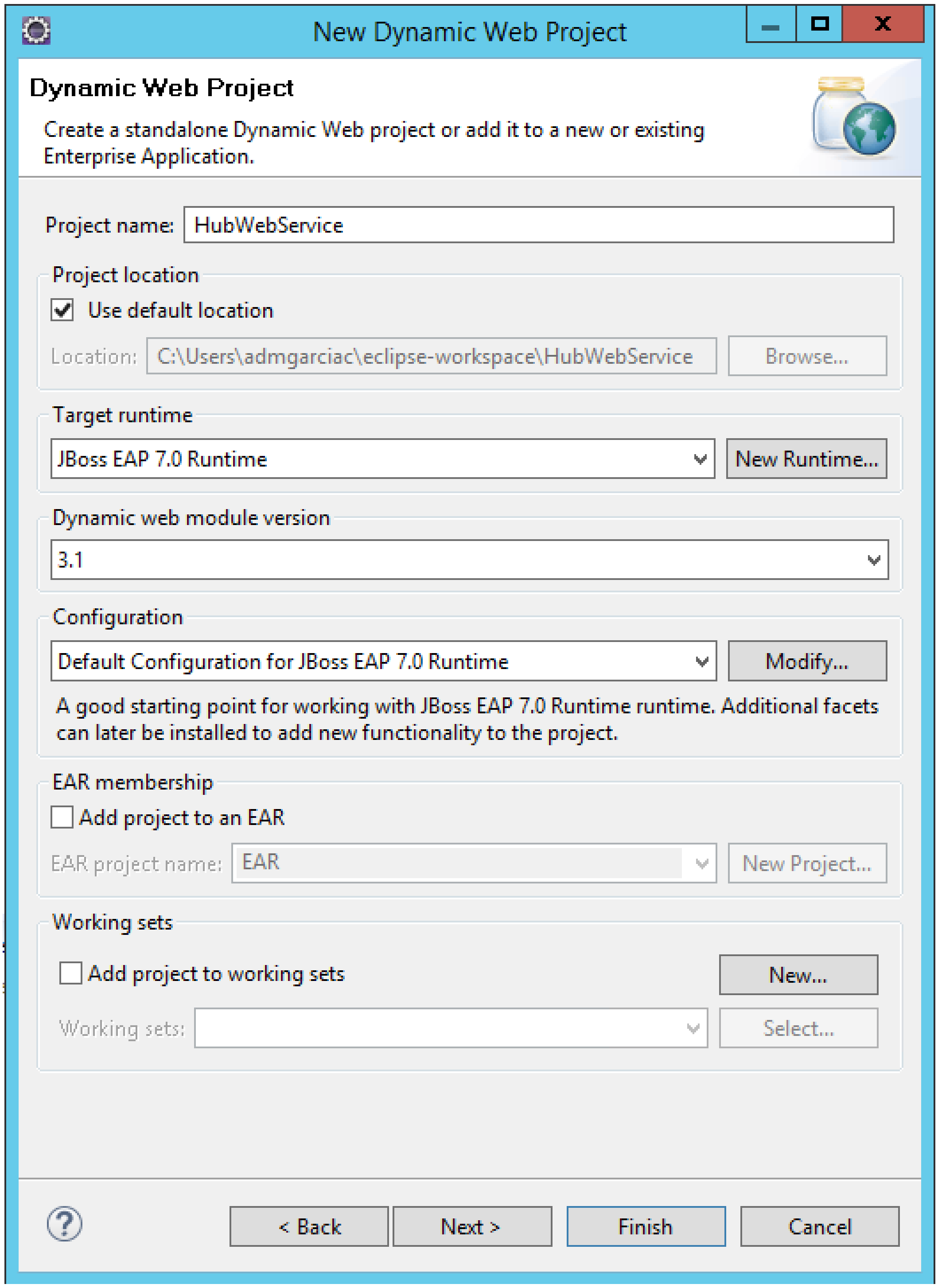
Delivery Retries* 20

Delivery Minutes* 5

Receive Tracking Info Update ☑

Here below a sample on how to create a basic endpoint using Eclipse and Apache Axis.

First, create a new Dynamic Web project in Eclipse (here we use JBoss as target runtime, please use the runtime that best suit your needs)



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: HubWebService

Project location

☑ Use default location

Location: C:\Users\admgarciac\eclipse-workspace\HubWebService

Target runtime: JBoss EAP 7.0 Runtime

Dynamic web module version: 3.1

Configuration: Default Configuration for JBoss EAP 7.0 Runtime

A good starting point for working with JBoss EAP 7.0 Runtime runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

□ Add project to an EAR

EAR project name: EAR

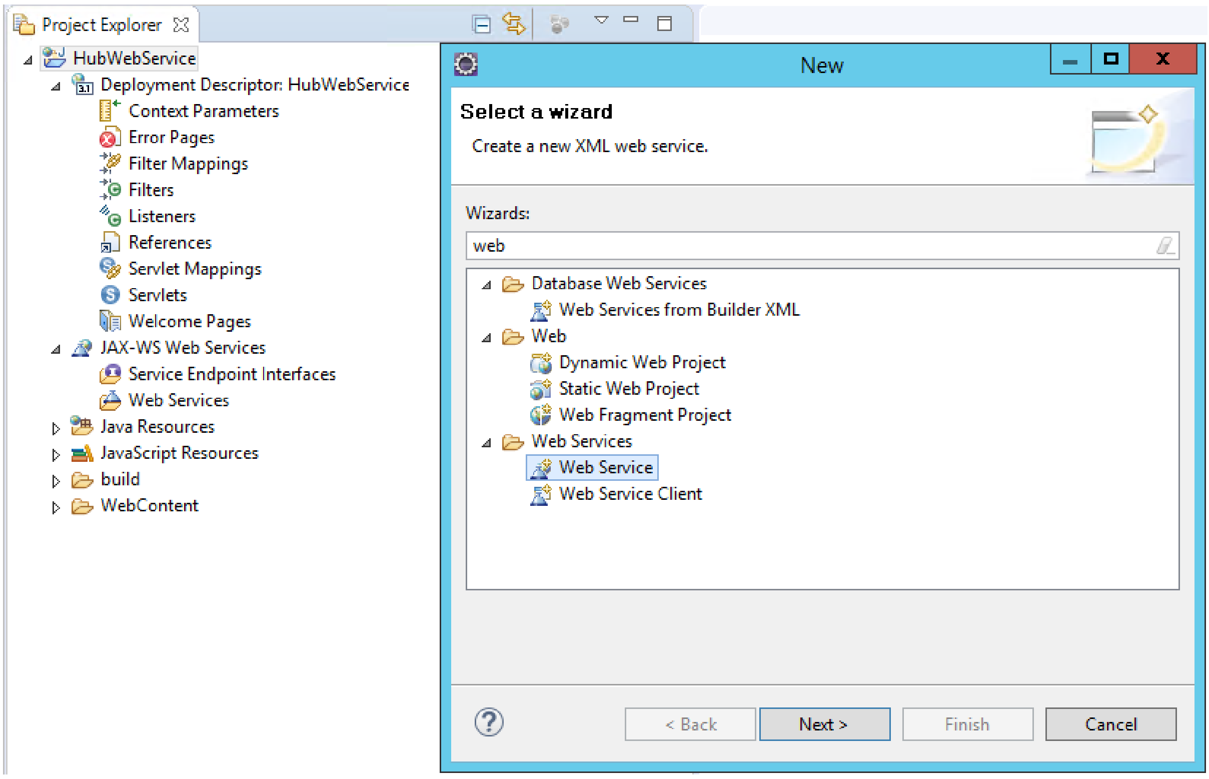
Working sets

□ Add project to working sets

Working sets:

You can click “Finish” in this dialog.

Once the project is there (skip the project creation if you want to implement the service into an existing project), you will need to create the classes implementing the web service interface. For this, we will add a new Web Service to the project by right clicking the project name and then “new” and “Other…”



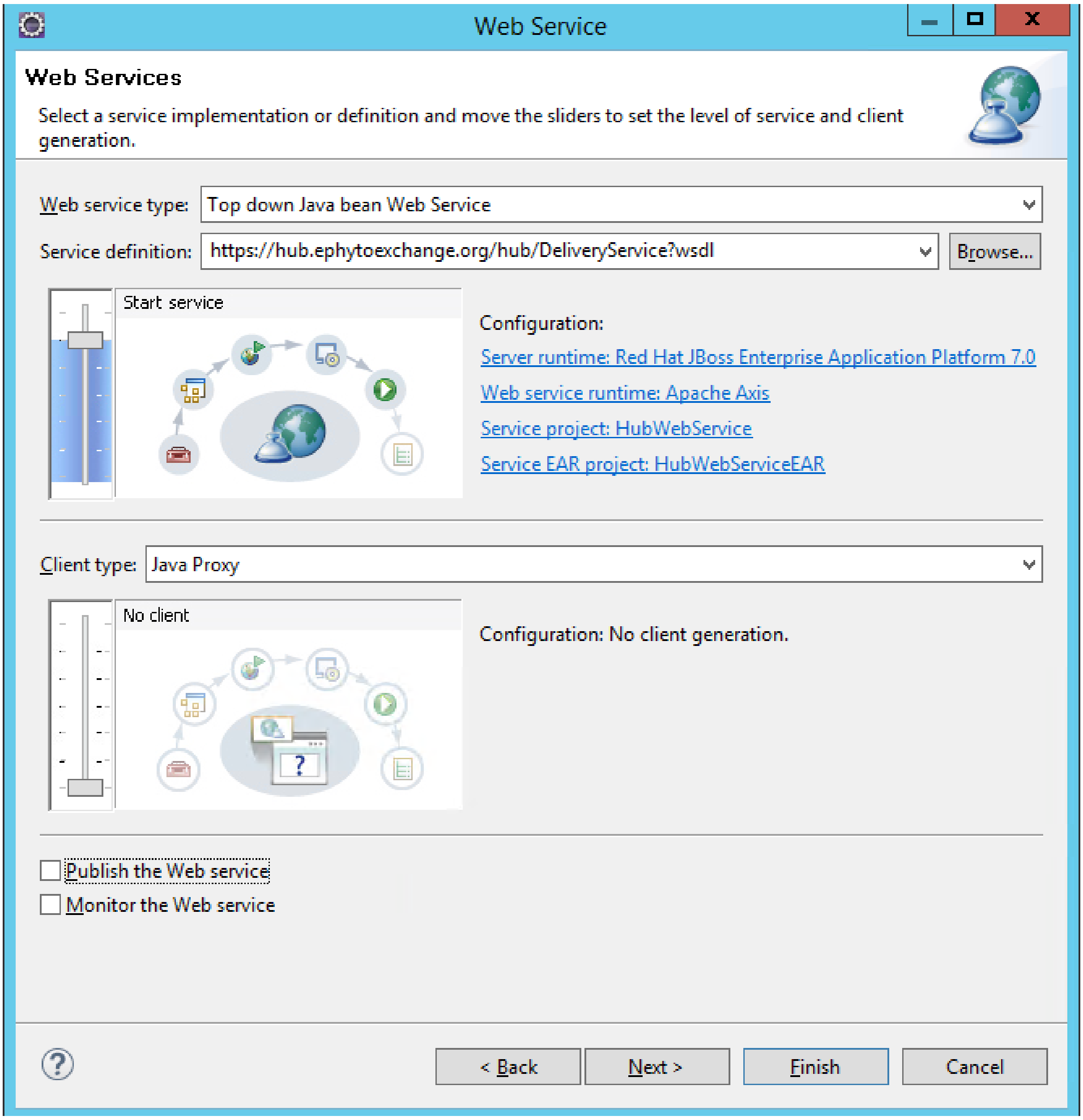
Select a wizard

Create a new XML web service.

Wizards:

web

Select “Web Service” and click “Next”



Web Service

Web Services

Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Top down Java bean Web Service

Service definition: https://hub.ephytoexchange.org/hub/DeliveryService?wsdl

Configuration:

Server runtime: Red Hat JBoss Enterprise Application Platform 7.0

Web service runtime: Apache Axis

Service project: HubWebService

Service EAR project: HubWebServiceEAR

Client type: Java Proxy

Configuration: No client generation.

☐ Publish the Web service

☐ Monitor the Web service

As we already have the WSDL file, select “Top down Java bean Web Service”

After that enter the Hub WSDL URL address in the service definition:

https://hub.ephytoexchange.org/hub/DeliveryService?wsdl

We will use “Apache Axis” and JBoss as our server for the deployment. If you have a different Application Server, just select it by clicking in the “Server runtime” link. Make sure that the Application Server is running and click “Finish”.

When the process finishes, Eclipse will open the file: DeliveryServiceSoapBindingImpl.java this is where the code has to be completed. In this case, we only need to implement the “deliverEnvelope” method, which is the one that will be called by the PUSH service in the HUB.

Java

```
    public _int.ippc.ephyto.HUB_Entities.EnvelopeHeader
deliverEnvelope(_int.ippc.ephyto.HUB_Entities.Envelope env) throws
java.rmi.RemoteException, _int.ippc.ephyto.HubWebException {
        saveEnvelope(env);
        return env;
    }

    private void saveEnvelope(_int.ippc.ephyto.HUB_Entities.Envelope env) {
      // do checks and store the envelope in the suitable place

      // acknowledge the reception
      HubClient.acknowledge(env);
    }
```

Save the envelope and then acknowledge the reception of the envelope to the HUB so it can be marked as delivered.

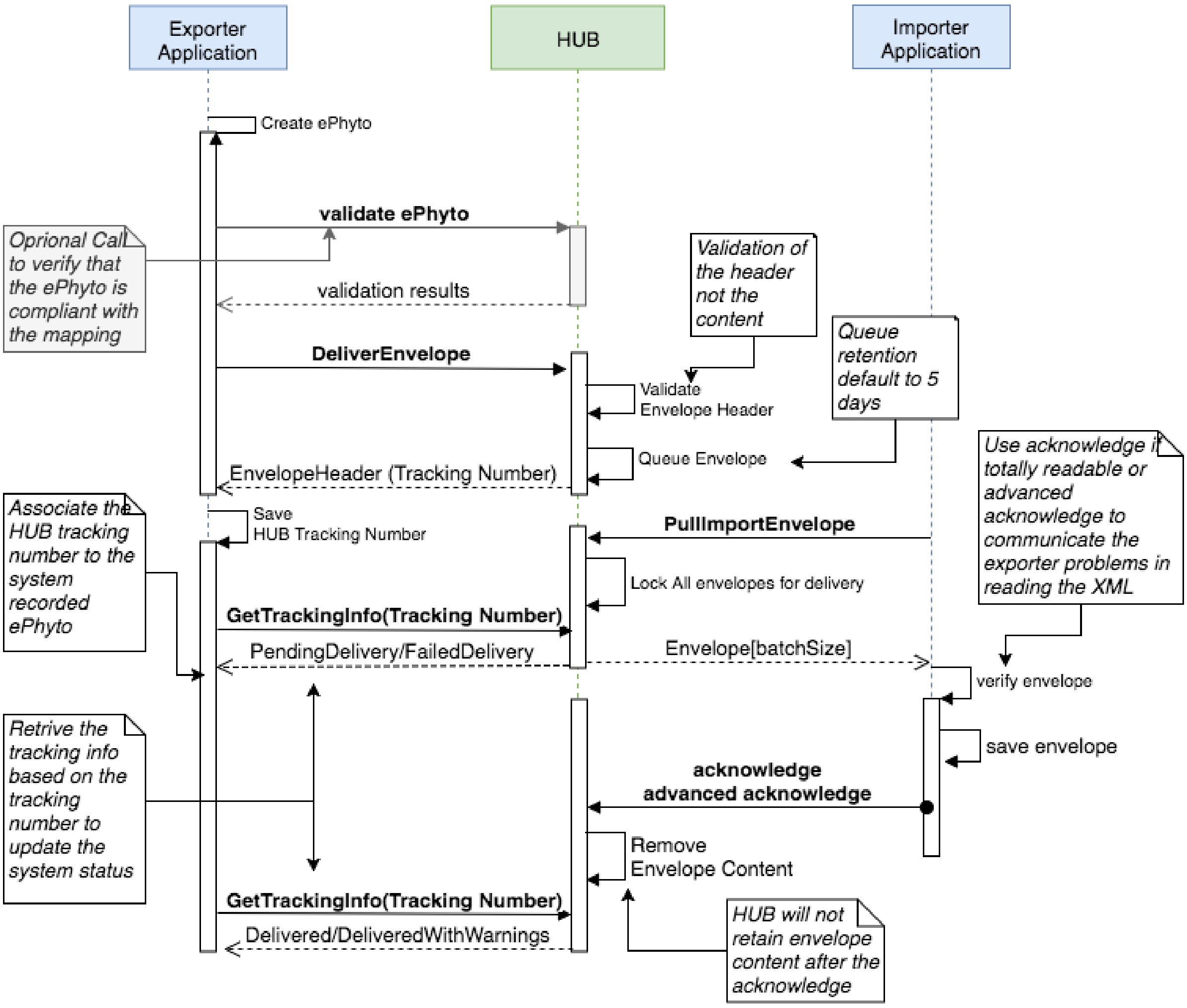
Note that the HubClient is referring to the object implementing the connection to the HUB web services.

In the example above we do not provide guidelines on how to setup the client certificate authentication as it may vary considerably depending on the underlying platform and infrastructure. To implement the push endpoint the receiving system should accept the provided HUB certificate for the client authentication (the client certificate will be provided with the response of the opening of the required ports to the country hosting IP ranges).

7. Sequence Diagrams

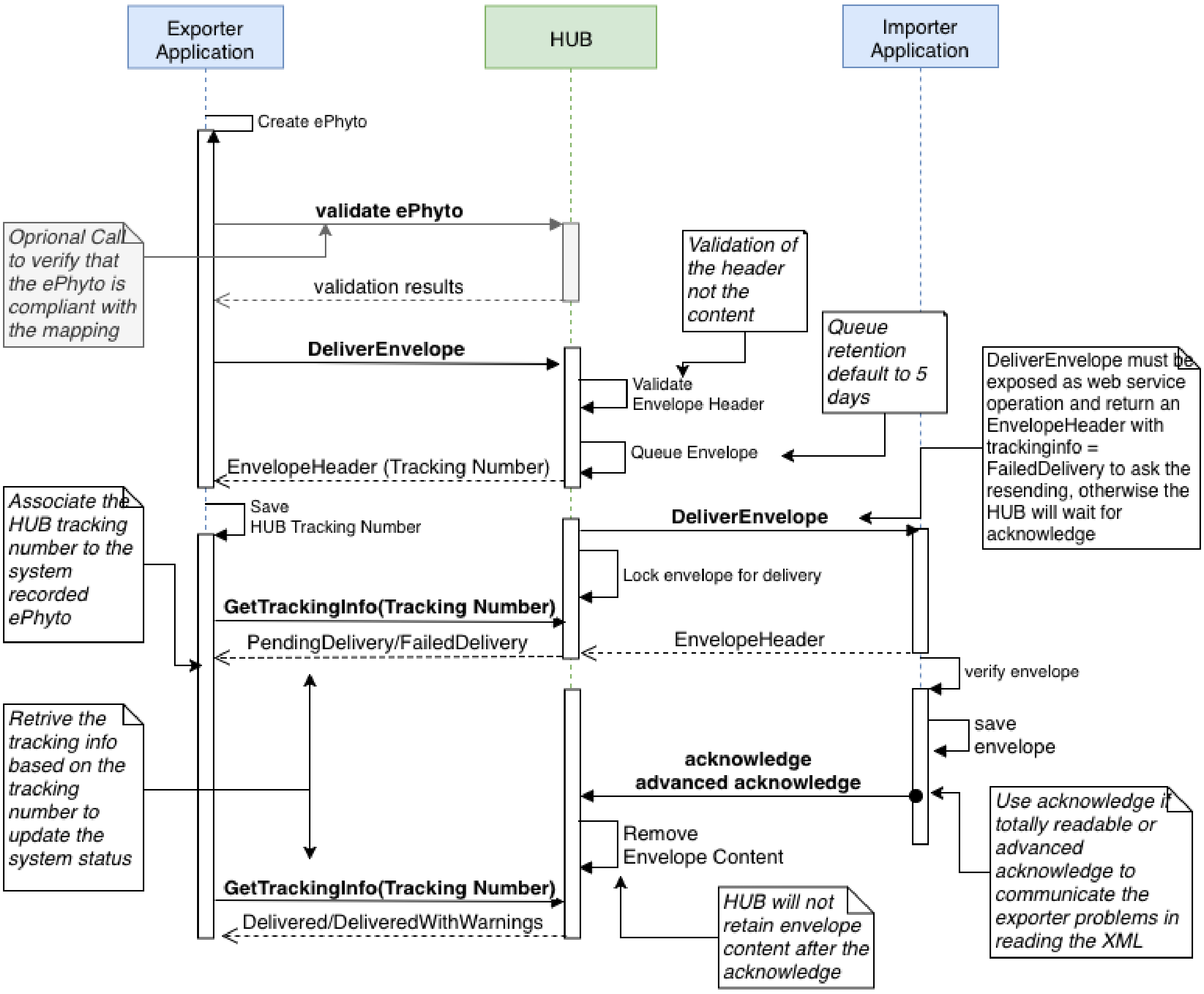
7.1 Deliver with PULL

Following a sequence diagram defining the optimal envelope delivery process interaction between the client applications and the HUB using the PULL receiving type.


Exporter Application
HUB
Importer Application
Create ePhyto
validate ePhyto
Oprional Call to verify that the ePhyto is compliant with the mapping
validation results
Validation of the header not the content
DeliverEnvelope
Validate Envelope Header
Queue retention default to 5 days
Queue Envelope
EnvelopeHeader (Tracking Number)
Use acknowledge if totally readable or advanced acknowledge to communicate the exporter problems in reading the XML
Associate the HUB tracking number to the system recorded ePhyto
Save HUB Tracking Number
PullImportEnvelope
Lock All envelopes for delivery
GetTrackingInfo(Tracking Number)
PendingDelivery/FailedDelivery
Envelope[batchSize]
verify envelope
save envelope
Retrive the tracking info based on the tracking number to update the system status
acknowledge
advanced acknowledge
Remove Envelope Content
GetTrackingInfo(Tracking Number)
Delivered/DeliveredWithWarnings
HUB will not retain envelope content after the acknowledge

7.2 Deliver with PUSH

Following a sequence diagram defining the envelope delivery process interaction between the client applications and the HUB using the PUSH receiving type.


Exporter Application
HUB
Importer Application
Create ePhyto
validate ePhyto
Oprional Call to verify that the ePhyto is compliant with the mapping
validation results
Validation of the header not the content
Queue retention default to 5 days
DeliverEnvelope
Validate Envelope Header
DeliverEnvelope must be exposed as web service operation and return an EnvelopeHeader with trackinginfo = FailedDelivery to ask the resending, otherwise the HUB will wait for acknowledge
Queue Envelope
EnvelopeHeader (Tracking Number)
Associate the HUB tracking number to the system recorded ePhyto
Save HUB Tracking Number
DeliverEnvelope
Lock envelope for delivery
GetTrackingInfo(Tracking Number)
PendingDelivery/FailedDelivery
EnvelopeHeader
verify envelope
save envelope
Retrive the tracking info based on the tracking number to update the system status
acknowledge
advanced acknowledge
Use acknowledge if totally readable or advanced acknowledge to communicate the exporter problems in reading the XML
Remove Envelope Content
GetTrackingInfo(Tracking Number)
HUB will not retain envelope content after the acknowledge
Delivered/DeliveredWithWarnings

8. Testing with Soap UI

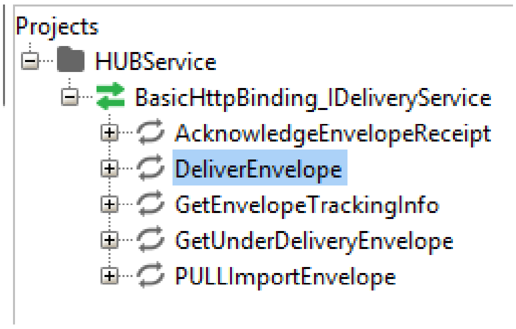
Note: Each time that SOAP UI is started, steps 9 to 15 have to be done again.

Please follow the next steps in order to test with SOAPUI:

1. Download and installs SOAP UI. We are using version 5.3
2. Go to the Installation folder bin directory and open the file SOAPUI-5.3.0.vmoptions. On windows machines, the file is located in “C:\Program Files\SmartBear\SoapUI-5.3.0\bin”, on Mac is under the /Applications/SoapUI-5.3.0.app/Contents/vmoptions.txt. You have to edit this file with Administrator rights.
3. Include this line at the end of the file (the hub web service accepts only TLSv1.1 and TLSv1.2):

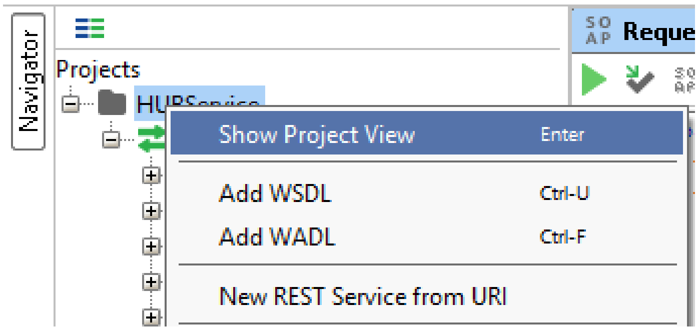
-Dsoapui.https.protocols=TLSv1.1,TLSv1.2

4. Save the file and open or reopen SOAP UI.
5. Go to File New SOAP Project.
6. In “Project Name” field, choose a descriptive project name.
7. In “Initial WSDL” field, choose the provided URL for the endpoint (this URL should finish in “?wsdl”), or choose the wsdl file, if you received the file or you saved the wsdl file in your computer.
8. After clicking OK, SOAP UI will generate some templates with the operation requests. Here you can see an example of this generated template requests:

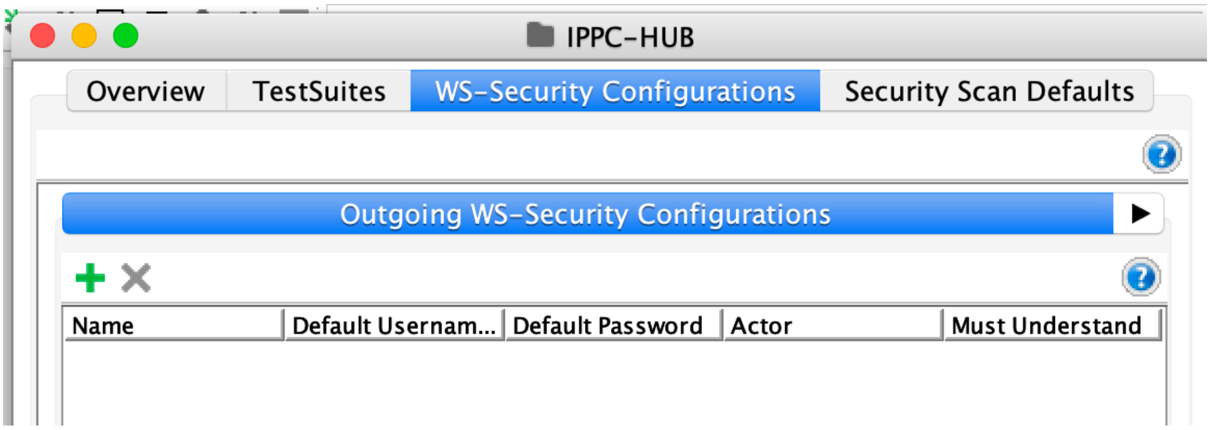


Projects
- HUBService
  - BasicHttpBinding_IDeliveryService
    - AcknowledgeEnvelopeReceipt
    - DeliverEnvelope
    - GetEnvelopeTrackingInfo
    - GetUnderDeliveryEnvelope
    - PULLImportEnvelope

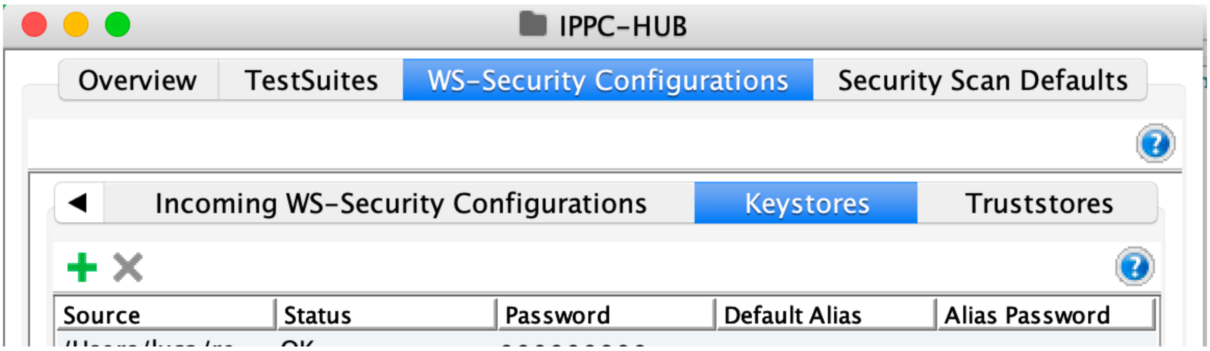
9. Right click in the project name, in our case “HUBService”. And choose the option “Show Project View”.



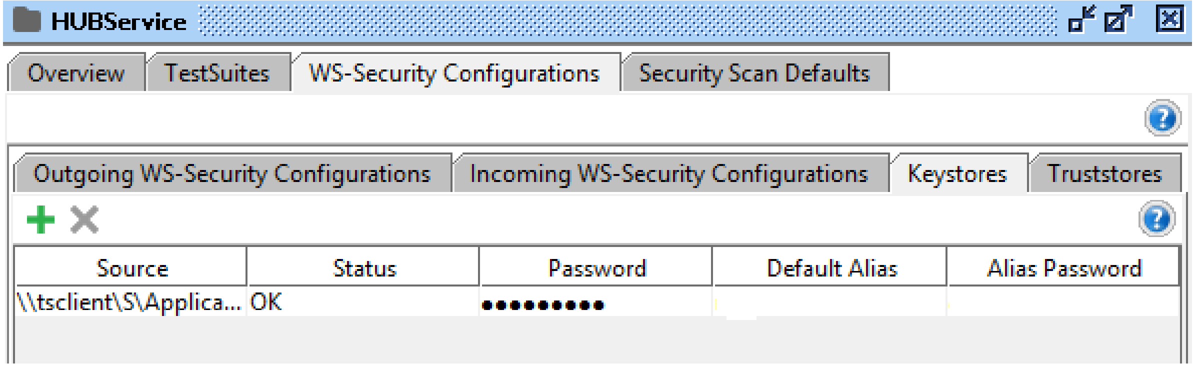
10. In the new dialog go to "WS-Security Configurations" tab



11. Inside this tab, go to the "Keystores" tab.

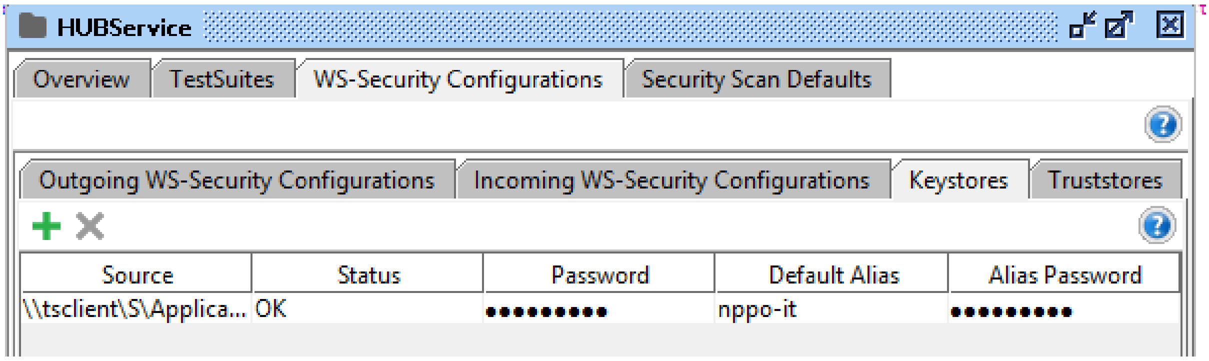


12. Click on the green plus symbol button in order to add your client certificate. This green plus symbol button is at the top left hand side corner of the window.
13. In the bowser window, choose your certificate keystore with P12 extension and format.
14. Write the keystore password in the prompt windows.
15. A new keystore rows appears in the window with Status OK.


HUBService
Overview
TestSuites
WS-Security Configurations
Security Scan Defaults
Outgoing WS-Security Configurations
Incoming WS-Security Configurations
Keystores
Truststores

Source	Status	Password	Default Alias	Alias Password
\\tsclient\S\Applica...	OK	•••••••••		

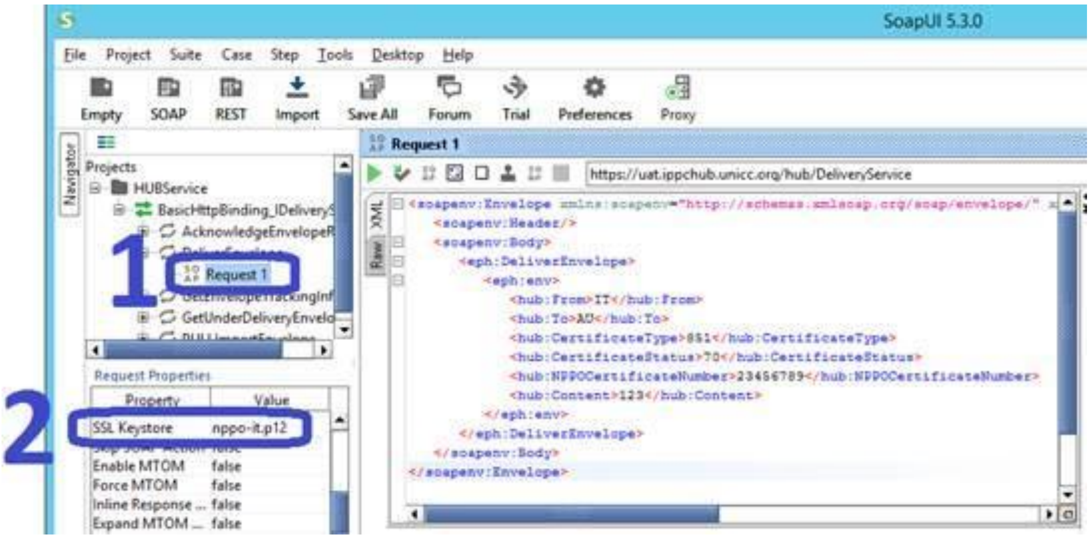
16. Add you certificate alias and password, in our case the certificate alias is "nppo-it"


HUBService
Overview
TestSuites
WS-Security Configurations
Security Scan Defaults
Outgoing WS-Security Configurations
Incoming WS-Security Configurations
Keystores
Truststores

Source	Status	Password	Default Alias	Alias Password
\\tsclient\S\Applica...	OK	●●●●●●●●●	nppo-it	●●●●●●●●●

17. Close this windows and click in DeliveryEnvelope request (the following procedure is valid for every request). A Request template appears:

18. Include your certificate in the request, in the filed SSL Keystore (do the same for the rest of the requests that you want to test):

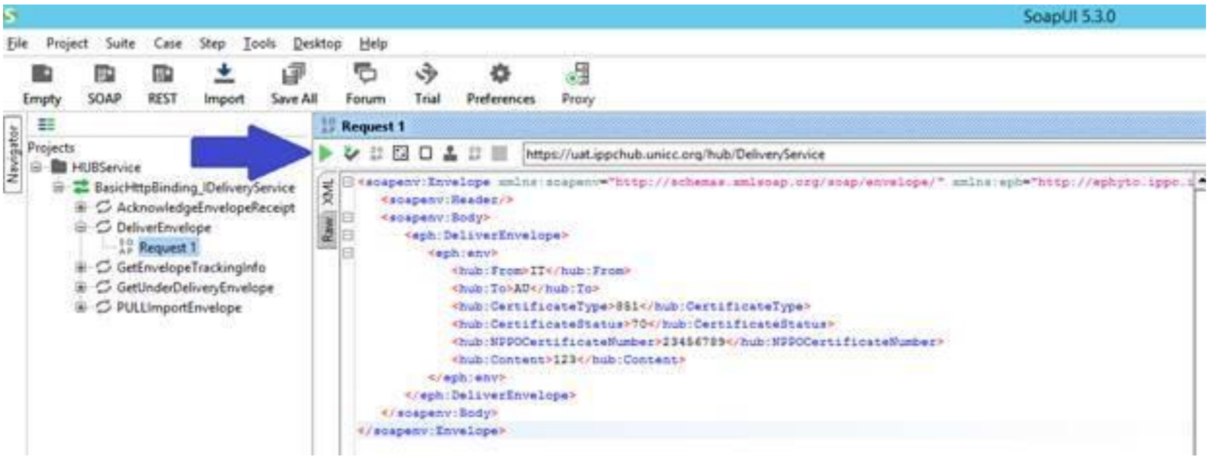


Property	Value
SSL Keystore	nppo-it.p12
Enable MTOM	false
Force MTOM	false
Inline Response ...	false
Expand MTOM ...	false

https://uat.ippchub.unicc.org/hub/DeliveryService

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" x
   <soapenv:Header/>
   <soapenv:Body>
      <eph:DeliverEnvelope>
         <eph:env>
            <hub:From>IT</hub:From>
            <hub:To>AU</hub:To>
            <hub:CertificateType>851</hub:CertificateType>
            <hub:CertificateStatus>70</hub:CertificateStatus>
            <hub:NPPOCertificateNumber>23456789</hub:NPPOCertificateNumber>
            <hub:Content>123</hub:Content>
         </eph:env>
      </eph:DeliverEnvelope>
   </soapenv:Body>
</soapenv:Envelope>
```

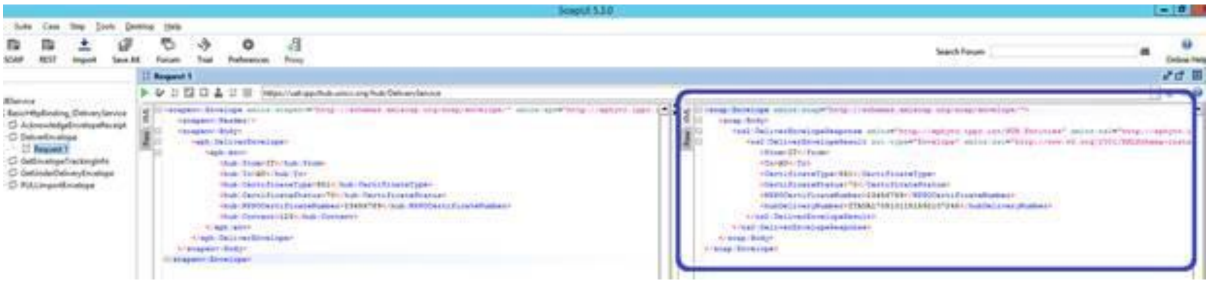
19. Fill the template with valid data like in the example, and click in RUN (remember “hub:From” in the code of your country and should be the same as your certificate country code):


SoapUI 5.3.0
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Navigator
Projects
HUBService
BasicHttpBinding_IDeliveryService
AcknowledgeEnvelopeReceipt
DeliverEnvelope
Request 1
GetEnvelopeTrackingInfo
GetUnderDeliveryEnvelope
PULLImportEnvelope
Request 1
XML
Raw

https://uat.ippchub.unicc.org/hub/DeliveryService

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:eph="http://ephyto.ippc.i
   <soapenv:Header/>
   <soapenv:Body>
      <eph:DeliverEnvelope>
         <eph:env>
            <hub:From>IT</hub:From>
            <hub:To>AU</hub:To>
            <hub:CertificateType>851</hub:CertificateType>
            <hub:CertificateStatus>70</hub:CertificateStatus>
            <hub:NPPOCertificateNumber>23456789</hub:NPPOCertificateNumber>
            <hub:Content>123</hub:Content>
         </eph:env>
      </eph:DeliverEnvelope>
   </soapenv:Body>
</soapenv:Envelope>
```

20. You receive the response:


SoapUI 5.3.0
Suite Case Step Tools Desktop Help
SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Request 1
XML
Raw

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <DeliverEnvelopeResponse xmlns="[illegible]" xmlns:[illegible]
         <DeliverEnvelopeResult [illegible]
            <From>IT</From>
            <To>AU</To>
            <CertificateType>851</CertificateType>
            <CertificateStatus>70</CertificateStatus>
            <NPPOCertificateNumber>23456789</NPPOCertificateNumber>
            <hubDeliveryNumber>[illegible]</hubDeliveryNumber>
         </DeliverEnvelopeResult>
      </DeliverEnvelopeResponse>
   </soap:Body>
</soap:Envelope>
```

21. If you save the project and close SOAP UI, you need to repeat steps 9 to 15 when you open SOAP UI again.