



Capario B2B EDI Transaction Connection
Technical Specification for B2B Clients

Revision History

Date	Version	Description	Author
02/03/2006	Draft	Explanation for external B2B clients on how to develop a client interface to connect to our Production B2B network over HTTP(s).	Capario
3/12/2009		Re branded to Capario and updated URLs as needed	Scott Codon

Table of Contents

Introduction	4
Overview	4
Transaction Process Flow	4
Connection Interface	4
How to get Capario root public certificate from Internet Explorer	5
Code Sample	6

Introduction

This document is for B2B clients who will be developing a client interface in order to connect to Capario's B2B production network via HTTPs. Included are details of the Java Connection along with set up information for the external interface of Capario's EDI Transaction Connection.

The Connection consists of a Java Connection hosted by JBoss application server along with a process used to test the integrity of the system. This document is limited to discussing set-up and details regarding the Connection. It does not include any information about the systems it supports, or the various transactions that pass through the Connection

Overview

The major purpose of Capario's EDI Transaction Connection is to:

1. Strip off the web based protocol (in this case https) from the inbound transactions
2. Pass them on to Capario's EDI system
3. Return the transaction results to the requestor

Each transaction is a synchronous request/response interaction, requiring the requestor to wait for a response before continuing.

The Connection runs under the control of JBoss application server and it can handle multiple, simultaneous inbound requests. The Connection is not limited to one request at a time. JBoss handles all threading issues, which greatly simplifies the internal workings of the Connection.

Transaction Process Flow

Listed here are the major transaction process flow steps when an external vendor communicates with Capario's EDI real time system using an internet available Secured HTTPs URL.

1. External vendor sends transactions and these "POST" to the Java Connection running on JBoss application server.
2. The transaction text must be passed in the body of the posted request. *If transaction text is not present the Connection will return a TA1 (Negative Transaction Acknowledgement) and terminate the transaction.*
3. If the transaction text is present in the body of the posted message, the Connection will process the request by forwarding it to Capario's back-end systems.
4. Each message sent has a message expiry timeout value. This is currently set to sixty (60) seconds.
5. If a response is received from Capario's back-end systems, the Connection will write the response string back to the external vendor's application.

Connection Interface

The EDI Transaction Connection has a public interface that can be used like any reusable software component. This means that any external vendor may use the Connection as long as they follow these specifications. The EDI Transaction Connection can support any number of new vendors and transactions without requiring coding changes as long as the interface detailed in this document meets the additional vendor and transaction requirements.

Use of Capario's EDI Transaction Connection requires these conditions so that vendors can connect to the Connection through the internet and pass transactions to Capario:

- Vendors must apply for a Service ID and password through Capario's standard Security Services request process. The ID/password combination MUST be included in ALL transaction requests to the EDI Transaction Connection.

- A digital certificate to establish server to server SSL communication is required. This is due to programming requirements of "POSTing" over HTTPs (SSL)
- Capario's Web Engineering group will assist in providing this certificate or refer to section below, "Obtaining Capario's root public certificate from Internet Explorer"
- During integration/testing phase a URL will be provided to the new vendor so they can open an HTTP(s) connection to the EDI Transaction Connection and "POST" the requested transaction text.
- As noted above the transaction text must be passed in the body of the posted request in order for the Connection to process.
- The transaction request must be a Capario supported transaction. Unsupported transactions will be ignored by Capario's back-end systems and will present a "timeout" condition to the requestor.
- ALL transactions are subject to the Connection's 60-second response timeout.
- If a timeout occurs while processing the transaction with payer system, appropriate time error message is returned back to the users of the Connection.

How to get Capario root public certificate from Internet Explorer

Follow these steps to get the Capario root public certificate from Internet Explorer

1. Open browser
2. Type the following in the browser address line: <https://b2b.Capario.net/b2b/X12Transaction> and click on the "Go" button.
3. You will get a "Blank Page"
4. Find the "gold lock" in the bottom right of the IE window and double click it to show the "Certificate" window
5. Click the Certificate Path tab at the top right of the Certificate window
6. Click on "Capario Root Certificate Authority" to highlight it
7. Click "View Certificate" button in the mid-right portion of the Certificate window
 - 7.1.1. This will open a new Certificate window for "Capario Root Certificate Authority"
8. Click the "Details" tab in this window--click "Copy to File..." button in the bottom right of the window
9. A wizard is displayed - click next
10. Select "Base-64 encoded X.509 (.CER)" radio button and click "Next >"
11. Select a file name to save the file to or click browse to find a location to save the file and click "Next >"
 - 11.1.1. You will need this location later for the keytool steps so write it down or save to the same location as your keystore (commonly <JAVA_HOME>/jre/lib/security)
12. Click finish and you will be prompted that the "export was successful"
13. Import your new certificate into the trust store.

Code Sample

The following code sample is written in the java programming language however other programming languages can be used to communicate with the Connection.

The sample code is given as is and is not supported by Capario. This is an example of what a client program must do in order to communicate with Capario's EDI Transaction Connection.

```
import javax.net.ssl.*;
import java.net.URL;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class CaparioB2BHTTPClient
{
    private String publicCertFilePath;

    public CaparioB2BHTTPClient(String publicCertFilePath)
    {
        this.publicCertFilePath = publicCertFilePath;
    }

    public String process(String urlStr, String request) throws Exception
    {
        TrustManagerFactory tmf = null;
        TrustManager[] trustManagers = null;
        KeyManager[] keyManagers = null;
        SSLContext ctx = null;
        SSLSocketFactory sslSocketFactory = null;
        URL url = null;
        HttpsURLConnection httpsUrlConn = null;

        StringBuffer responseData = new StringBuffer();

        try
        {
            // Create X509 certificate from the public cert file of the remote URL
            CertificateFactory cf = CertificateFactory.getInstance("X.509");
            X509Certificate cert = (X509Certificate) cf.generateCertificate(new
            FileInputStream(publicCertFilePath));

            KeyStore ks = KeyStore.getInstance("JKS");
            ks.load(null, null);
            ks.setCertificateEntry(cert.getSubjectDN().getName(), cert);
```

```

tmf = TrustManagerFactory.getInstance("SunX509");
tmf.init(ks);
trustManagers = tmf.getTrustManagers();

ctx = SSLContext.getInstance("SSL");
ctx.init(keyManagers, trustManagers, new java.security.SecureRandom());
sslSocketFactory = ctx.getSocketFactory();

url = new URL(urlStr);
httpsUrlConn = (HttpsURLConnection) url.openConnection();
httpsUrlConn.setSSLSocketFactory(sslSocketFactory);
httpsUrlConn.setRequestProperty("Content-type", "text/xml");
httpsUrlConn.setDoOutput(true);
httpsUrlConn.setDoInput(true);
httpsUrlConn.setRequestMethod("POST");

PrintWriter out = new PrintWriter(httpsUrlConn.getOutputStream());
out.print(request);
out.flush();
out.close();

int responseCode = httpsUrlConn.getResponseCode();
BufferedReader buffReader = null;
if (responseCode == 200)
{
    buffReader = new BufferedReader(new InputStreamReader(httpsUrlConn.getInputStream()));
}
else
{
    buffReader = new BufferedReader(new InputStreamReader(httpsUrlConn.getErrorStream()));
}

String str = "";
while (null != ((str = buffReader.readLine())))
{
    responseData.append(str);
}
}
catch (Exception e)
{
    e.printStackTrace();
    throw e;
}

return responseData.toString();
}

public static void main(String args[]) throws Exception
{
    String publicCertFilePath = args[0];
    String request = args[1];
    String urlStr = "https://b2b.Capariohealth.net/b2b/X12Transaction";

```

```
CaparioB2BHTTPClient testClient = new CaparioB2BHTTPClient(publicCertFilePath);
String response = testClient.process(urlStr, request);
System.out.println("Response \n" + response);
}
}
```