

Iowa Immunization Registry Information System (IRIS)

Web Services Data Exchange Setup

Version 1.0

Last Updated: February 11, 2013



Table of Contents

- SSL Certificate Creation..... 3**
- Generating a Key and Certificate Signing Request (CSR) 3**
 - Generating a Private Key and CSR using OpenSSL 3
 - Step 1: Generate Private Key 3
 - Step 2: Generate the CSR..... 3
 - Step 3: Send CSR and request for Username and Password to HP and the Iowa Department of Public Health..... 5
 - Step 4: Backup the private key 5
 - Step 5: Receiving Your Signed Certificate 5
- Generating a Private Key and CSR using Microsoft Windows 7**
 - Step 1: Creating the Private Key and CSR 7
- SSL Trust Stores in a Web Services Context..... 8**
 - Keystores 8
 - Truststores..... 8
 - IRIS Web Services Client Operations..... 8
- Get information about Secure Sockets Layer (SSL) certificates..... 9**
- Introduction to SOAP Web Services in IRIS 11**
- Getting the WSDL 12**
 - Obtaining the WSDL: Extract the version from this document 12
- Appendix A: WSDL File Contents 13**
- Appendix B: URL for OpenSSL..... 18**

SSL Certificate Creation

The Secure Sockets Layer (SSL) is a commonly-used protocol for managing the security of a message transmission on the Internet. SSL client and server certificates are used as an added security feature for transmitting data for IRIS Web services transactions. IRIS requires both the client and server install certificates generated by the Hewlett Packard Enterprise Services (HP) Immunization Services personnel. To accomplish this, you must first create a private key for each machine that will be accessing the IRIS Web services machine. This private key is then used to create a Certificate Signing Request (CSR) which will be sent to HP. HP will create the SSL certificate which will be returned to you for installation on your client machine ("client" in this instance will most likely be the server that communicates with the IRIS Web services servers).

Generating a Key and Certificate Signing Request (CSR)

To generate a CSR, a key pair must be created for the server. These two items are a digital certificate key pair and cannot be separated. If the public/private key file is lost or changed before the SSL certificate is installed, the SSL certificate will need to be re-issued. The private key, CSR, and certificate must all match in order for the installation to be successful. The following sequence of commands will generate a 2048 bit key using the OpenSSL software. Below are instructions for creating the CSR in a Windows environment. It is recommended to use the domain name or IP address that will be used for the certificate as the prefix of the filenames. Also make sure that any existing keys and CSRs are NOT overwritten.

Generating a Private Key and CSR using OpenSSL

Step 1: Generate Private Key

Type the following command at the prompt:

```
openssl genrsa -out my.server.com.key 2048
```

This command generates a 2048 bit RSA private key and stores it in the file, *my.server.com*.key

Note: For all SSL certificates, the CSR key bit length must be 2048. The text in italic bold (i.e. *my.server.com*) is only an example. Replace it with a name that is meaningful to you. It does not have to be a website but MUST have the ".key" extension.

Step 2: Generate the CSR

Type the following command at the prompt:

```
Openssl.exe req -new -key my.server.com.key -out my.server.com.csr
```

This command will prompt for the following attributes of the certificate:

Field	Required / Optional	Description
Country Name	R	Use the two-letter code without punctuation for country. Example: US or CA
State or Province	R	Spell out the state completely; do not abbreviate the state or province name. Example: Iowa
Locality or City	R	The Locality field is the city or town name; do not abbreviate. Example: Mount Pleasant , not Mt. Pleasant

Company	R	If the company or department has an &, @, or any other symbol using the shift key in its name, the symbol must be spelled out or omitted. Example: XY & Z Corporation would be XYZ Corporation or XY and Z Corporation.
Organizational Unit	O	Can be used to help identify certificates registered to an organization. The Organizational Unit (OU) field is the name of the department or organization unit making the request. To skip the OU field, press Enter on the keyboard.
Common Name	R	The Common Name is the Host + Domain Name. It looks like "my.server.com".
Email Address	NA	Do not enter anything in this field.
Challenge Password	NA	Do not enter anything in this field.
Optional Company Name	NA	Do not enter anything in this field.

The certificate is used to secure the transaction between provider EHR and IRIS. Each certificate is required to have a unique common name. Combining your host and domain names will ensure your common name is unique.

A public/private key pair has now been created.

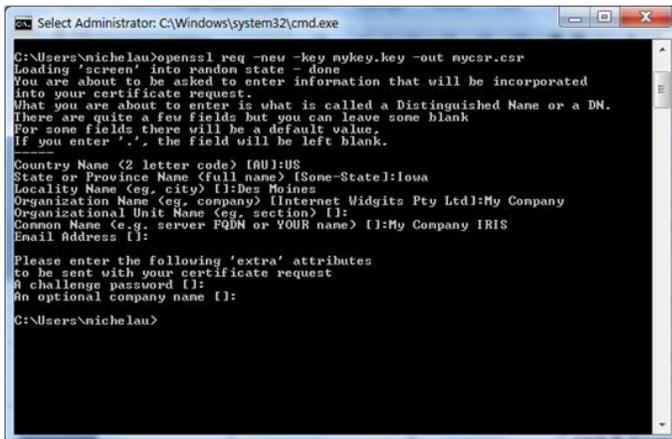
Private Key:

- File Name example: my.server.com.key
- Stored locally on the server machine
- Used for decryption (IMPORTANT – Maintain in a safe place).

Public Key

- File Name example: my.server.com.csr
- In the form of a Certificate Signing Request (CSR)
- Used for certificate enrollment

Screenshot of the screen showing the attributes of the certificate:



Step 3: Send CSR and request for Username and Password to HP and the Iowa Department of Public Health

The CSR is an ASCII text file that can be attached to an email and should be sent to David Thrall at david.thrall@hp.com and copy Kim Tichy at Kimberly.Tichy@idph.iowa.gov.

The username and password will be used later when setting up the WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol) protocols.

Step 4: Backup the private key

It is recommended to back-up the `.key` file. An acceptable option is to create a copy of this file onto a flash drive or other removable media. While backing up the private key is not required, having one will be helpful in the instance of server failure.

Step 5: Receiving Your Signed Certificate

Once HP is done processing your CSR, you will receive the signed certificates as an email attachment. They must be unzipped and placed in the same folder as your request. These files must be installed in the Trusted Store of the computer for which it was generated. The files are:

1. The CA certificate, `ca.crt`. This is common to all Immunization Information System (IIS) users and establishes HP's server as a valid Certificate Authority, which tells your server to trust Certificates issued by HP.
2. The specific server certificate, such as `csr-site-request.crt`. This is unique to the server it was generated from.

The instructions for this will vary depending on your environment. Two common trusted stores are Public-Key Cryptography Standards #12 (PKCS#12 or PFX) and Java Key Store (JKS). A good source of reference for this information is Google (search for: importing trusted root certificates).

Join your private key with the signed certificate and certificate authority files e-mailed by HP, so the browser and OS can use it.

Here is an example of creating a `.pfx` file using `openssl`.

```
openssl pkcs12 -export -out www.example.com.pfx -inkey www.example.com.key -  
in www.example.com.crt -certfile cacert.crt
```

Here are what the example file names represent:

[www.example.com.pfx](#) = this will be the output file – which you'll install into Windows 7 so IE can use it

[www.example.com.key](#) = this is the key that was generated by step 1

[www.example.com.crt](#) = this is the signed certificate provided in response to the CSR

[cacert.crt](#) = this is the CA (Certificate Authority) file which was provided. This is needed by `openssl` to verify the first file was signed.

Refer to the following steps if you are using a `.pfx` file. Otherwise, [skip](#) this section.

1. Go the file where the "[www.example.com.pfx](#)" file has been created.
2. Right click on the file.
3. Select Install PFX.

4. Click on Next.



5. Click on Next.



6. No password is required. Ensure that all three check boxes are selected. Click on Next.



8. Click on Next.



9. Click on Finish.



Generating a Private Key and CSR using Microsoft Windows

Step 1: Creating the Private Key and CSR

1. Open the **Microsoft Management Console (MMC)**. On the Start menu, click Run, type MMC, and then click OK. MMC opens with an empty console.
2. Right-click the default Web site, click **New**, and then click **Site**. Create a new site and give it a temporary name.
3. Right-click the new site, click **Properties**, click the **Directory Security** tab, and then click **Server certificate**.
4. Select **Create new certificate** and follow the wizard to create a new CSR. Use the information from the OpenSSL instructions above in Step 2, when filling out the request. When prompted, select **Prepare the request now but send it later**.
5. Use the CSR that you just created to request a new certificate from HP.
6. See the OpenSSL instructions Steps 3, 4, and 5, above, to finish the process.

SSL Trust Stores in a Web Services Context

Keystores

A keystore is a database of private keys and their associated X.509 certificate chains authenticating the corresponding public keys. A key is a piece of information that controls the operation of a cryptographic algorithm. For example, in encryption, a key specifies the particular transformation of plain text into ciphertext, or vice versa during decryption. Keys are used in digital signatures for authentication.

Truststores

The truststore contains the Certificate Authority (CA) certificates and the certificate(s) of the other party to which this entity intends to send encrypted (confidential) data. This file must contain the public key certificates of the CA as well as the client's public key certificate.

IRIS Web Services Client Operations

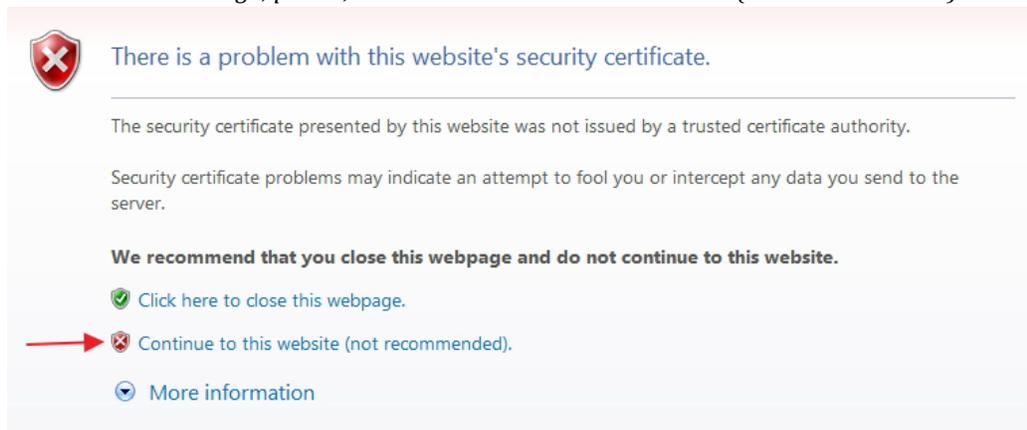
For all provider servers which will be accessing the IRIS Web Service, a Truststore needs to be established. The Truststore should contain the signed certificate received from HP plus the CA certificate(s) from the Web Services server.

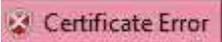
Disclaimer: The following screen shots were taken using Internet Explorer 7 and may differ depending upon the browser you are utilizing.

The CA certificate(s) can be acquired, by pointing your web browser at the IRIS Web Services site (e.g., using a WSDL request).



Notice: You may receive the following notification "There is a problem with this web site's security certificate." If you do receive this message, please, click the "Continue to this website (not recommended)." link.



Right click on the browser's lock icon  to display the certificate. If the lock icon  is not present, you may see . Click on the Certificate Error.

Get information about Secure Sockets Layer (SSL) certificates

When you connect to a website, Internet Explorer uses a secure connection that uses Secure Sockets Layer (SSL) technology to encrypt the transaction. The encryption is based on a certificate that provides Internet Explorer with the information it needs to communicate securely with the website. Certificates also identify the website and owner. You can view a certificate to validate the identity of a website before providing information.

To validate the identity of a website:

1. Open Internet Explorer
2. Go to the website you want to validate.
3. Click the Lock icon , which is located to the right of the Address bar.

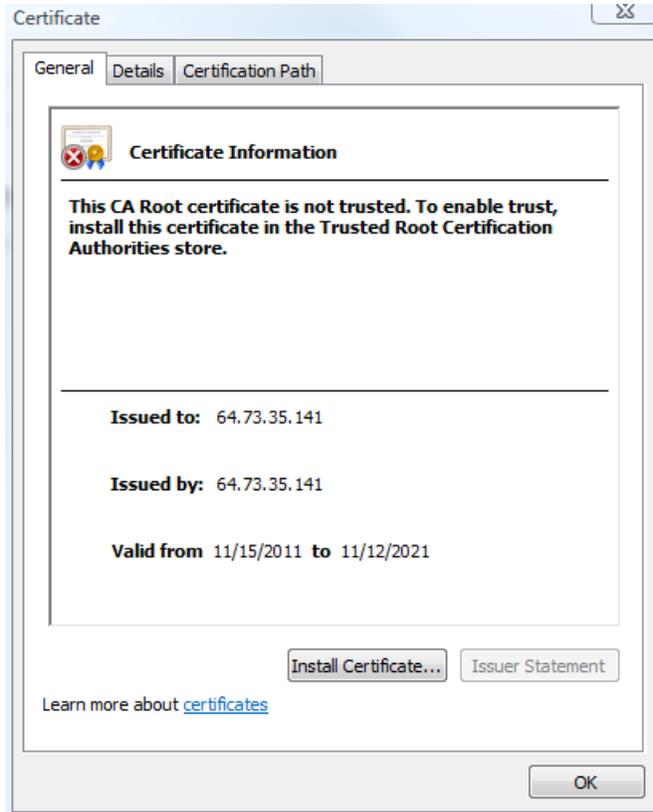
Basic certificate information (for example, the name and address of the website owner and information about who certified the site) will be displayed. To see additional information, click View Certificates.

Note:

If a lock icon does not appear in the Address bar in step 3 above, the connection is not secure. You should now see a notification “Untrusted Certificate”. Click on “View certificates”



You should now be presented with the Certificate dialogue box.



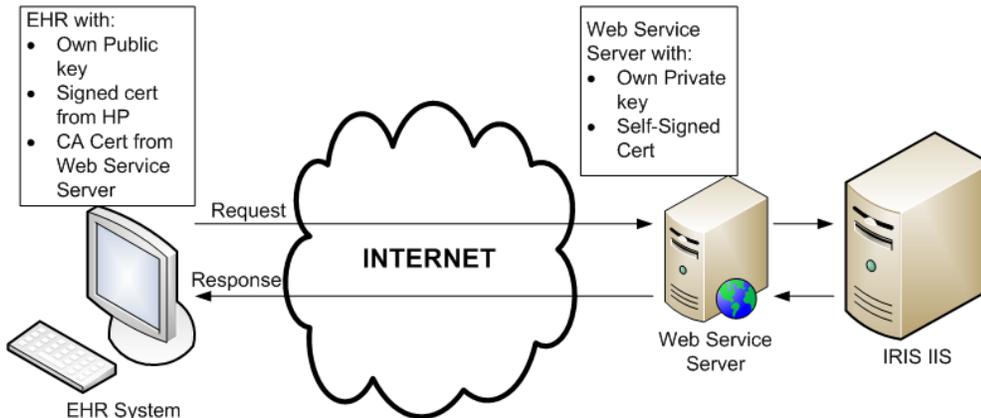
Depending on the browser model and version used, you will now be able to import the CA certificates into the machine's truststore, or export them for importing into a file by openssl.

Whether you need to import the machine's truststore or create a truststore file is determined by the type of Web Service Client you are creating and should be explained in the documentation for that system.

Introduction to SOAP Web Services in IRIS

A Web Service is a standards-based method of allowing one computer to access functions in another computer through the Internet. The functions are accessed through the same ports used by Internet browsers, making them very likely to be allowed through firewalls.

Security is implemented by installing public and private key pairs (known as X.509 certificates) on both the IRIS Web Services Server and the Electronic Health Record (EHR) computer requesting the information. Using this technique, both the client and server are identified to each other to establish trust, and the communication is encrypted.



IRIS supports the following functions through Web Services:

1. Connectivity Test - this is used to test connection to server
2. Submit Single Message - this is used to submit HL7 formatted messages

Web Service functions are called using Simple Object Access Protocol (SOAP) requests, which are formatted as XML (eXtensible Markup Language) messages.

Each SOAP request is made up of the following elements:

1. The envelope, which identifies the message as a SOAP request
2. The function name
3. The parameters of the function call. In most cases, the parameters are individual pieces of data, such as First Name, Last Name, etc. For the IRIS Web Services, there is a single parameter which contains an entire HL7-formatted message, wrapped in a CDATA section to keep it from being misinterpreted by the parser.

Here is a simple VXU message sent to the Update History Web Service function. The pieces are numbered from the list above.

(1) SOAP Envelope Start	<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:urn="urn:cdc:iisb:2011"> <soap:Header/> <soap:Body>
(2) Function and parameter Start	<urn:submitSingleMessage> <urn:username>USERNAME</urn:username> <urn:password>PASSWORD</urn:password> <urn:facilityID>ORG ID</urn:facilityID> <urn:hl7Message
CDATA section Start	<![CDATA[

(3) Parameter, the HL7 message, segments vary depending on the function	MSH ^~\& Immunization Generator^1.4.1 JIM'S CLINIC^6740^ 20120731134645 VXU^V04 64443 P^ 2.4^^^^^^^^^^^^^^ ER PID 3558^^^^PI^ HOST^JAMIE^^^^L^ NAME^MAIDEN^^^^L^ 20110101 F 665 EAST ST^^HOMETOWN^IA^86702^^^^^^ Y PD1 Y NK1 1 HOST^JAMIE^^^^L^ SEL^SELF^HL70063^^^ 665 EAST ST^^HOMETOWN^IA^86702^^^^^^ Y RXA 0 999 20120731 20120731 ^^^ACTHIB^^WVTN 999 A 20120731134645
CDATA section End]]>
(2) Function and parameter End	></urn:hl7Message>
(1) SOAP Envelope End	</urn:submitSingleMessage> </soap:Body> </soap:Envelope>

The definitions of the functions are specified in the WSDL (Web Services Description Language) file. Modern development environments (such as Java and .NET) can take the WSDL and turn it into a programming interface to simplify implementation. The WSDL can be supplied by David Thrall at david.thrall@hp.com or retrieved from the server once the certificates are installed.

For More Information

<http://www.w3schools.com/webservices/default.asp> is a good source for information about Web Services. The Summary page includes links to information about WSDL and SOAP.

Getting the WSDL

The WSDL (Web Services Description Language, pronounced *whiz-dul*) is a method used to describe Web Services functions that can be accessed through the Internet from other computers.

Using the WSDL file, modern development environments (such as Java and .NET) can turn the WSDL file into a programming interface to simplify implementation.

Obtaining the WSDL: Extract the version from this document

1. Highlight the lines in Appendix A (the WSDL section only) from `<?xml version="1.0" encoding="UTF-8"?>` to `</definitions>`
2. Copy the highlighted text to the clipboard with Control-C.
3. Open up Notepad and paste the contents of the clipboard into the empty Notepad file.
4. Verify the contents include all of the lines you highlighted in Step 1.
5. Save document as "cdc.wsdl" to folder of choice.

Getting the URL

The final step is to request the production URL. This will be where your messages are sent and your responses will be returned from.

Appendix A: WSDL File Contents

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:wsaw="http://www.w3.org/2005/08/addressing"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:tns="urn:cdc:iisb:2011"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="urn:cdc:iisb:2011"
  name="IISServiceNew">

  <!-- schema for types -->
  <types>
    <xsd:schema elementFormDefault="qualified" targetNamespace="urn:cdc:iisb:2011">

      <xsd:complexType name="connectivityTestRequestType">
        <xsd:sequence>
          <xsd:element name="echoBack" type="xsd:string" minOccurs="1"
maxOccurs="1" nillable="true"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="connectivityTestResponseType">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:string" minOccurs="1"
maxOccurs="1" nillable="true"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="submitSingleMessageRequestType">
        <xsd:sequence>
          <xsd:element name="username" type="xsd:string" minOccurs="0"
maxOccurs="1" nillable="true"/>
          <xsd:element name="password" type="xsd:string" minOccurs="0"
maxOccurs="1" nillable="true"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>

```

```
                <xsd:element name="facilityID" type="xsd:string" minOccurs="0"
maxOccurs="1" nillable="true"/>
                <xsd:element name="hl7Message" type="xsd:string" minOccurs="1"
maxOccurs="1" nillable="true"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="submitSingleMessageResponseType">
            <xsd:sequence>
                <xsd:element name="return" type="xsd:string" minOccurs="1"
maxOccurs="1" nillable="true"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="soapFaultType">
            <xsd:sequence>
                <xsd:element name="Code" type="xsd:integer" minOccurs="1"/>
                <xsd:element name="Reason" type="xsd:string" minOccurs="1"/>
                <xsd:element name="Detail" type="xsd:string" minOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="UnsupportedOperationFaultType">
            <xsd:sequence>
                <xsd:element name="Code" type="xsd:integer" minOccurs="1"/>
                <xsd:element name="Reason" fixed="UnsupportedOperation"/>
                <xsd:element name="Detail" type="xsd:string" minOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="SecurityFaultType">
            <xsd:sequence>
                <xsd:element name="Code" type="xsd:integer" minOccurs="1"/>
                <xsd:element name="Reason" fixed="Security"/>
                <xsd:element name="Detail" type="xsd:string" minOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="MessageTooLargeFaultType">
            <xsd:sequence>
                <xsd:element name="Code" type="xsd:integer" minOccurs="1"/>
                <xsd:element name="Reason" fixed="MessageTooLarge"/>
            </xsd:sequence>
        </xsd:complexType>
```

```
        <xsd:element name="Detail" type="xsd:string" minOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>

    <xsd:element name="connectivityTest" type="tns:connectivityTestRequestType"/>
    <xsd:element name="connectivityTestResponse"
type="tns:connectivityTestResponseType"/>
    <xsd:element name="submitSingleMessage"
type="tns:submitSingleMessageRequestType"/>
    <xsd:element name="submitSingleMessageResponse"
type="tns:submitSingleMessageResponseType"/>
    <xsd:element name="fault" type="tns:soapFaultType"/>
    <xsd:element name="UnsupportedOperationFault"
type="tns:UnsupportedOperationFaultType"/>
    <xsd:element name="SecurityFault" type="tns:SecurityFaultType"/>
    <xsd:element name="MessageTooLargeFault"
type="tns:MessageTooLargeFaultType"/>

</xsd:schema>
</types>

<!-- Message definitions -->
<message name="connectivityTest_Message">
    <documentation>connectivity test request</documentation>
    <part name="parameters" element="tns:connectivityTest" />
</message>

<message name="connectivityTestResponse_Message">
    <documentation>connectivity test response</documentation>
    <part name="parameters" element="tns:connectivityTestResponse" />
</message>

<message name="submitSingleMessage_Message">
    <documentation>submit single message request.</documentation>
    <part name="parameters" element="tns:submitSingleMessage" />
</message>

<message name="submitSingleMessageResponse_Message">
    <documentation>submit single message response</documentation>
    <part name="parameters" element="tns:submitSingleMessageResponse" />
</message>
```

```
<message name="UnknownFault_Message">
  <part name="fault" element="tns:fault"/>
</message>

<message name="UnsupportedOperationFault_Message">
  <part name="fault" element="tns:UnsupportedOperationFault"/>
</message>

<message name="SecurityFault_Message">
  <part name="fault" element="tns:SecurityFault"/>
</message>
<message name="MessageTooLargeFault_Message">
  <part name="fault" element="tns:MessageTooLargeFault"/>
</message>

<!-- Operation/transaction declarations -->
<portType name="IIS_PortType">
  <operation name="connectivityTest">
    <documentation>the connectivity test</documentation>
    <input message="tns:connectivityTest_Message"
wsaw:Action="urn:cdc:iisb:2011:connectivityTest"/>
    <output message="tns:connectivityTestResponse_Message"
wsaw:Action="urn:cdc:iisb:2011:connectivityTestResponse"/>
    <fault name="UnknownFault" message="tns:UnknownFault_Message"/>      <!-- a general
soap fault -->
    <fault name="UnsupportedOperationFault"
message="tns:UnsupportedOperationFault_Message"/>      <!-- The UnsupportedOperation soap
fault -->
  </operation>

  <operation name="submitSingleMessage">
    <documentation>submit single message</documentation>
    <input message="tns:submitSingleMessage_Message"
wsaw:Action="urn:cdc:iisb:2011:submitSingleMessage"/>
    <output message="tns:submitSingleMessageResponse_Message"
wsaw:Action="urn:cdc:iisb:2011:submitSingleMessageResponse"/>
    <fault name="UnknownFault" message="tns:UnknownFault_Message"/>      <!-- a general
soap fault -->
    <fault name="SecurityFault" message="tns:SecurityFault_Message"/>
    <fault name="MessageTooLargeFault" message="tns:MessageTooLargeFault_Message"/>
  </operation>
</portType>
```

```
<!-- SOAP 1.2 Binding -->
<binding name="client_Binding_Soap12" type="tns:IIS_PortType">
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="connectivityTest">
    <soap12:operation soapAction="urn:cdc:iisb:2011:connectivityTest" />
    <input><soap12:body use="literal" /></input>
    <output><soap12:body use="literal" /></output>
    <fault name="UnknownFault"><soap12:fault use="literal" name="UnknownFault"/></fault>
    <fault name="UnsupportedOperationFault"><soap12:fault use="literal"
name="UnsupportedOperationFault"/></fault>
  </operation>
  <operation name="submitSingleMessage">
    <soap12:operation soapAction="urn:cdc:iisb:2011:submitSingleMessage" />
    <input><soap12:body use="literal" /></input>
    <output><soap12:body use="literal" /></output>
    <fault name="UnknownFault"><soap12:fault use="literal" name="UnknownFault"/></fault>
    <fault name="SecurityFault"><soap12:fault use="literal" name="SecurityFault"/></fault>
    <fault name="MessageTooLargeFault"><soap12:fault use="literal"
name="MessageTooLargeFault"/></fault>
  </operation>
</binding>

<!-- Service definition -->
<service name="cdc">
  <port binding="tns:client_Binding_Soap12" name="client_Port_Soap12">
    <soap12:address location="http://localhost/WebApp/IISService" />
  </port>
</service>
</definitions>
```

Appendix B: URL for OpenSSL

OpenSSL can be downloaded from:

<http://slproweb.com/products/Win32OpenSSL.html>

The version that is downloaded must be the version the technical team is using.

Win64 OpenSSL v1.0.1c	16MB Installer	Installs Win64 OpenSSL v1.0.1c (Only install this if you are a software developer needing 64-bit OpenSSL for Windows. Only installs on 64-bit versions of Windows. Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.
---------------------------------------	----------------	---