Blue Cross & Blue Shield of Rhode Island

BCBSRI 27x Real Time API
Connection Procedures

Version 1.0
Status: Draft

August 26, 2005

PREFACE

The *BCBSRI 27x Real Time API* document supplements the *BCBSRI Companion Guides*. Its purpose is to provide trading partners with instructions on transmitting electronic data to Blue Cross & Blue Shield of Rhode Island (hereinafter "BCBSRI").

DISCLAIMER

This document is considered a living document, and as such, the information provided herein will be subject to change after October 16, 2003 in the event that BCBSRI revises its policies or HIPAA Transactions and Code Sets law is updated or amended.

Table of Contents

1.0 Introduction

This document provides instructions for connecting to and using the 27x Real Time API, a Web-based application utilizing Internet browser technology (IP Protocol). It allows the programmatic submission and receipt of individual HIPAA transaction responses. Use this document in conjunction with the *BCBSRI EDI Companion Guides* for each transaction.

2.0 Scope

The procedures in *BCBSRI 27x Real Time API* apply to all of the following transactions:

270/271 Health Care Eligibility Benefit Inquiry and Response
276/277 Health Care Claim Status Request and Response
278 Health Care Services Review – Request for Review and Response

3.0 Contact Information

BCBSRI will work closely with its trading partners to establish effective communication protocols and to resolve any connectivity issues that may arise regarding the exchange of HIPAA-related electronic transactions.

The following contact information is provided to assist in the process of implementing all transactions:

For Partner Testing:

BCBSRI HIPAA EDI Testing Support: 401-459-1970

HIPAA EDI Testing Support business hours are Monday through Friday, 8:15 AM to 4:30 PM.

Email Address: HIPAA.EDI.Support@bcbsri.org

Applicable Web sites: www.BCBSRI.com

For Production:

Call the Perot Systems Service Desk, which supports BCBSRI, at 401- 751-1673 or 1-800-343-5743.

4.0 Using the Web Service

4.1 Logging On

1. Using your Internet browser, enter the Web address provided to you upon completion of the registration process. This address will connect to the BCBSRI 27x Real Time API.

2. The Web Service has been implemented in SOAP, Simple Object Access Protocol, as a single conduit for all transactions and as such there is only one operation in the API, **realTimeX12Transaction**. There is a parameter in the web service operation to denote the transaction requested (270, 276, and 278).

3. To eliminate the need to use CDATA sections in order for the X12 not to trip up XML parsers, binary hexadecimal encoding is applied to the X12 content. This allows the X12 to be a defined parameter to the web service. All development platforms should allow for the simple conversion of text to binary hex. This does roughly double the size of the text but these transactions are small since they are real time which only allows one inquiry per transaction (one ST segment).

NOTE: Both the User ID and password are case sensitive. An alpha-leading User ID and a randomly assigned password will be provided by BCBSRI. The User ID prefixes will change from "T" (for Test) to "P" (for Production). The unique password assigned for testing will be replaced by a new production password.

4.2 Security

Security is accomplished via the use of SSL and an authentication to the service. There are 2 parameters, username and password to pass the credentials to the service. Since all submittals are direct to BCBSRI.COM servers and no intermediaries are used SSL will accomplish confidentiality of all data included.

4.3 Web Service Description (WSDL)

.

The Web Service Operation is **realTimeX12Transaction** and this takes 4 parameters as input.

1. transaction – the HIPAA transaction code (i.e. 270)
2. userName - user ID assigned to you
3. password - password assigned to you
4. inputX12Transaction - X12 real-time HIPAA transaction encoded in hexadecimal binary form

It is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://webservice.realtime"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:intf="http://webservice.realtime"
xmlns:impl="http://webservice.realtime"
xmlns:tns2="http://fault.webservice.realtime">
  <wsdl:types>
    <schema elementFormDefault="qualified"
targetNamespace="http://webservice.realtime"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:impl="http://webservice.realtime"
xmlns:intf="http://webservice.realtime"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <complexType name="RealTimeX12Request">
    <sequence>
     <element name="userName" nillable="true" type="xsd:string"/>
     <element name="password" nillable="true" type="xsd:string"/>
     <element name="transaction" nillable="true" type="xsd:string"/>
     <element name="inputX12Transaction" type="xsd:hexBinary"/>
    </sequence>
   </complexType>
   <element name="realTimeX12Transaction">
    <complexType>
     <sequence>
```

```
      <element name="parameters" nillable="true"
type="impl:RealTimeX12Request"/>
     </sequence>
    </complexType>
   </element>
   <complexType name="RealTimeX12Response">
    <sequence>
     <element name="outputX12Transaction" type="xsd:hexBinary"/>
     <element name="status" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="realTimeX12TransactionResponse">
    <complexType>
     <sequence>
      <element name="realTimeX12TransactionReturn" nillable="true"
type="impl:RealTimeX12Response"/>
     </sequence>
    </complexType>
   </element>
  </schema>
    <schema elementFormDefault="qualified"
targetNamespace="http://fault.webservice.realtime"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:impl="http://webservice.realtime"
xmlns:intf="http://webservice.realtime"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <complexType name="AuthenticationFailedFault">
    <complexContent>
     <extension base="tns2:RealTimeFault">
      <sequence/>
     </extension>
    </complexContent>
   </complexType>
   <complexType name="RealTimeFault">
    <sequence>
     <element name="message" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="AuthenticationFailedFault" nillable="true"
type="tns2:AuthenticationFailedFault"/>
   <complexType name="SystemUnavailableFault">
    <complexContent>
     <extension base="tns2:RealTimeFault">
      <sequence/>
     </extension>
    </complexContent>
   </complexType>
   <element name="SystemUnavailableFault" nillable="true"
type="tns2:SystemUnavailableFault"/>
  </schema>
  </wsdl:types>
  <wsdl:message name="realTimeX12TransactionRequest">
    <wsdl:part name="parameters" element="intf:realTimeX12Transaction"/>
  </wsdl:message>
  <wsdl:message name="SystemUnavailableFault">
    <wsdl:part name="fault" element="tns2:SystemUnavailableFault"/>
  </wsdl:message>
  <wsdl:message name="realTimeX12TransactionResponse">
```

```
    <wsdl:part name="parameters"
element="intf:realTimeX12TransactionResponse"/>
  </wsdl:message>
  <wsdl:message name="AuthenticationFailedFault">
    <wsdl:part name="fault" element="tns2:AuthenticationFailedFault"/>
  </wsdl:message>
  <wsdl:portType name="X12RequestHandlerService">
    <wsdl:operation name="realTimeX12Transaction">
      <wsdl:input name="realTimeX12TransactionRequest"
message="intf:realTimeX12TransactionRequest"/>
      <wsdl:output name="realTimeX12TransactionResponse"
message="intf:realTimeX12TransactionResponse"/>
      <wsdl:fault name="AuthenticationFailedFault"
message="intf:AuthenticationFailedFault"/>
      <wsdl:fault name="SystemUnavailableFault"
message="intf:SystemUnavailableFault"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="X12RequestHandlerServiceSoapBinding"
type="intf:X12RequestHandlerService">
    <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="realTimeX12Transaction">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="realTimeX12TransactionRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="realTimeX12TransactionResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="AuthenticationFailedFault">
        <wsdlsoap:fault name="AuthenticationFailedFault" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="SystemUnavailableFault">
        <wsdlsoap:fault name="SystemUnavailableFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="X12RequestHandlerServiceService">
    <wsdl:port name="X12RequestHandlerService"
binding="intf:X12RequestHandlerServiceSoapBinding">
      <wsdlsoap:address
location="http://localhost:9081/RealTimeServicesWeb/services/X12RequestHandle
rService"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

4.4 Error Conditions

Several faults have been defined that may be thrown from the service.

1. AuthenticationFailedFault – the authentication credentials (username and password) did not match our records
2. AccountLockedFault–authentication failed more than the maximum allowable times and the account was locked as a precaution. This will require a phone call to get the account unlocked.
3. SystemUnavailableFault – the system is unable to respond at this time or did not respond in a timely manner

4. UnknownTransactionFault – the transaction was not one of the permitted values (270, 270T, 276, 276T, 278, 278T).

Note that general soap faults will be thrown in the case of unforeseen issues.

4.5 Sample Input XML

Following is a sample XML file that could be submitted to the real time web service

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<realTimeX12Transaction xmlns="http://webservice.realtime">
<parameters>
<userName>WSUSER1</userName>
<password>abc123</password>
<transaction>270</transaction>
<inputX12Transaction>43412a30302a3030202020202020202020202a30302a3030202020202020202020202a5a5
a2a4e4548454e3030382020202020202a5a5a2a4e4548454e3031302020202020202020202a30333032313
42a313134372a552a30303430312a31323334353635738392a302a502a3a7e47532a48532a30303030303
031522a4e4548454e3031302a32303033303231342a313134372a363434352a582a30303430313058303
9327e53542a3237302a313233347e4248542a303032322a31332a3735343139342d36303131333134392
d33393636323139322a32303033303231342a313134377e484c2a312a2a32302a317e4e4d312a522a322
a426c75653c3e2043726f737320426c756520536869656c642052492a2a2a2a2a50492a4e4548454e3031
307e484c2a322a312a32312a317e4e4d312a31502a322a4c6966657370616e2a2a2a2a2a50492a4e4548
454e3030387e5245462a4e372a545a4b7e484c2a332a322a32322a317e54524e2a312a3735343139342d
363031313331343932d33393636323139322a39394e4548454e3030387e4e4d312a494c2a312a4d554c4c
454e2a414d592a452a2a2a4d492a303337323332311303030337e444d472a44382a31393732303330382
a467e484c2a342a332a32332a307e54524e2a312a3735343139342d363031313331343932d33393636323
139322a39394e4548454e3030387e4e4d312a30332a312a4d554c4c454e2a424142594749524c7e52454
62a53592a3838383838388387e4e342a2a52497e444d472a44382a32303032303330332a467e494e532a
4e2a31397e4454502a3330372a44382a32303032313231377e45512a33302a2a46414d7e53452a32312a
313233347e47452a312a363434357e4945412a312a3132333435363738397e</inputX12Transaction>
</parameters>
</realTimeX12Transaction>
</soapenv:Body></soapenv:Envelope>

4.6 Test Mode

There is a test mode which simply loops back. There is a dummy response returned but it has no correlation to the request. This can be used to ensure that you are communicating correctly with the web service prior to submitting live requests. To enable test mode simply add a “T” suffix to the transaction parameter (i.e. 270T, 276T, and 278T).

Sample output from a 270 submitted in test mode is shown below:

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body><realTimeX12TransactionResponse
xmlns="http://webservice.realtime"><realTimeX12TransactionReturn><outputX12Transaction>4953412a
30302a3030303030303030302a30302a3030303030303030302a5a5a2a3631303031372020202020
202020202a5a5a2a5530303937313331202020202020202a3033303530392a313731312a552a30303430
312a3030303030303039342a302a502a3a0a47532a48422a3631303031372a55303039373133312a3230
3033303530392a313731323030302a3934303030312a582a3030343031305830393241310a53542a3237

312a303030310a4248542a303032322a31312a3237302d3033382d414b2a32303033303530392a313731
320a484c2a312a2a32302a310a4e4d312a50522a322a4243425352492a2a2a2a2a50492a30303337300a
484c2a322a312a32312a310a4e4d312a38302a322a5745535445524c5920484f53504954414c2a2a2a2a
2a53562a303030303030303030390a5245462a4e352a553030393373133310a484c2a332a322a32322a31
0a4e4d312a494c2a312a50414e4f5349414e2a50455445522a2a2a2a4d492a3036353731333531343130
30370a494e532a592a31380a484c2a342a332a32332a300a54524e2a322a3132322d3237302d3033382d
414b2a393030303030303030390a4e4d312a30332a312a434841554c4b2a4a4f534855410a4e332a3132
34205345434f4e442053540a4e342a5041575455434b45542a52492a30323836310a444d472a44382a31
393833303730362a4d0a494e532a4e2a31392a2a2a2a2a2a2a4e2a4e0a45422a362a2a33300a45422a36
2a2a33350a53452a32302a303030310a47452a312a3934303030310a4945412a312a3030303030303039
34</outputX12Transaction><status>DUMMY TEST
RESPONSE</status></realTimeX12TransactionReturn></realTimeX12TransactionResponse></soapenv:
Body></soapenv:Envelope>

4.7 Sample Java SOAP Client Code

Note that the only package used here that is NOT standard Java Web Services Developer Pack API can be downloaded from Apache Jakarta Commons at http://jakarta.apache.org/commons/codec/. It is used to encode/decode the X12 string to/from binary hex. Other jars used are part of the WSPD, Web Services Developer's Pack that can be downloaded directly from SUN.

```
/*
 * Created on Jun 13, 2005
 */
package realtime.webservice.test;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.net.URL;
import java.util.Iterator;

import javax.xml.rpc.Call;
import javax.xml.rpc.Stub;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.Name;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;

import org.apache.commons.codec.binary.Hex;

/**
 * SAMPLE CLIENT APPLICATION USING SOAP
 *
 */
public class SoapTest
{
	private static byte[] soapTest(byte[] bytes) throws Exception
	{
		SOAPConnectionFactory scf =
		  SOAPConnectionFactory.newInstance();
		SOAPConnection con = scf.createConnection();

		MessageFactory mf = MessageFactory.newInstance();
```

```
SOAPMessage msg = mf.createMessage();

SOAPPart part = msg.getSOAPPart();
SOAPEnvelope envelope = part.getEnvelope();

SOAPBody body = envelope.getBody();

Name realTimeX12RequestName = envelope.createName("realTimeX12Transaction", "","http://webservice.realtime");
SOAPBodyElement realTimeX12RequestElement = body.addBodyElement(realTimeX12RequestName);

Name parmsName = envelope.createName("parameters");
SOAPElement parmsElement = realTimeX12RequestElement.addChildElement(parmsName);

Name userName = envelope.createName("userName");
SOAPElement userNameElement = parmsElement.addChildElement(userName);
userNameElement.addTextNode("userName");

Name passwordName = envelope.createName("password");
SOAPElement passwordNameElement = parmsElement.addChildElement(passwordName);
passwordNameElement.addTextNode("password");

Name transactionName = envelope.createName("transaction");
SOAPElement transactionNameElement = parmsElement.addChildElement(transactionName);
transactionNameElement.addTextNode("270T");

Name inputX12TransactionName = envelope.createName("inputX12Transaction");
SOAPElement inputX12TransactionElement = parmsElement.addChildElement(inputX12TransactionName);
char[] hexChars = Hex.encodeHex(bytes);
String x12textNode = new String(hexChars);
inputX12TransactionElement.addTextNode(x12textNode);

msg.saveChanges();
/**
 * Note that the URL here should be modified to reflect the proper host and port
 * the rest of the URL should be correct
 */
URL endpoint = new URL("http://localhost:9080/RealTimeServicesWeb/services/X12RequestHandlerService");
SOAPMessage response = null;
try
{
        response = con.call(msg, endpoint);
}
catch (SOAPException e)
{
        e.printStackTrace();
}
con.close();

/*
 * this parsing code is far from elegant but it gets the parms out!
 */
SOAPPart replyPart = response.getSOAPPart();
SOAPEnvelope replyEnvelope = replyPart.getEnvelope();
SOAPBody replyBody = replyEnvelope.getBody();
String replyX12 = null;
SOAPElement childElement = (SOAPElement) ((Iterator) replyBody.getChildElements()).next();
SOAPElement x12responseElement = (SOAPElement) ((Iterator) childElement.getChildElements()).next();
String status = null;
String X12 = null;

for (Iterator dataElements = x12responseElement.getChildElements(); dataElements.hasNext();)
{
        SOAPElement soapElement = (SOAPElement) dataElements.next();
        Name elName = soapElement.getElementName();
        if (elName.getLocalName().equals("status"))
        {
                status = soapElement.getValue();
        }
        else
        {
                replyX12 = soapElement.getValue();
        }
}
byte[] output = Hex.decodeHex(replyX12.toCharArray());
```

```
        return output;
    }

    private static byte[] getFileBytes(String filename) throws Exception
    {
        // read the XML file contents into a string
        byte[] c;
        try
        {
            File x12Request = new File(filename);
            FileReader rIn = new FileReader(x12Request);

            int len = (int) x12Request.length();
            char[] chars = new char[len];
            rIn.read(chars,(int) 0, len);
            c = new String(chars).getBytes();
        }
        catch (FileNotFoundException e)
        {
            System.out.println("Error:" + e.getMessage());
            e.printStackTrace();
            throw e;
        }
        catch (IOException e)
        {
            System.out.println("Error:" + e.getMessage());
            e.printStackTrace();
            throw e;
        }
        System.out.println("sent image size = " + c.length);
        return c;
    }

    private static void setFileBytes(String filename, byte[] bytes) throws Exception
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        baos.write(bytes);
        FileOutputStream ios = new FileOutputStream(filename);
        ios.write(bytes);
        System.out.println(
        "got image size = "
            + (bytes == null ? "null" : "" + bytes.length));
    }

    public static void main(String[] args) throws Exception
    {
        if (args.length < 2)
        {
            System.out.println("Please supply 2 arguments: \n1. Input filename (X12 request) \n2. Output file name");
        }
        byte[] bytes = getFileBytes(args[0]);
        byte[] output = soapTest(bytes);
        setFileBytes(args[1], output);
    }
}
```

5.0 Document Version Control

Version Number	Date	Modified By	Comments/Revision Details
1.0	August 26 2005		Issued for publication.