Java CAPS 5.1.3
Field Implementation Notes

WS-Security 1.0 (2004) Username Token Profile

Michael.Czapski@sun.com
October 15, 2007

Table of Contents

Introduction

As at release 5.1.3 Java CAPS supports WS-Security 1.0 (2004) Username Token Profile. Java CAPS eInsight-exposed web service can be configured to require Username/Password credentials before performing its programmed function.

Implementation of Username Token-based access control to a web service requires:

1. development of the service implementation
2. configuration of the web service external system to require Username Token
3. creation of one or more Integration Server users as authentication targets
4. configuration of web service access associating credentials with the service
5. deployment of the service to the Integration Server which hosts the user

Implementation of Username Token-based access by a web service client requires:

1. development of the service client implementation
2. configuration of the web service client external system to supply credentials
3. deployment of the service to an Integration Server

The following sections deal with each of the topics in turn by developing and deploying a Java CAPS eInsight service implementation, testing this implementation using SoapUI, developing a Java CAPS eInsight Client implementation and exercising the combined solution.

Exports of Java CAPS projects discussed in this document are available in the same place as this document.

Implement Service

Any eInsight web service can be configured to require WS-Security Username Token. The minimalist service discussed below will be implemented to facilitate further discussion. The service will accept a string as an input message and will return the content of that string, prefixed and suffixed with two asterisks (**) as the output message, also a string. Input of “AAA” with result in output of “**AAA**”.

The following XML Schema document describes the XML Element of type string which will be used as input and output messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://mcz02:12000/XSDWSUserAuthString"
    targetNamespace="http://mcz02:12000/XSDWSUserAuthString">

    <xsd:element name="elmString" type="xsd:string"/>
</xsd:schema>
```

The following Web Service Description Language document, which uses the XML Schema shown above, describes the service interface.

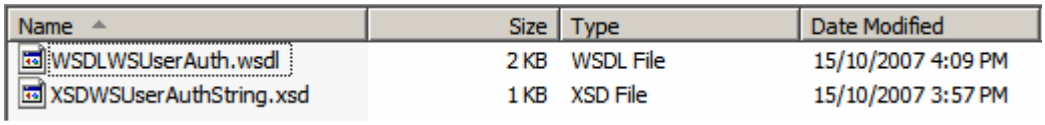
Note that whilst eInsight-based web service is constructed in a “contract first” manner, starting with the WSDL, the Java Collaboration web service implementation starts with the input and output message structures, supplied as XSD Nodes, and results in a WSDL being generated during service build.

Note that whilst implementing a service using a Java Collaboration is different from implementing a service using an eInsight business process, configuring WS-Security Username Token Profile support is identical.

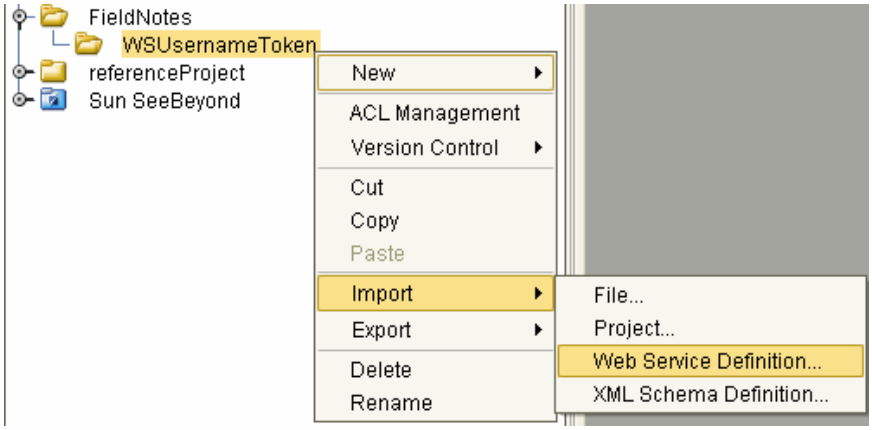
```
<definitions
             targetNamespace="http://mcz02:12000/wsdlWSUserAuth"
             name="wsdlWSUserAuth"
             xmlns:tns="http://mcz02:12000/wsdlWSUserAuth"
             xmlns:ns0="http://mcz02:12000/XSDWSUserAuthString"
             xmlns:imps0="http://mcz02:12000/XSDWSUserAuthString"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

    <import namespace="http://mcz02:12000/XSDWSUserAuthString"
            location="XSDWSUserAuthString.xsd"/>
    <types>
    </types>

    <message name="msgInput">
        <part name="prtInput"
              element="ns0:elmString"/>
    </message>
    <message name="msgOutput">
        <part name="prtOutput"
              element="ns0:elmString"/>
    </message>

    <portType name="PortTypewsdlWSUserAuth">
        <operation name="opWSUserAuth">
            <input name="inputMsg"
```

```
                    message="tns:msgInput"/>
            <output name="outputMsg"
                    message="tns:msgOutput"/>
        </operation>
    </portType>

</definitions>
```

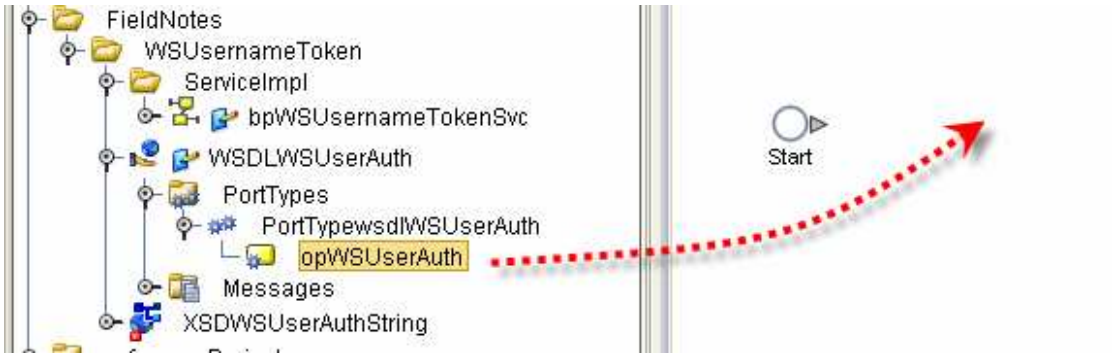
The XSD and the WSDL are assumed to exist, in the same directory, in the file system.

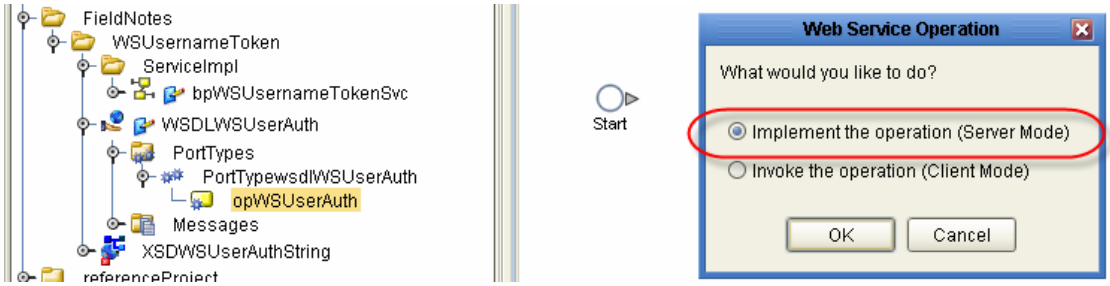


Let's create a project, WSUsernameToken, for the service implementation and import the WSDL file. Importing the WSDL file also imports the XSD file.

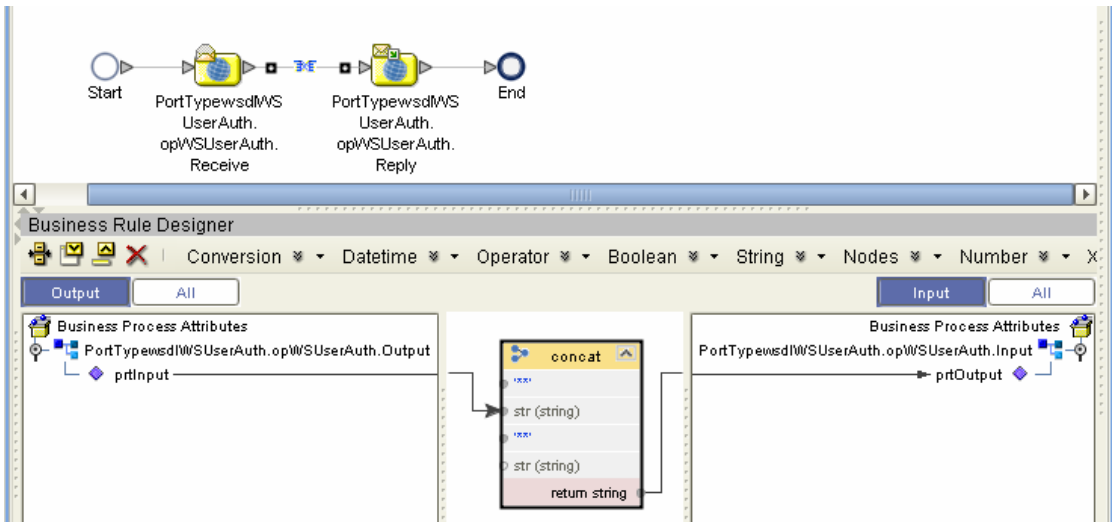


Let's create a subprocess, ServiceImpl, and new eInsight Business Process, within that subprocess, bpWSUsernameTokenSvc. Let's drag the WSDL operation onto the eInsight process canvas and choose to implement the operation opWSUserAuth.

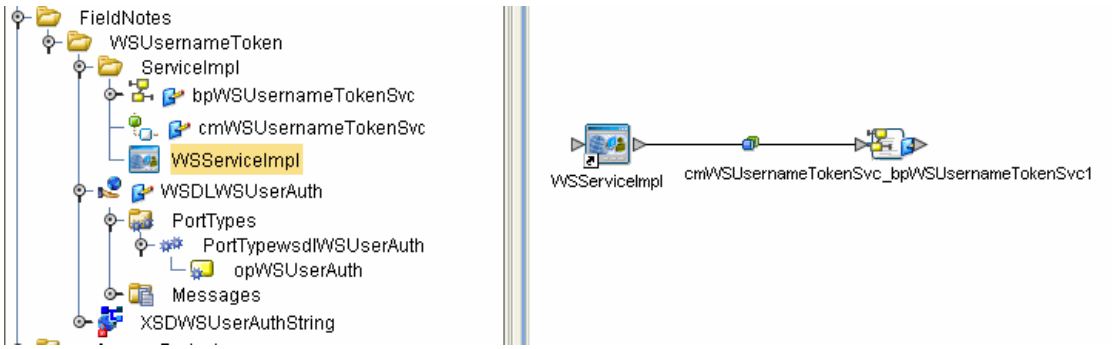




Connect all activities and concatenate request output with asterisks to form response input.



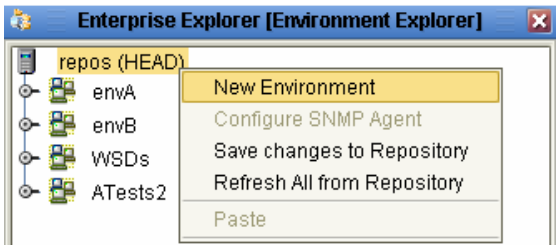
Create a Connectivity Map, cm WSUsernameTokenSvc, as shown below.



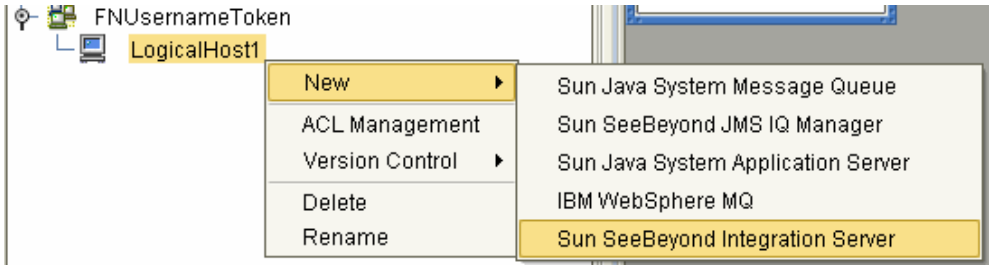
This concludes development of the logical service implementation. The environment, for which to build and to which to deploy this solution, will be created and configured next.

Create Java CAPS Environment

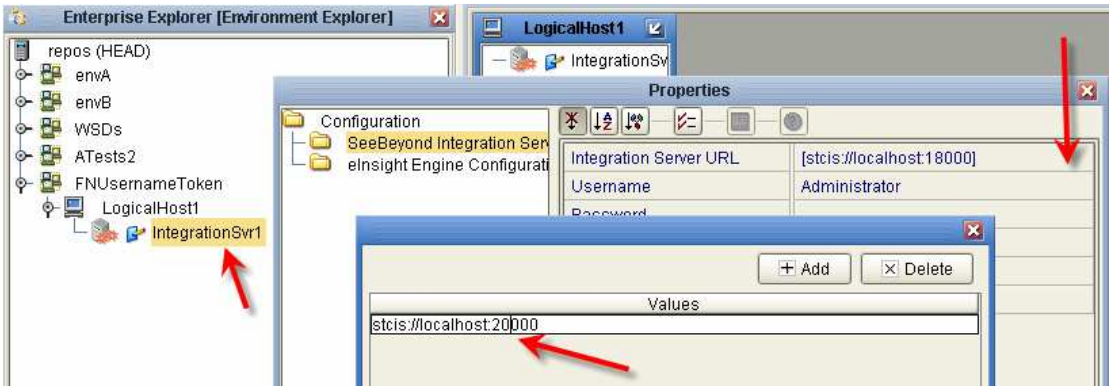
In Java CAPS 5.1.3 Enterprise Applications are deployed to Integration Servers/Application Servers, which at design time are subordinate to ‘logical hosts’ in environments. We will create a design-time Java CAPS Environment, related to a runtime Java CAPS domain, creation of which will not be discussed here. It is assumed in this document that the Sun SeeBeyond Integration Server will be the runtime target.



Let's name this new environment FNUsernameToken and, subordinate to it, let's create a Logical Host and a Sun SeeBeyond Integration Server.



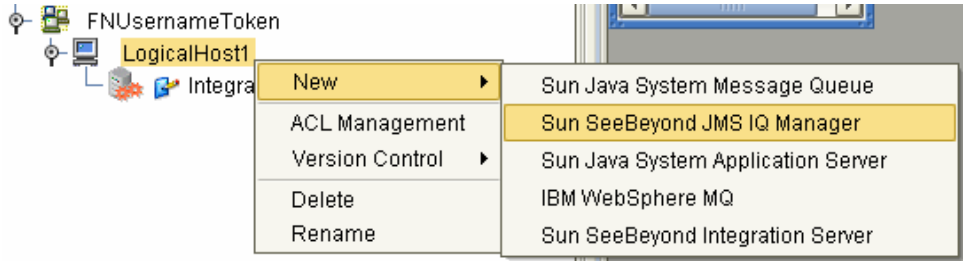
The runtime domain uses port 20000 as the administration port. To allow us to deploy enterprise applications directly from eDesigner let's configure the Integration Server to point to the runtime domain at the correct host and port, and populate the Administrator password property.



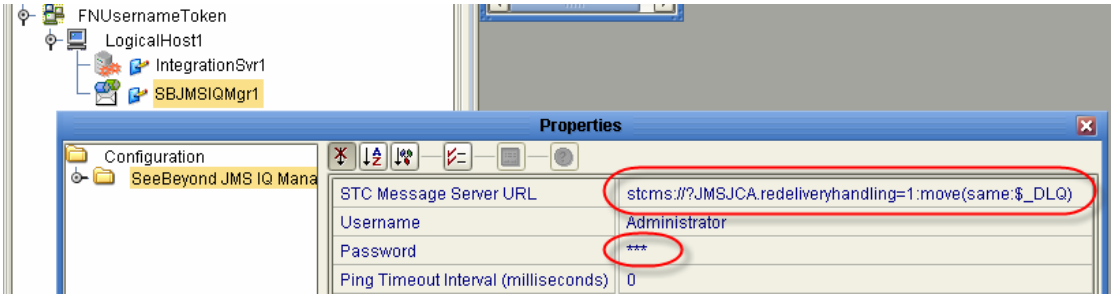
By the end of this document we will have built a service implementation and a service consumer. Each could be deployed to a different environment and both could be deployed to a single environment. The Integration Server in the logical host can be shared by the service implementation and the service consumer. In Java CAPS 5.1 service implementation requires an external system container separate from that required for the service consumer. Service implementation, at build time, can populate a UDDI Registry. To invoke a service consumer we will use an Enterprise Manager to place a trigger message in a JMS Queue. To do this we need a JMS Message Server in the environment. This message server will be used only by the service consumer. In summary, the service implementation will use the Integration Server, the UDDI Registry and the SOAP/HTTP External System in Server mode, whereas the service consumer will use the Integration Server, the JMS Message Queue (IQ Manager) and the SOAP/HTTP External System in Client mode. You

could create separate environments for each. In this exercise we will create a single environment for both.

Let's add the Sun SeeBeyond IQ Manager, the JMS Message Server through which the service consumer will be triggered.



To ensure that messages are not continuously rolled back and re-delivered in case of trouble, let's configure the STC Message Server URL with appropriate redelivery handling parameters and populate the Administrator password.

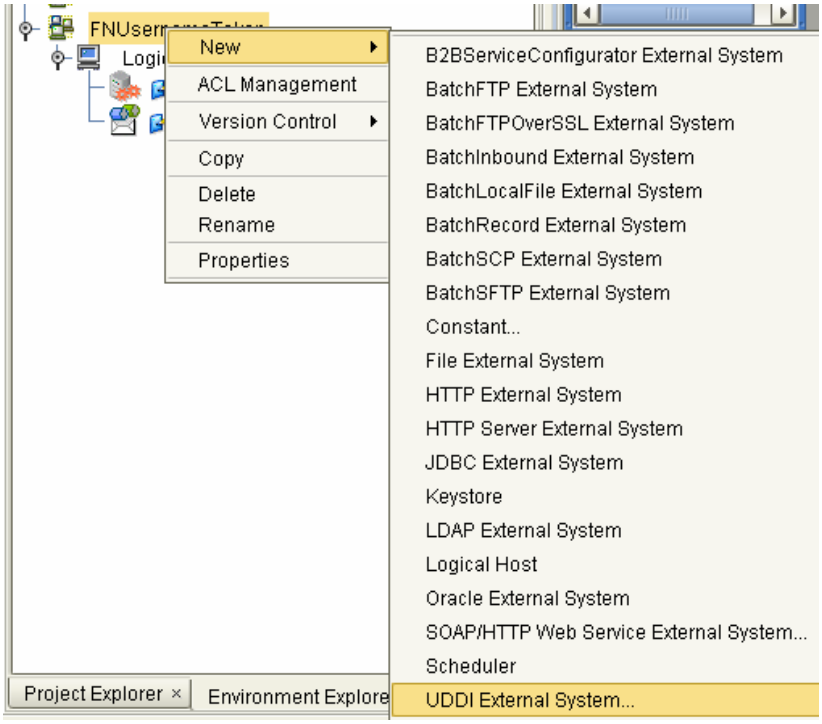


The STC Message Server URL with redelivery handling parameters above is:

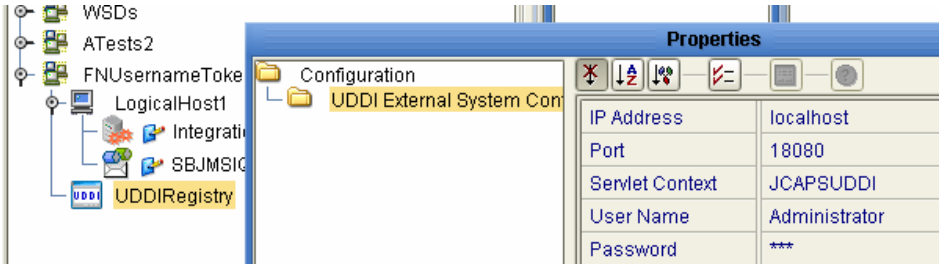
```
stcms://?JMSJCA.redeliveryhandling=1:move(same:$_DLQ)
```

This translates to: attempt to deliver ***1*** time. On failure ***move*** the message to the ***same*** kind of JMS Destination type (Queue or Topic) whose name is to be the name of the original destination ***$*** with the literal ***_DLQ*** appended.

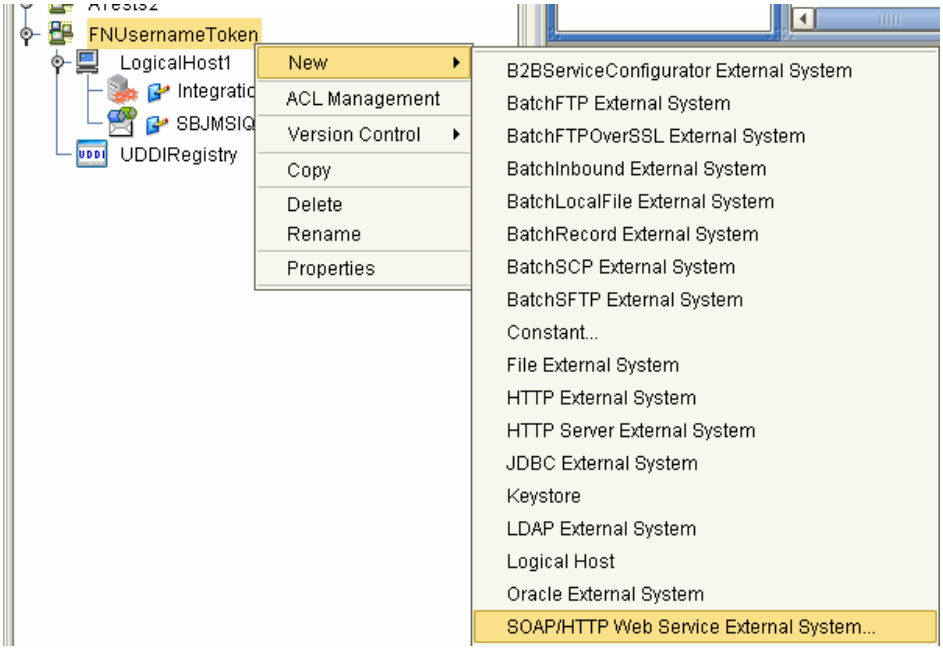
Let's add the UDDI Registry configuration information so that, at build time, the UDDI Registry can be populated with service implementation information.

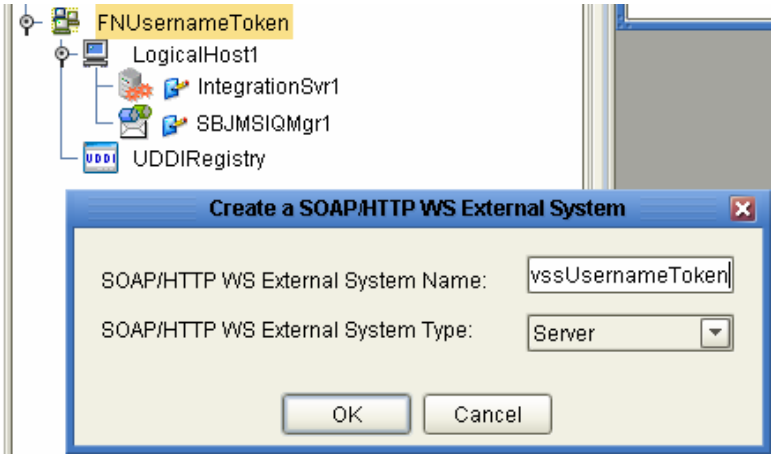


Use configuration property values appropriate to your installation.

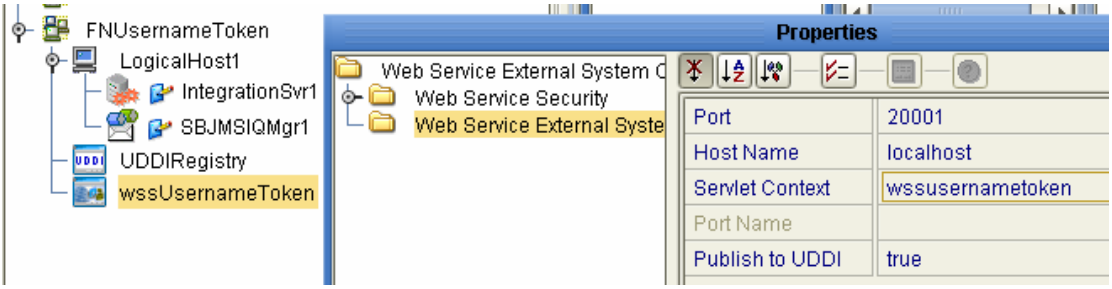


Let’s create a SOAP/HTTP External System container for the service implementation, in Server mode.



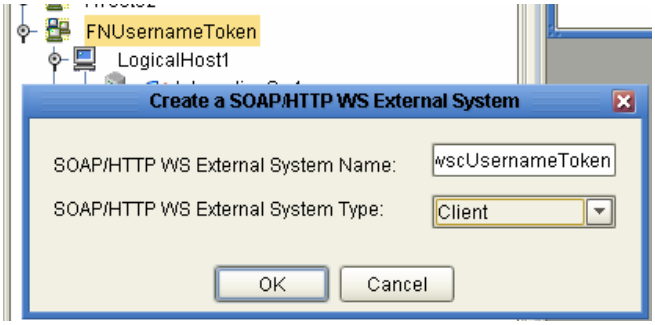


Configure this container's host and port as appropriate for your installation, making sure to populate the servlet context with the literal "wssusernametoken".



Note that the port is the port of the Web Server Container, 18001 by default, not the administration port, 18000 by default.

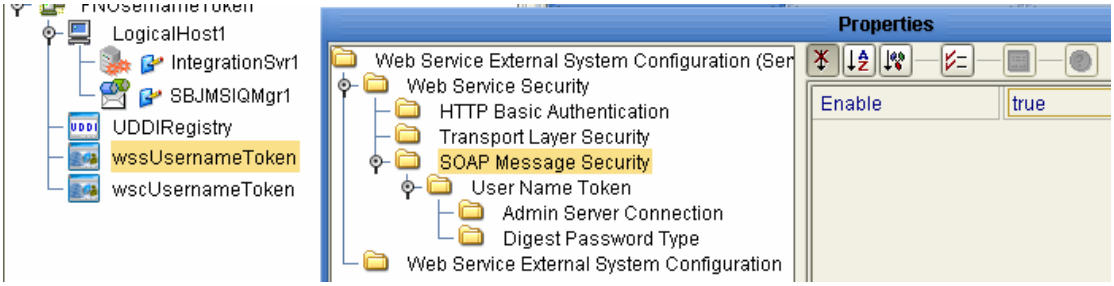
Let's now create the SOAP/HTTP External System container, wscUsernameToken, for the service consumer, in Client mode.



Since the servlet context of the service consumer is different from that of the service implementation, and we don't yet know what that context will be, we will configure this container later.

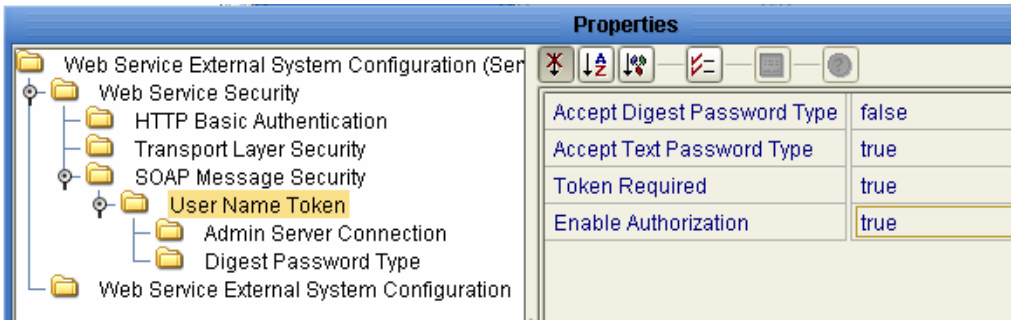
Configure SOAP/HTTP Web Service External Systems

So far we configured the SOAP/HTTP external System container for the service implementation to provide the basic, unrestricted service. To require service consumers to provide Username Token for authentication we need to enable the container's Web Services Security -> SOAP Message Security



and configure User Name Token and Admin Server Connection properties.

Note that, once the SOAP Message Security property is set to true a SOAP Request that does not contain any WS-Security headers will generate a SOAP Fault.

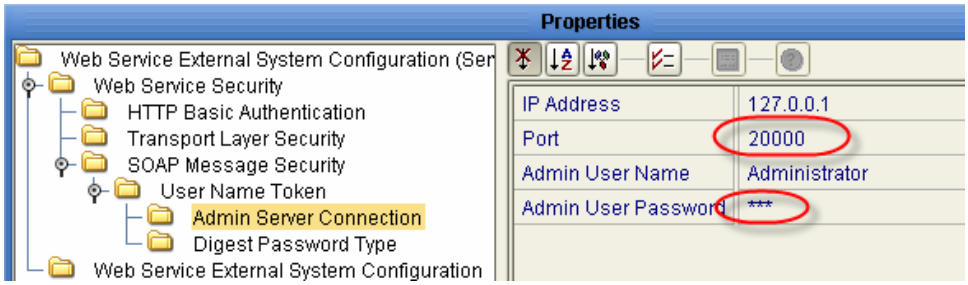


Here we disable digest password type, enable Token Required and enable Authorization. To facilitate authorisation we need to configure the Admin Server Connection properties as appropriate for the Integration Server to which this service will be deployed.

Note that if "Enable Authorization" is set as "false" the service implementation will not be protected as well as it could. Any username and password credentials that

correspond to a valid Integration Server user will be acceptable. To ensure that only specific users have access to the service implementation “Enable Authorization” must be set to “true” and web service access, of which later, has to be configured.

Note that if Token Required is set to “false” the service will not be protected and supplying a WS-Security Username Token SOAP header will result in a SOAP fault being returned to the service consumer.



The service is now configured to require the WS-Security SOAP Header with the WS-Security Username Token stanza and to validate credentials provided through these means against the user directory configured for the Integration Server.

Build Service Implementation

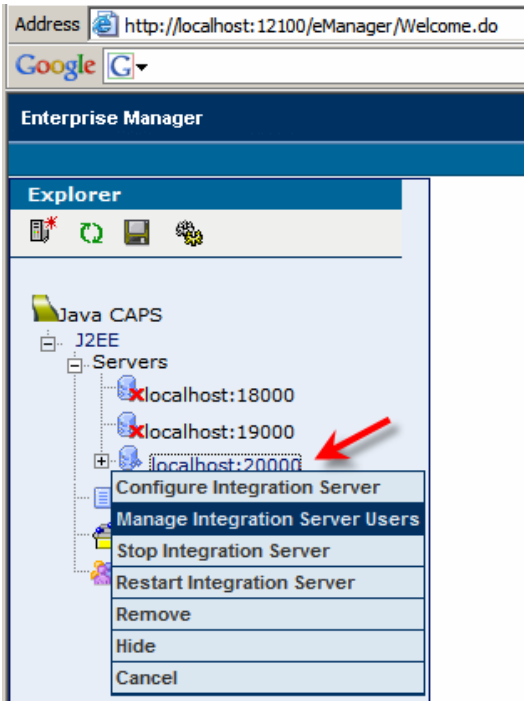
Now that the Java CAPS Environment with the appropriate components is available we can build and deploy the service.

Let’s create a deployment profile, dpWSSUsernameToken, map components to the appropriate containers in the environment, build and deploy.

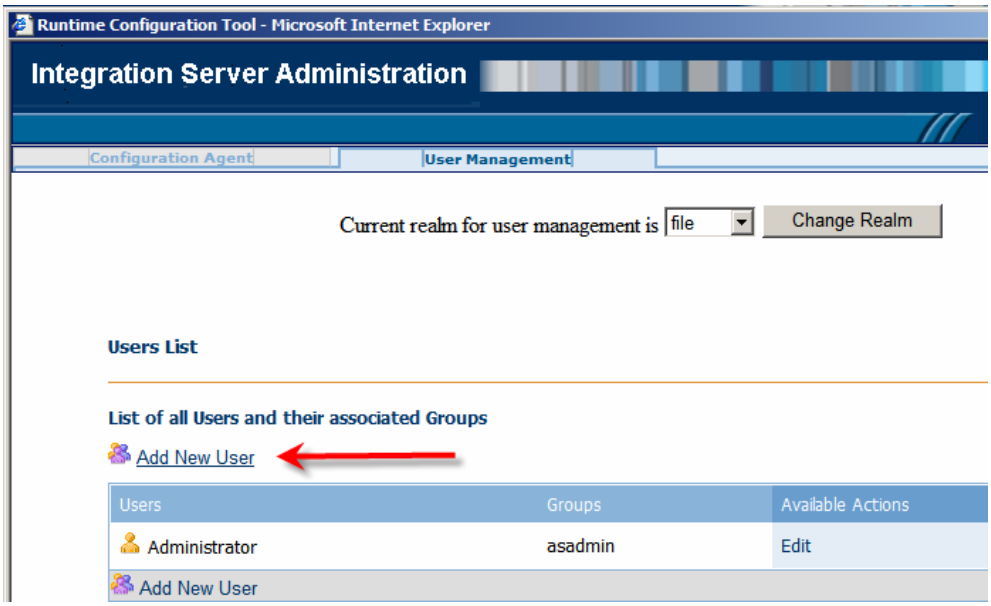
Create Integration Server Users

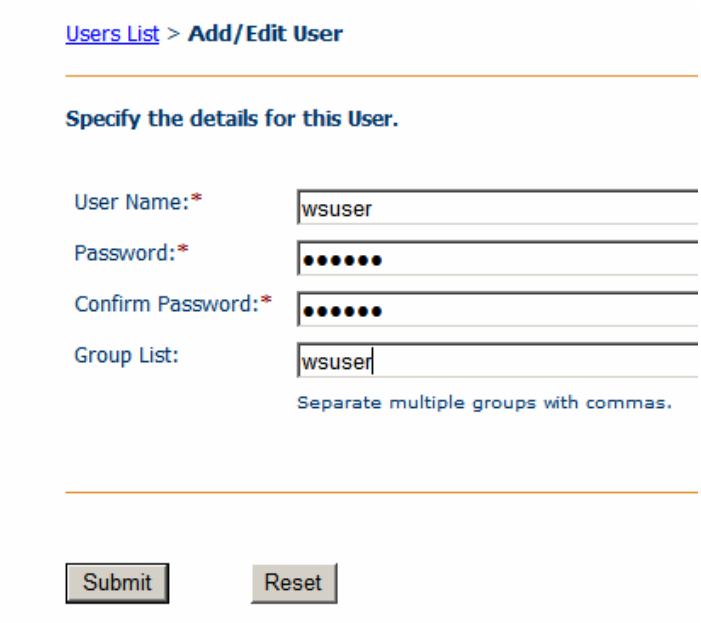
Only Integration Server users can invoke web services that use WS-Security 1.0 Username Token Profile. By default, the only Integration Server User defined during Java CAPS installation is the user Administrator. Whilst it is possible to require web service clients to use the user Administrator for authentication it necessitates disclosure of the password for the administrative user, potentially outside the enterprise – not a good idea at all.

To support more granular access, from the authentication perspective, to web services exposed through Java CAPS, it is necessary to create one or more Integration Server users.



Add a user, wsuser, with the password of wsuser and a group list of wsuser. These are just arbitrary strings.



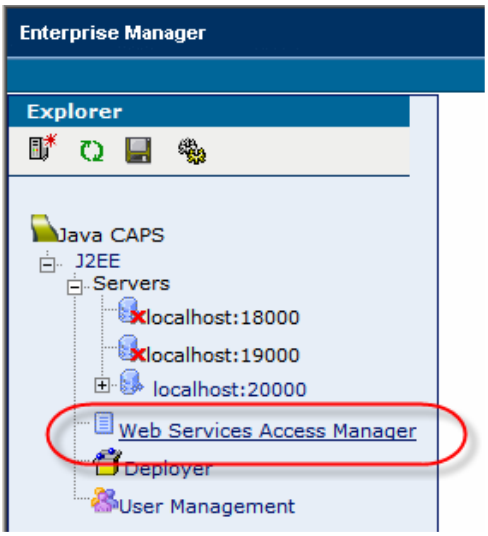


Whilst Group List is shown as optional, not providing a value will prevent creation of a user entry in 5.1.3.

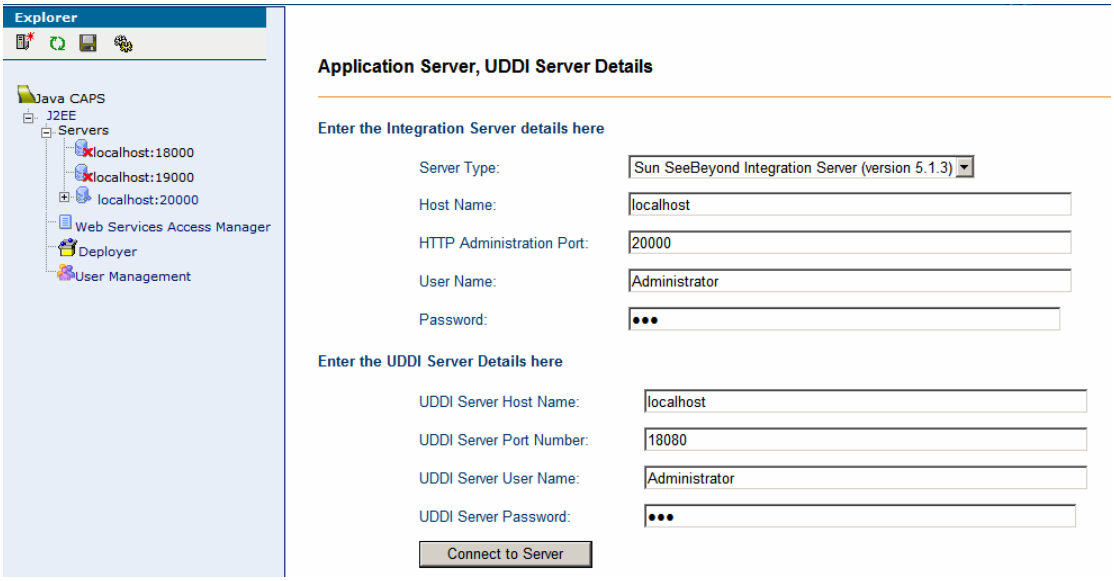
Configure web service access

As noted before, if SOAP/HTTP External System container's Web Services Security -> User Name Token -> Enable Authorization property is set to "true", web service access will have to be configured to designate that subset of Integration Server users that will be able to access the service.

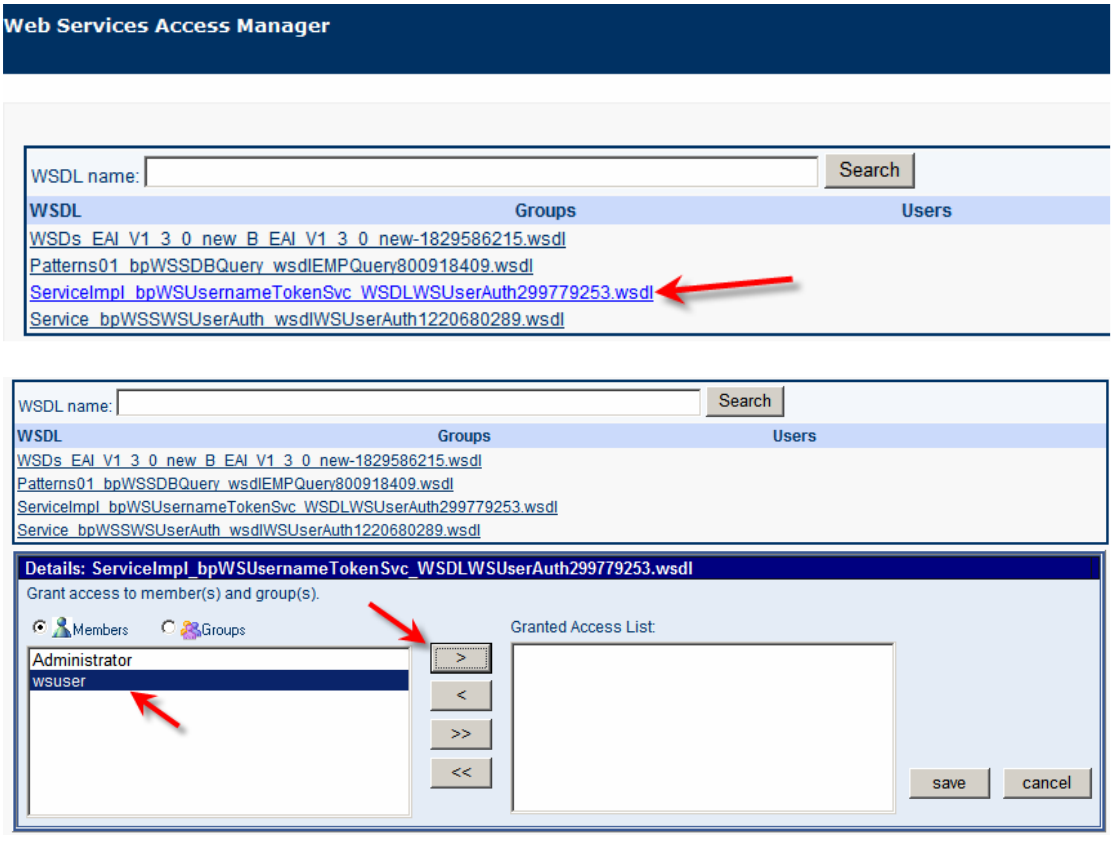
Web service access is managed through the Enterprise Manager.

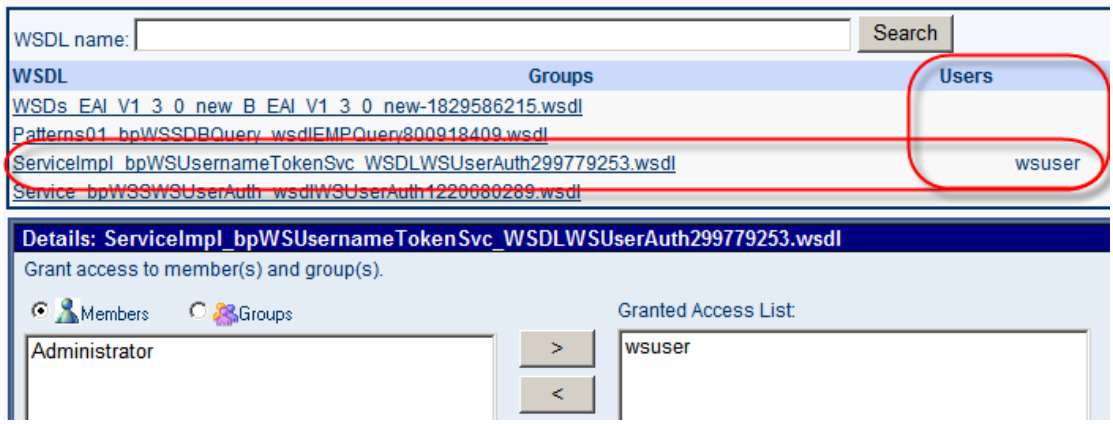


To manage access to services deployed to a particular Integration Server both the Integration Server and the UDDI Registry URLs and credentials need be provided.



Choosing the web service, from the list of web services published to the UDDI Registry, will allow selection of Integration Server users that can be granted access to the service.



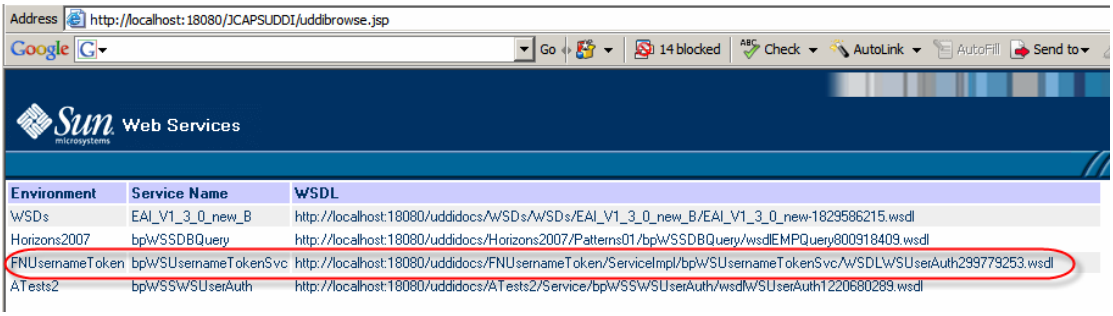


Additional users can be granted access as required. Credentials of Integration Server users who are not granted access will be rejected.

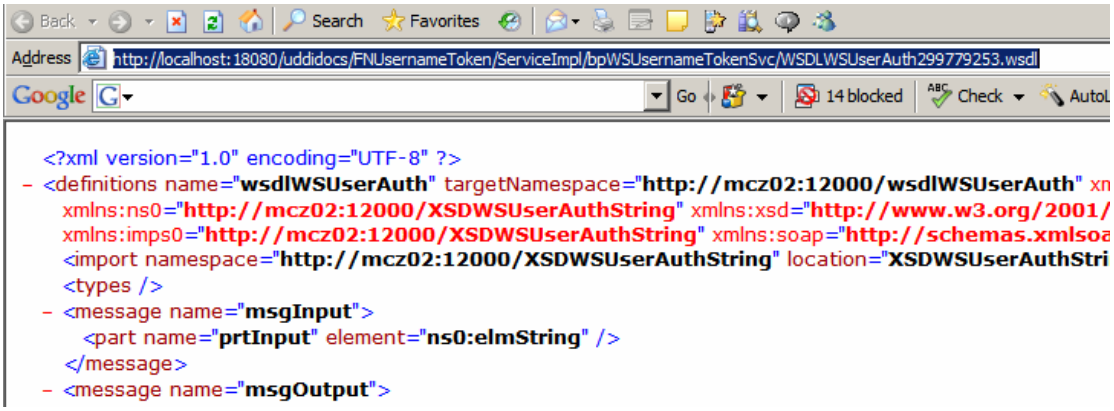
Test service implementation with soap UI

The fastest way to test service implementation, including WS-Security Username Token Profile processing, is to use soap UI. soap UI can be downloaded from http://www.soapui.org/. For this exercise soap UI 1.7.6 has been used.

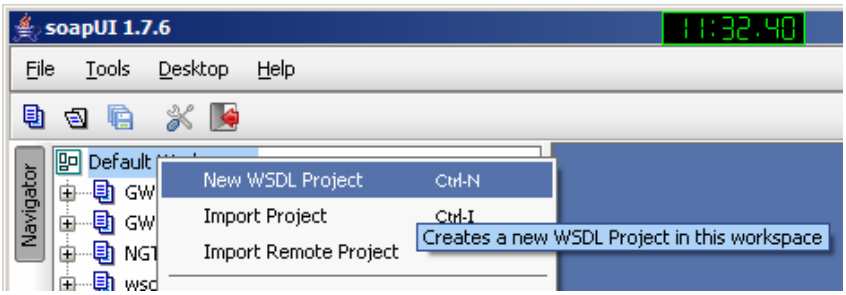
Use Java CAPS UDDI Registry UI to determine the URL of the WSDL document corresponding to the service implementation.



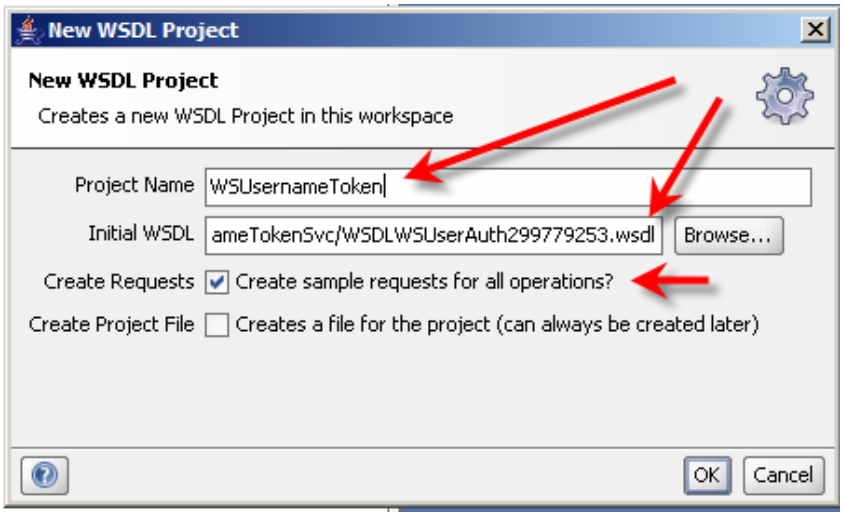
Click on the link, select and copy the URL.



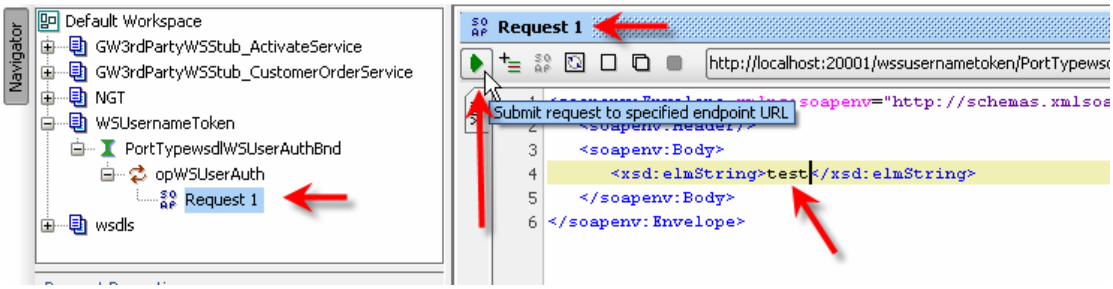
Start soap UI and create a new project.



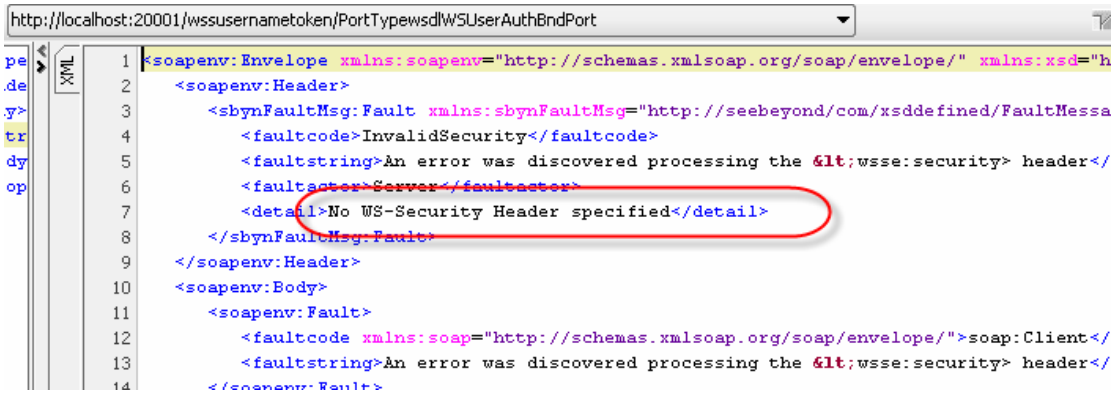
Name the project WSUsernameToken and paste the WSDL URL into the text box provided.



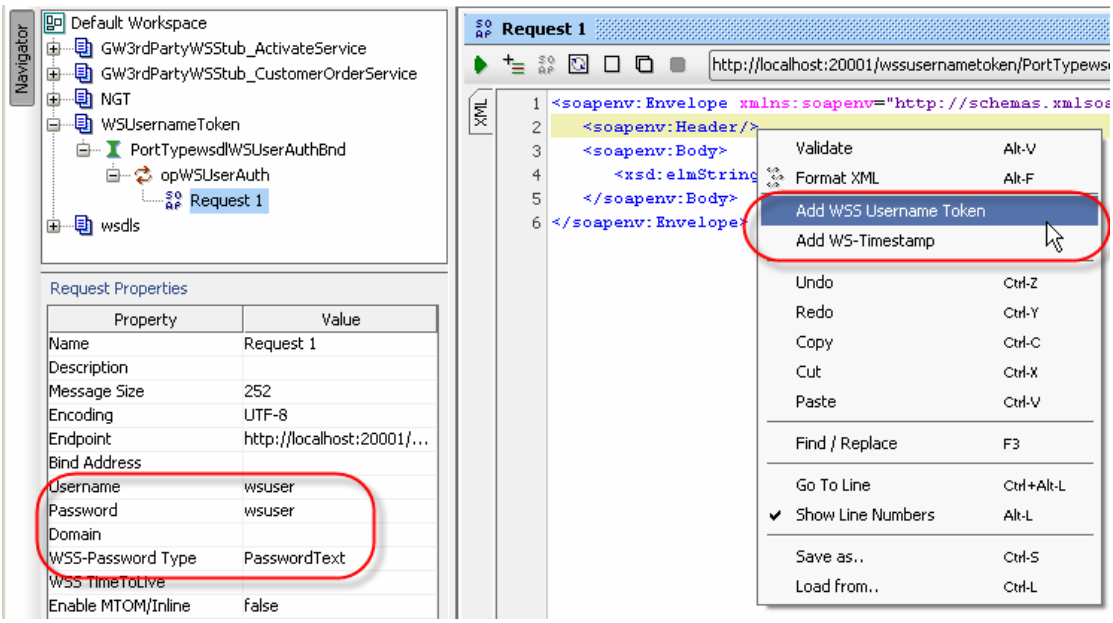
Open the request, populate the content of the elmString node and submit the request.

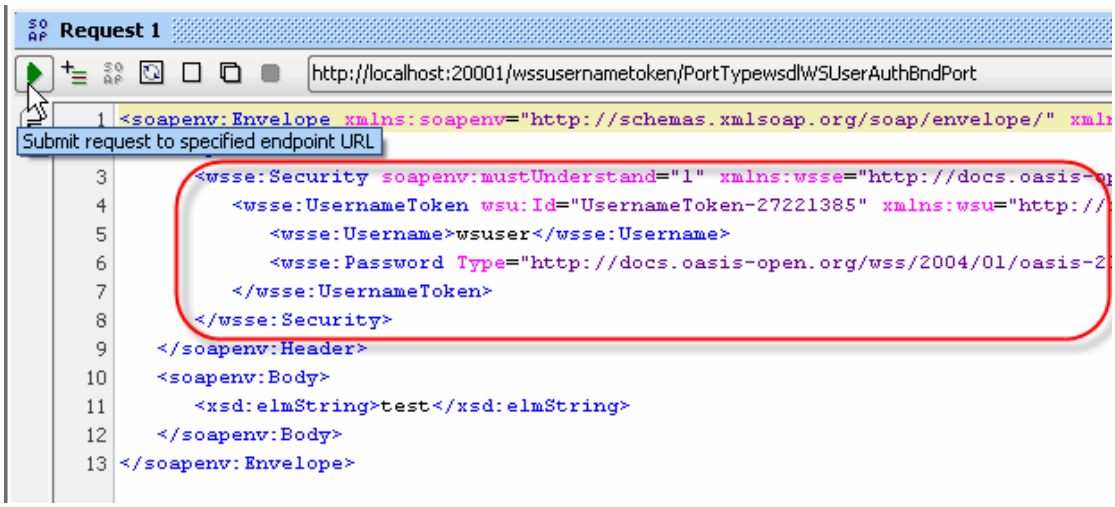


Since the request does not contain WS-Security Username Token header the request will be rejected.



Let’s configure soap UI to provide the WS-Security Username Token header and re-submit the request.

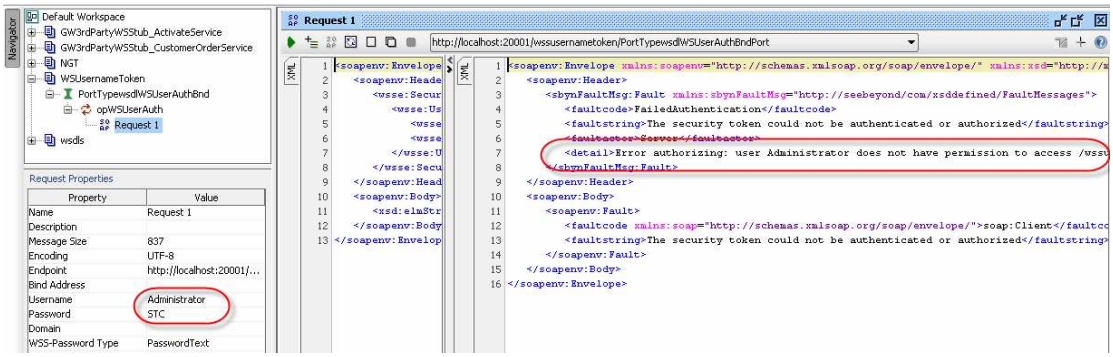




The service executes and returns a result.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:enc="ht
   <env:Body>
      <elmString
        xmlns="http://mcz02:12000/XSDWSUserAuthString"
        xmlns:tns="http://mcz02:12000/XSDWSUserAuthString">**test**</elmString>
   </env:Body>
</env:Envelope>
```

To test that indeed user credentials are validated, let's use a different username and password, for an Integration Server user that has not been granted access to the service.



Let's now send a request credentials that correspond to no user of which the Integration Server is aware.

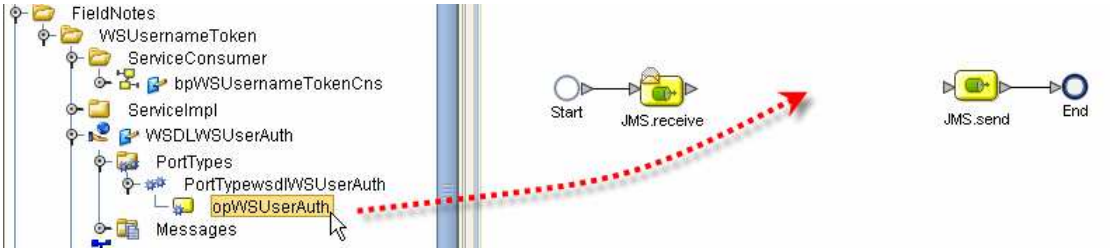
Develop client implementation

soap UI tests, described above, demonstrated that the service is configured to require the WS-Security Username Token header, that is validates credentials to ensure they correspond to an Integration Server user and that is permits access to the service implementation only to the users who have been explicitly granted acces to that service implementation.

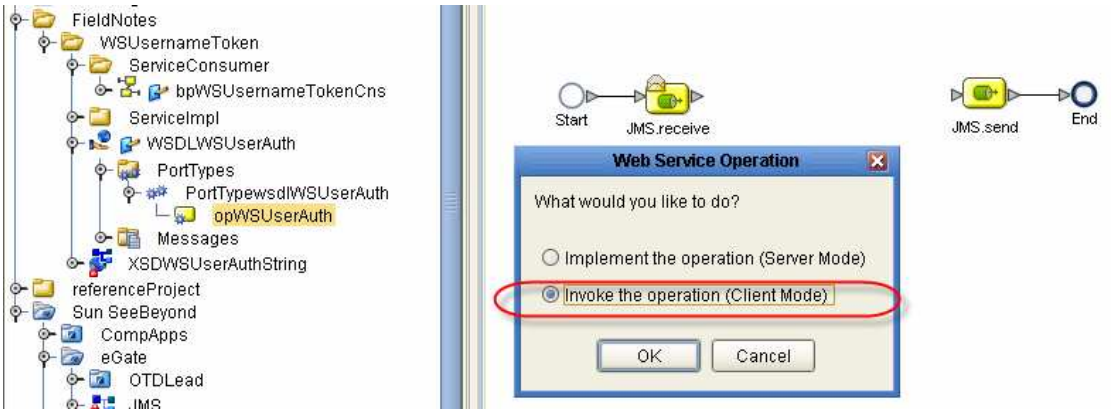
Let's now develop a Java CAPS service consumer that will invoke the service.

Let's create a subproject, ServiceConsumer, and an eInsight business process, bpWSUsernameTokenCns, that will invoke the service. This process will be triggered by a JMS Receive activity, will map JMS Text Message to the elmString service input message and will deposit service result string, as a Text Message, in a JMS Queue.

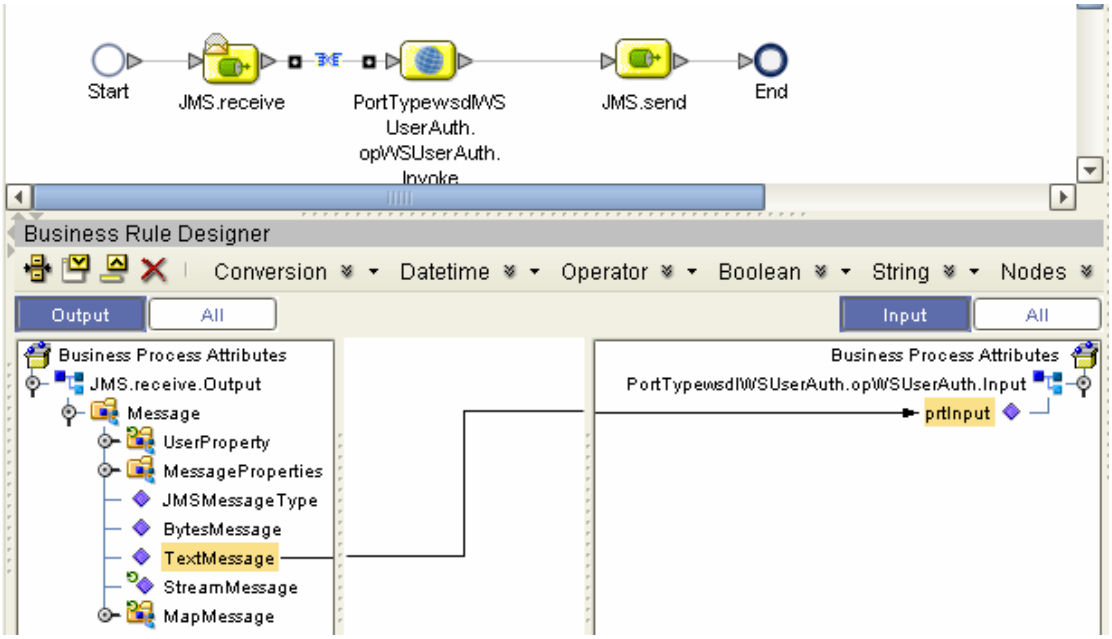
Add JMS Receive and Send, and drag the service operation onto the canvas.



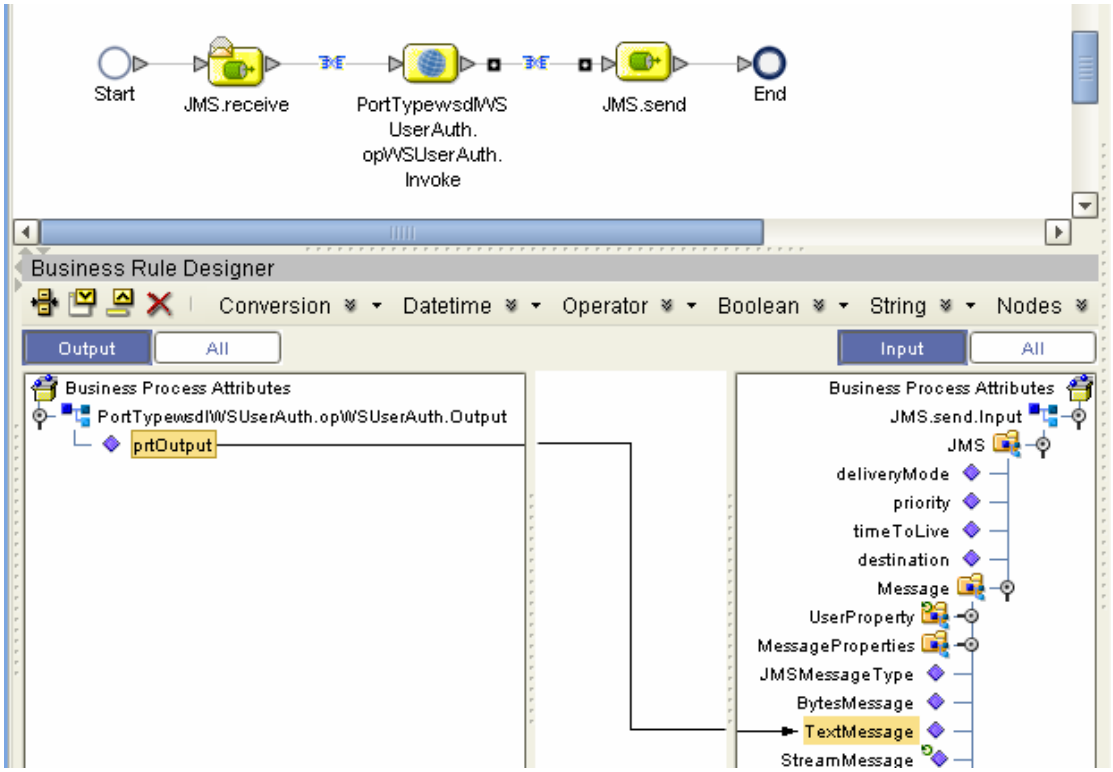
Choose “Invoke the operation …”.



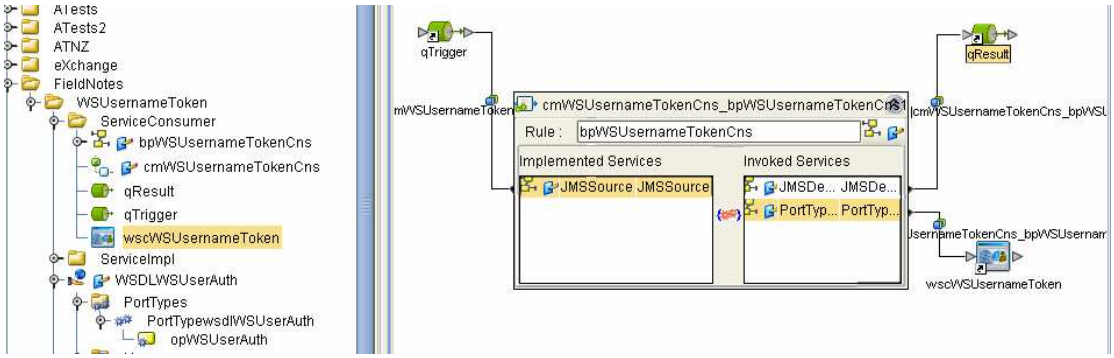
Map JMS Text Message to service input.



Map service output to JMS Text Message.



Create a connectivity map, cmWSUsernameTokenCns, as shown.

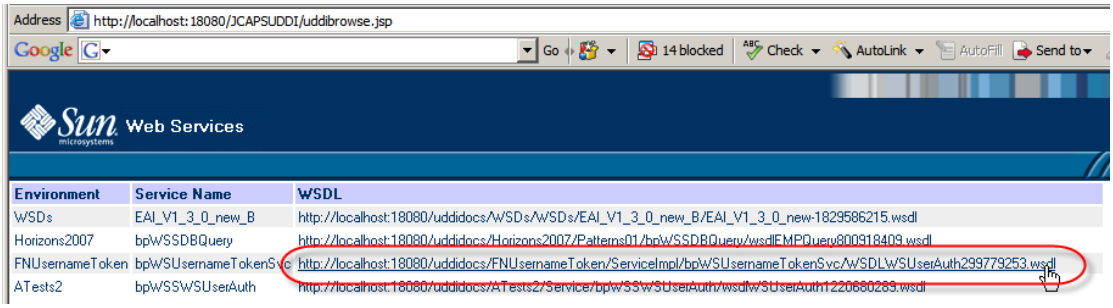


This project will be built and deployed later.

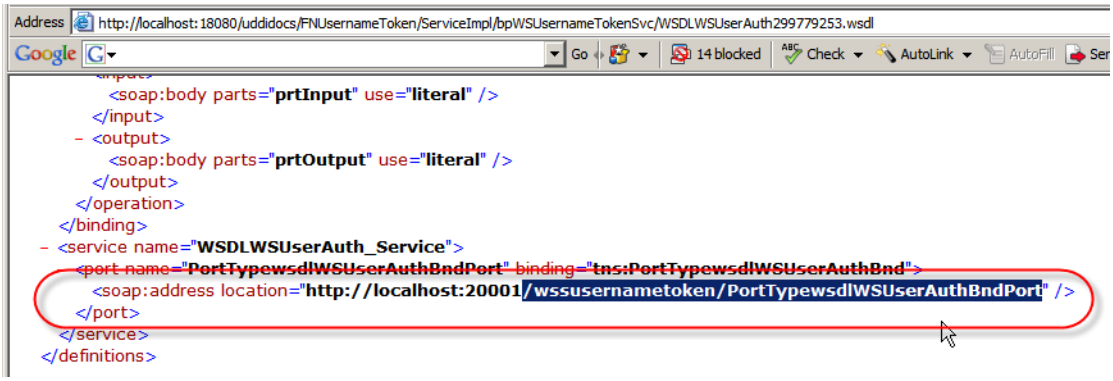
Configure SOAP/HTTP Client External System

The SOAP/HTTP External System for the service consumer was left unconfigured. Let's now configure it.

Let's determine the servlet context of the service which to invoke.



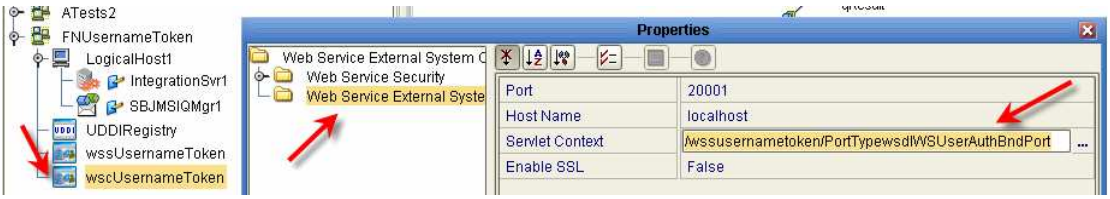
Let's view the WSDL and locate the
`<soap:address location=" http://localhost:20001/wssusernametoken/PortTypewsdlWSUserAuthBndPort " />`
attribute.



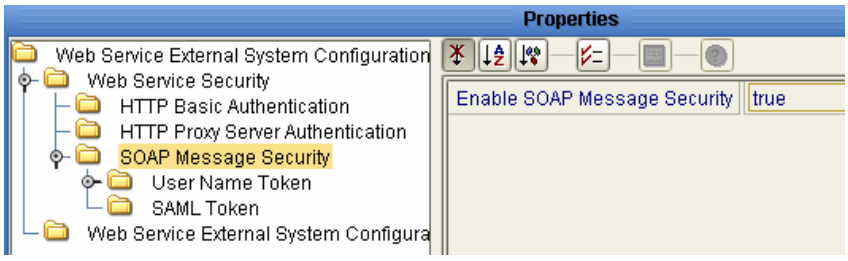
The servlet context is the entire string following the port number in the URL:

`/wssusernametoken/PortTypewsdlWSUserAuthBndPort`

Let's copy this text and paste it into the Servlet Context property of the SOAP/HTTP External System container of the service consumer in the Java CAPS Environment.

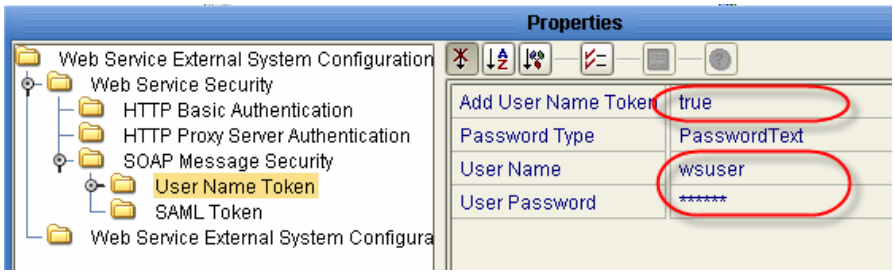


Configure the Port and the Host Name as appropriate to your runtime environment.



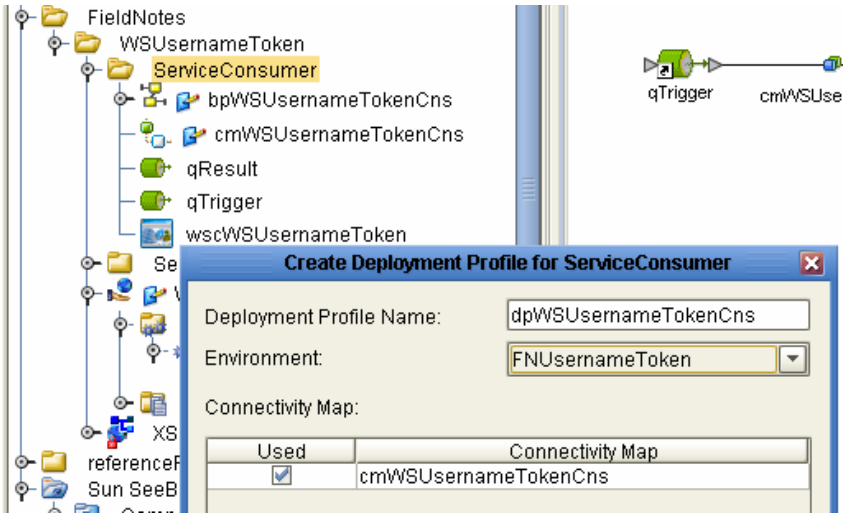
Set Web Services Security -> SOAP Message Security -> Enable SOAP Message Security property to "true".

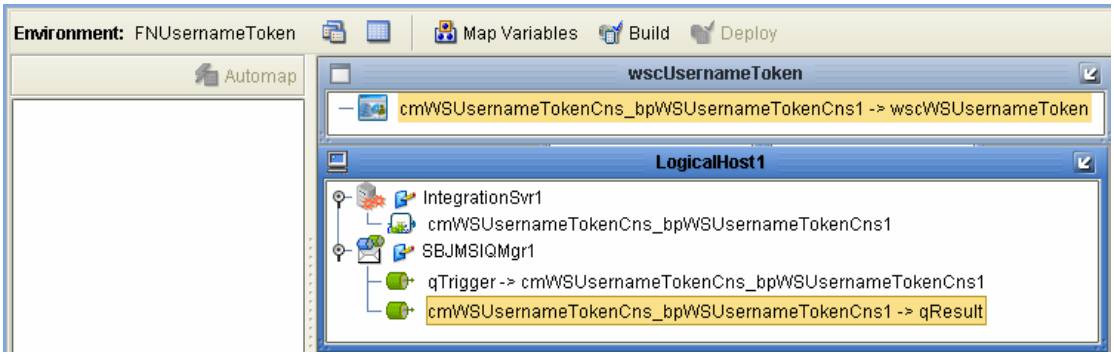
Configure SOAP Message Security -> Usner Name Token properties “Add User Name Token”, User Name and User Password, to enable sending WS-Security Username Token Profile header and provide credentials with access to the service.



Deploy Service Consumer

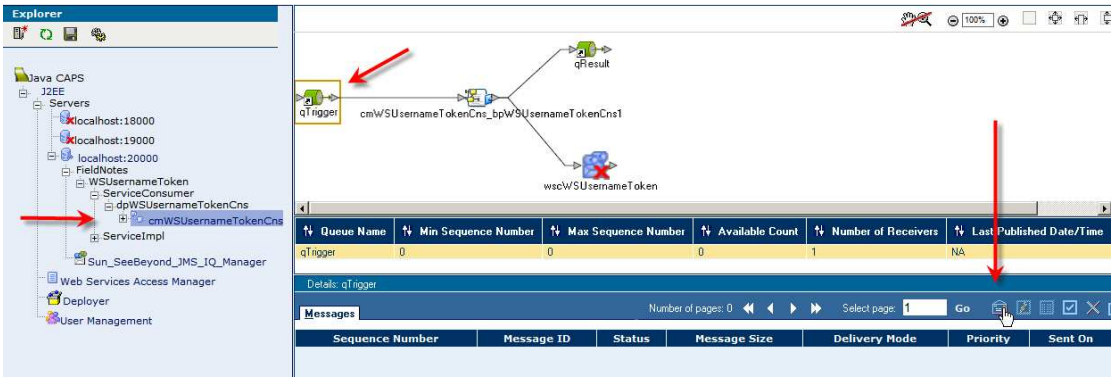
Create a deployment profile, map components, build and deploy the service consumer.



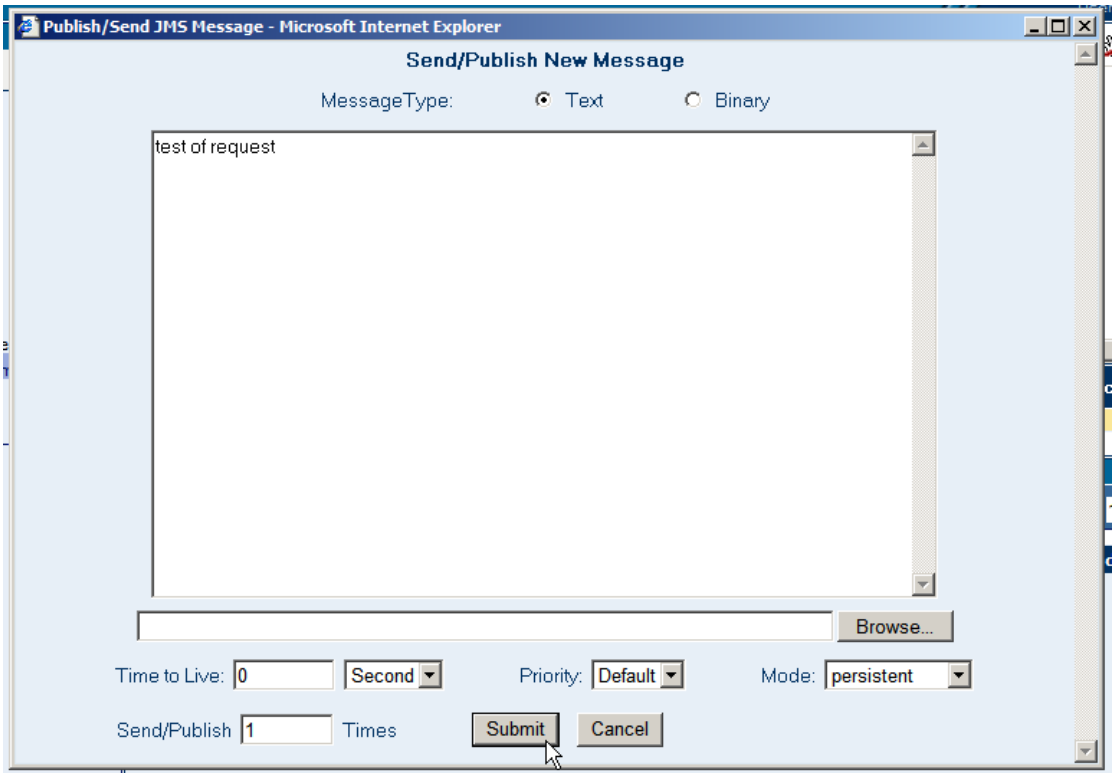


Test End-to-End Solution

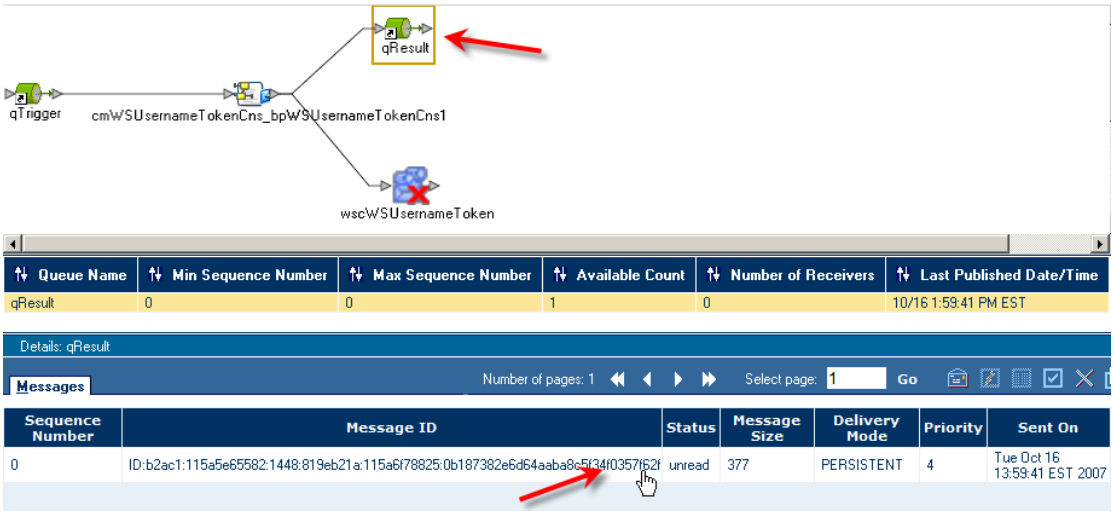
Use the Enterprise Manager to open the service consumer implementation connectivity map graphic. Click on the JMS Queue qTriger and click on the Send/Publish toolbar tool.



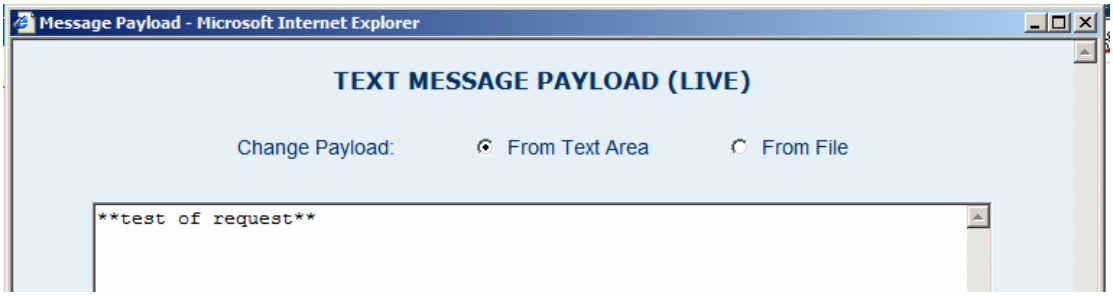
In the resulting dialogue box enter some text, which is to be the content of the inpt message, and click “Submit”.



Click on the JMS Queue qResult and double-click on the JMS Message in the lower pane to inspect message content.



The message contains results of web service invocation.



Because we configured the service implementation to require a username token and to validate it and verify access to the service by the user we know that the service consumer provided correct credentials. If it did not the invocation would have failed.

Conclusion

Developing a service implementation and service consumer for a web service solution that requires the use of WS-Security Username Token Profile takes much less time then it took to write this document. The SOAP/HTTP Web Service External System container is where Username Token Profile parameters are set. For service consumer this is all that is required to provide support. For service implementation appropriate Integration Server users must also be created and, if required, access to the specific web service must be granted to these users. Whether implemented as an eInsight business process or as a Java Collaboration, configuration of the service implementation to support WS-Security Username Token Profile requires the same steps.