Monetra®

XML Protocol Specification

Programmer's Guide v5.1.7
Updated July 2008

This page intentionally left blank.

Table of Contents

1 Revision History

Version	*Date*	*Notes*
V5.1.7	07/10/08	Updated perl example for to fix SSL negotiation with self-signed certificates.
V5.1.6	01/11/08	Updated Java example to show how to use self-signed certificates
v5.1.5	11/16/06	Added Cold Fusion Example
v5.1.4	04/04/06	Added Java Example
v5.1.3	03/22/06	Added PERL Curl Example
v5.1.2	01/26/06	VB6 HTTPS update to ignore SSL errors
v5.1.1	09/20/05	Added VB6 example
v5.1	08/02/05	Initial release

2 Introduction

2.1 XML Overview

XML stands for eXtensible Markup Language and is used to specify the formatting/representation of the data to be sent to Monetra. You will also be required to review the Monetra Client Interface Protocol Specification (http://www.monetra.com/documentation.html) to cross-reference each transaction type, which will have multiple corresponding key/value pairs (ie. username, password, action, etc). You may also reference the examples found later in this document for both formatting and transport of the XML data.

XML parsers are included in many modern languages, or as add-on components. LibXML is a common parser and can be found at http://www.libxml.org.

For more information on the XML standard itself, you may see the official reference at http://www.w3.org/TR/2004/REC-xml-20040204/.

2.2 Basic Transaction Structure

Each XML transaction set must begin with a **<MonetraTrans>** start tag and may contain multiple transactions, each with a corresponding **<Trans>** tag, which must also specify an ***identifier*** element. The sub tags for **<Trans>** are the key/value pairs found in the Monetra Client Interface Protocol Specification.

XML Responses are encapsulated in **<MonetraResp>** tags, and a **<Resp>** tag will exist for every **<Trans>** tag sent, assuming the structure was initially correct. The **<Resp>** tag will have an element of ***identifier,*** which will echo the identifier sent with each **<Trans>** tag. An additional **<DataTransferStatus>** tag will always be sent with an element of ***code***, which may read SUCCESS or FAIL. On FAIL, an error message will be enclosed in the **<DataTransferStatus></DataTransferStatus>** tags.

Every **<Resp>** tag will have a **<Code>** tag, but on comma-delimited responses, you will also have a **<DataBlock>** tag that holds the response. On non-comma delimited data, the response tags will be those defined in the Monetra Client Interface Protocol Specification.

Some transactional characters may need to be encoded when inserting them into the XML data stream. Characters such as '**>**' will become '**>**', '**<**' will become '**<**' and '**&**' will become '**&**'.

2.3 Transport of Data

2.3.1 XML over HTTP or HTTPS

NOTE: *Over public networks (like the Internet) or untrusted private networks, it is required by PABP/CISP that HTTPS is used instead of HTTP for communication to ensure card numbers and other critical data is not transmitted in plain text.*

HTTP is the method of data transfer that webservers use. When you visit a website, that content is delivered to you using the HTTP protocol. HTTPS is simply HTTP tunneled over an SSL connection.

Monetra only supports a small subset of the HTTP protocol for XML transport (keep it simple!). An HTTP(S)-XML request to Monetra should be formatted as a simple HTTP 1.0 or 1.1 POST command. A Content-Length descriptor is also required to allow the Monetra parser to efficiently identify the start and end of the XML data being sent. Please verify that the host and port numbers reference your configuration (IP address of Monetra server, and configured ports for HTTP or HTTPS).

Most programming languages will provide HTTP routines to simplify the programming process. You may also wish to look into CURL as an alternative (http://curl.haxx.se).

HTTP 1.0 is defined in RFC 1945 (http://www.faqs.org/rfcs/rfc1945.html), and HTTP 1.1 is defined in RFC 2616 (http://www.faqs.org/rfcs/rfc2616.html).

2.3.2 XML DropFile

NOTE: *Because of security related issues (such as account numbers being stored in plain text for short periods of time), it is unclear whether or not DropFile support meets the requirements for PABP/CISP communication. It is therefore recommended you use either the HTTP or HTTPS communication methods.*

The DropFile method refers to placing a file in a pre-determined directory, having another program read that file, then writing a result to a file with a similar naming scheme.

In this case, the filename being written must end in the **`.xmlin`** suffix. Monetra will scan for these files in the directory it is configured to search, read the files, remove them, and when processing is complete, write a response file with the **`.xmlout`** suffix.

When a file is found, Monetra will wait up to 30 seconds for the contents of that file to be fully parse-able. If Monetra cannot parse that file within that time-frame, it will delete the file, and write a negative response file with the **`<DataTransferStatus>`** set to a negative state.

There are no headers or footers required for the DropFile method, simply the XML data itself.

3 Examples

3.1 Formatted Transactions

3.1.1 Multi-Transaction Request

```
<MonetraTrans>
      <Trans identifier="1">
            <Username>vitale</Username>
            <Password>test</Password>
            <Action>SALE</Action>
            <Account>4012888888881</Account>
            <ExpDate>0512</ExpDate>
            <Amount>12.00</Amount>
      </Trans>
      <Trans identifier="2">
            <Username>vitale</Username>
            <Password>test</Password>
            <Action>SALE</Action>
            <Account>5454545454545454</Account>
            <ExpDate>0512</ExpDate>
            <Amount>11.00</Amount>
      </Trans>
      <Trans identifier="3">
            <Username>vitale</Username>
            <Password>test</Password>
            <Action>ADMIN</Action>
            <Admin>GUT</Admin>
      </Trans>
</MonetraTrans>
```

3.1.2 Multi-Transaction Response

```
<MonetraResp>
        <DataTransferStatus code="SUCCESS"/>
        <Resp identifier="2">
                <Code>AUTH</Code>
                <Verbiage>APPROVAL 123456</Verbiage>
                <Batch>1</Batch>
                <Item>1</Item>
                <AVS>STREET</AVS>
                <CV>GOOD</CV>
                <TTID>112</TTID>
        </Resp>
        <Resp identifier="3">
                <Code>SUCCESS</Code>
                <DataBlock>
                        ttid,type,capture,...
                        1,SALE,1,...
                        5,PREAUTH,0,...
                        7,RETURN,1,...
                </DataBlock>
        </Resp>
        <Resp identifier="1">
                <Code>DENY</Code>
                <Verbiage>CVV2 MISMATCH</Verbiage>
                <AVS>GOOD</AVS>
                <CV>BAD</CV>
                <TTID>112</TTID>
        </Resp>
</MonetraResp>
```

3.1.3 Response for Bad Structure Transaction

```
<MonetraResp>
        <DataTransferStatus code="FAIL">My descriptive failure reason</
DataTransferStatus>
</MonetraResp>
```

3.2 Source Code: Sending Transactions to Monetra

3.2.1 Command Line

- Take the request example found under section 3.1.1 Multi-Transaction Request
- Write the request to a file named `request.xml`
- If curl (http://curl.haxx.se/) is installed on your system, you can execute it by running the following command:
  `curl -d@request.xml http://testbox.monetra.com:8555/`
- Your transaction will then be sent from the command line to Monetra, and a response will be written in XML to your console.

3.2.2 PHP

If you are using PHP, and have curl enabled (http://www.php.net/curl), you can use something like the following:

```
<?php
        $url="https://localhost:8666"; // Or http://localhost:8555
        $xml_in="<?xml version=\"1.0\" ?>\n" .
             "<MonetraTrans>\n" .
             "\t<Trans identifier=\"1\">\n" .
             "\t\t<Username>vitale</Username>\n" .
             "\t\t<Password>test</password>\n" .
             "\t\t<Action>SALE</Action>\n" .
             "\t\t<Account>4012888888881</Account>\n" .
             "\t\t<ExpDate>0512</ExpDate>\n" .
             "\t\t<Amount>12.00</Amount>\n" .
             "\t</Trans>\n" .
             "\t<Trans identifier=\"2\">\n" .
             "\t\t<Username>vitale</Username>\n" .
             "\t\t<Password>test</password>\n" .
             "\t\t<Action>SALE</Action>\n" .
             "\t\t<Account>5454545454545454</Account>\n" .
             "\t\t<ExpDate>0512</ExpDate>\n" .
             "\t\t<Amount>12.12</Amount>\n" .
             "\t</Trans>\n" .
             "</MonetraTrans>\n";

        $ch = curl_init($url);
        curl_setopt($ch, CURLOPT_POST, 1);
        // If using SSL, don't verify stuff as we're
        // not using a real cert
        if (strncasecmp($url, "https://", 8) == 0) {
            curl_setopt($ch, CURLOPT_SSL_VERIFYHOST,  0);
            curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
        }
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_POSTFIELDS, $xml_in);

        echo "XML_IN: \n";
        echo "$xml_in";
        echo "\n\n";

        $xml_out=curl_exec($ch);

        echo curl_error($ch);
        curl_close ($ch);

        echo "XML_OUT: \n";
        echo "$xml_out";
        echo "\n\n";
?>
```

3.2.3 Perl

If you are using Perl, you may use the WWW::Curl::Easy module in order to perform HTTP and HTTPS XML POSTs. Some examples are available from:
http://search.cpan.org/~crisb/WWW-Curl-2.0/easy.pm.in
The WWW::Curl module may be installed via CPAN or the source may be downloaded from:
http://search.cpan.org/~crisb/WWW-Curl-3.02/
Then to install it, you'd simply extract the archive and 'perl Makefile.PL ; make ; make install'
Here is a quick demonstration for how to use the Perl Curl module with Monetra:

```
#!/usr/bin/perl

use WWW::Curl::Easy;

sub chunk {
      my ($data,$pointer)=@_;
      ${$pointer}.=$data;
      return(length($data));
}
sub chunk_away {
      my ($data,$pointer)=@_;
      # Trash that data
      return(length($data));
}

my $url = "https://localhost:8666/"; # Or http://localhost:8555
my $xml_in = "<?xml version=\"1.0\" ?>\n" .
      "<MonetraTrans>\n" .
      "\t<Trans identifier=\"1\">\n" .
      "\t\t<Username>vitale</Username>\n" .
      "\t\t<Password>test</password>\n" .
      "\t\t<Action>SALE</Action>\n" .
      "\t\t<Account>4012888888881</Account>\n" .
      "\t\t<ExpDate>0512</ExpDate>\n" .
      "\t\t<Amount>12.00</Amount>\n" .
      "\t</Trans>\n" .
      "\t<Trans identifier=\"2\">\n" .
      "\t\t<Username>vitale</Username>\n" .
      "\t\t<Password>test</password>\n" .
      "\t\t<Action>SALE</Action>\n" .
      "\t\t<Account>5454545454545454</Account>\n" .
      "\t\t<ExpDate>0512</ExpDate>\n" .
      "\t\t<Amount>12.12</Amount>\n" .
      "\t</Trans>\n" .
      "</MonetraTrans>\n";

my $xml_out = "";

my $ch = WWW::Curl::Easy::init();
WWW::Curl::Easy::setopt($ch, WWW::Curl::Easy::CURLOPT_URL, $url);
WWW::Curl::Easy::setopt($ch, WWW::Curl::Easy::CURLOPT_POST, 1);
```

```
if (substr($url, 0, 8) eq "https://") {
      print("Using HTTPS\n");
      WWW::Curl::Easy::setopt($ch,
            WWW::Curl::Easy::CURLOPT_SSL_VERIFYHOST, 0);
      WWW::Curl::Easy::setopt($ch,
            WWW::Curl::Easy::CURLOPT_SSL_VERIFYPEER, 0);
}

WWW::Curl::Easy::setopt($ch,
            WWW::Curl::Easy::CURLOPT_POSTFIELDS, $xml_in);

# Let's retrieve the data into a variable, we have to register
# our write callback of 'chunk' from above
WWW::Curl::Easy::setopt($ch,
            WWW::Curl::Easy::CURLOPT_WRITEFUNCTION, \&chunk);
# Not actually writing to a file, just memory!
WWW::Curl::Easy::setopt($ch,
            WWW::Curl::Easy::CURLOPT_FILE, \$xml_out);
# Throw away bytes from header, if we don't do this, it will
# print header to stdout
WWW::Curl::Easy::setopt($ch,
            WWW::Curl::Easy::CURLOPT_HEADERFUNCTION, \&chunk_away);


print("XML In:\n" . $xml_in . "\n\n");

$ret = WWW::Curl::Easy::perform($ch);
if ($ret != 0) {
      print("XML POST FAILED\n");
      WWW::Curl::Easy::cleanup($ch);
      exit(1);
}

print("XML Out:\n" . $xml_out . "\n\n");
WWW::Curl::Easy::cleanup($ch);
exit(0);
```

3.2.4 Java

This Java example makes use of the standard classes that come with a default install of recent Sun Java JDKs. It has no external dependancies other than those. In the example, depending on if http:// or https:// is specified, it will use the corresponding class in order to establish an unencrypted or encrypted session. Also included is an XML parsing example. To compile and test, simply copy and paste the example into a file called MHTTPTest.java and run '`javac MHTTPTest.java`' to compile the application and '`java MHTTPTest`' to run the application.

```
/* HTTP/HTTPS post */
import java.net.*;
import java.io.*;
import javax.net.ssl.*;

/* XML Parsing */
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

public class MHTTPTest {
      public static void main(String[] args) {
            /* Or http://localhost:8555 */
            String url = "https://localhost:8666";
            String xmlin =    "<?xml version=\"1.0\" ?>\r\n" +
                        "<MonetraTrans>\r\n" +
                        "\t<Trans identifier='1'>\r\n" +
                        "\t\t<Username>vitale</Username>\r\n" +
                        "\t\t<Password>test</password>\r\n" +
                        "\t\t<Action>SALE</Action>\r\n" +
                        "\t\t<Account>4012888888881</Account>\r\n" +
                        "\t\t<ExpDate>0512</ExpDate>\r\n" +
                        "\t\t<Amount>12.00</Amount>\r\n" +
                        "\t</Trans>\r\n" +
                        "\t<Trans identifier='2'>\r\n" +
                        "\t\t<Username>vitale</Username>\r\n" +
                        "\t\t<Password>test</password>\r\n" +
                        "\t\t<Action>SALE</Action>\r\n" +
                        "\t\t<Account>5454545454545454</Account>\r\n" +
                        "\t\t<ExpDate>0512</ExpDate>\r\n" +
                        "\t\t<Amount>12.12</Amount>\r\n" +
                        "\t</Trans>\r\n" +
                        "</MonetraTrans>\r\n";
            String xmlout = "";
            try {
                  URL u = new URL(url);
                  HttpsURLConnection hsc = null;
                  HttpURLConnection hc = null;
                  OutputStream out = null;
                  InputStream in = null;
                  System.out.print("Connecting to " + url + " ...");
                  if (url.startsWith("https://")) {
```

```
                        /* If your server uses a self-signed
                           certificate, default to trusting
                           all certs */
                        X509TrustManager trustAllCerts =
                              new X509TrustManager() {
                                    public
                              java.security.cert.X509Certificate[]
                                    getAcceptedIssuers() {
                                    return null;
                                    }
                              public void checkServerTrusted(
                              java.security.cert.X509Certificate[]
                                    certs, String authType) throws
                              java.security.cert.CertificateException {
                                    return;
                                    }
                              public void checkClientTrusted(
                              java.security.cert.X509Certificate[]
                                    certs, String authType) throws
                              java.security.cert.CertificateException {
                                    return;
                                    }
                        };
                        try {
                              SSLContext sc =
                                    SSLContext.getInstance("TLS");
                              sc.init(null, new TrustManager[] {
                                    trustAllCerts
                                    }, null);
                        HttpsURLConnection.setDefaultSSLSocketFactory(
                              sc.getSocketFactory());
                        } catch (Exception e) {
                        }

                        /* Open Connection */
                        hsc = (HttpsURLConnection)u.openConnection();

                        /* If your hostname does not match the
                           certificate, disable checking */
                        HostnameVerifier hnverify =
                              new HostnameVerifier() {
                              public boolean verify(String hostname,
                                          SSLSession session) {
                                    /* Dont care about host
                                     * validation */
                                    return true;
                              }
                        };
                        hsc.setHostnameVerifier(hnverify);

                        /* Remaining flags */
                        hsc.setDoOutput(true);
                        hsc.setDoInput(true);
```

```
                    hsc.setRequestMethod("POST");
                    out = hsc.getOutputStream();
              } else {
                    hc = (HttpURLConnection)u.openConnection();
                    hc.setDoOutput(true);
                    hc.setDoInput(true);
                    hc.setRequestMethod("POST");
                    out = hc.getOutputStream();
              }

              System.out.println("success!");
              System.out.println(     "---------------------" +
                                      "----------------");
              System.out.println("Sending XML");
              System.out.println(     "---------------------" +
                                      "----------------");
              System.out.println(xmlin);

              /* Write XML data */
              OutputStreamWriter wout =
                    new OutputStreamWriter(out, "UTF-8");
              wout.write(xmlin);
              wout.flush();
              out.close();

              if (url.startsWith("https://")) {
                    in = hsc.getInputStream();
              } else {
                    in = hc.getInputStream();
              }

              /* Read xml from input stream */
              StringBuffer myout = new StringBuffer();
              byte[] b = new byte[4096];
              for (int n; (n = in.read(b)) > 0;) {
                    myout.append(new String(b, 0, n));
              }
              xmlout = myout.toString();

              in.close();

              if (url.startsWith("https://")) {
                    hsc.disconnect();
              } else {
                    hc.disconnect();
              }

        }
        catch (IOException e) {
              System.err.println(e);
              e.printStackTrace();
        }
```

```
            System.out.println(     "--------------------" +
                                    "----------------");
            System.out.println("Read XML");
            System.out.println(     "--------------------" +
                                    "----------------");
            System.out.println(xmlout);
            System.out.println(     "--------------------" +
                                    "----------------");
            System.out.println("Parsing XML");
            System.out.println(     "--------------------" +
                                    "----------------");
            /* Parse the XML */
            try {
                  DocumentBuilderFactory builderFactory =
                        DocumentBuilderFactory.newInstance();
                  DocumentBuilder builder =
                        builderFactory.newDocumentBuilder();

                  InputStream xmlstream =
                        new ByteArrayInputStream(
                              xmlout.getBytes("UTF-8")
                              );
                  Document response = builder.parse(xmlstream);

                  Element docEle = response.getDocumentElement();
                  NodeList nl;
                  Element el;

                  /* Make sure the response in general is good */
                  nl = docEle.getElementsByTagName(
                              "DataTransferStatus"
                              );
                  if (nl == null || nl.getLength() <= 0) {
                        System.exit(1);
                  }
                  el = (Element)nl.item(0);
                  String docstatus = el.getAttribute("code");
                  if (docstatus.compareToIgnoreCase("SUCCESS") != 0) {
                        System.out.println("DataTransferStatus" +
                                           " returned: "
                                           + docstatus);
                        System.exit(1);
                  }

                  /* Ok, so Monetra said the data we sent was good,
                   * so we should have a good response coming
                   * back... */

                  /* Cycle through responses */
                  nl = docEle.getElementsByTagName("Resp");
                  if (nl == null || nl.getLength() <= 0) {
                        System.out.println("No responses returned!");
                        System.exit(1);
```

```
                }
                for (int i = 0; i < nl.getLength(); i++) {
                      el = (Element)nl.item(i);
                      String id = el.getAttribute("identifier");
                      System.out.println("RESPONSE READ: " +
                                         " Identifier '" +
                                         id + "'");

                      NodeList childNl = el.getChildNodes();
                      if (childNl == null ||
                          childNl.getLength() <= 0) {
                            System.out.println("No data " +
                                               "in response!");
                            System.exit(1);
                      }
                      for (int j=0; j<childNl.getLength(); j++) {
                            Node childEl = childNl.item(j);
                            if (childEl.getNodeType() !=
                                childEl.ELEMENT_NODE)
                                  continue;

                            String key = childEl.getNodeName();
                            String val =
                              childEl.getFirstChild().getNodeValue();
                            System.out.println(key + " = " + val);

                            if (key.compareToIgnoreCase("code") != 0)
                                  continue;

                            /* key is 'code', check to see if
                             * transaction was successful */
                            if (val.compareToIgnoreCase("auth") == 0)
                                  System.out.println("**Transaction"
                                        + " Successful!**");
                            else
                                  System.out.println("**Transaction"
                                        + " Failed!**");
                      }
                      System.out.println("----------------------" +
                                         "----------------");
                }
          }
          catch (Exception e) {
                System.err.println(e);
                e.printStackTrace();
          }
     }
}
```

3.2.5 Visual Basic 6

To perform an HTTP XML POST with Visual Basic 6, you will need to ensure that you have MSXML2 libraries installed. You may download those from:
http://www.microsoft.com/downloads/details.aspx?FamilyID=3144b72b-b4f2-46da-b4b6-c5d7485f2b42&DisplayLang=en
For which the methods are documented here:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/4ead2a13-de31-487e-b826-53f132f25cbb.asp

Please ensure you select custom when installing the downloaded msxml.msi, and install the SDK components.

Within Visual Basic 6, you will also need to go under Project->References to ensure 'Microsoft XML, v4.0' is selected to utilize MSXML2.

```
Const strURL = "https://localhost:8666"  ' or http://localhost:8555
Const strXML = "<?xml version='1.0' ?>" _
      & "<MonetraTrans>" _
      & "<Trans identifier='1'>" _
      & "<Username>vitale</Username>" _
      & "<Password>test</password>" _
      & "<Action>SALE</Action>" _
      & "<Account>4012888888881</Account>" _
      & "<ExpDate>0512</ExpDate>" _
      & "<Amount>12.00</Amount>" _
      & "</Trans>" _
      & "<Trans identifier='2'>" _
      & "<Username>vitale</Username>" _
      & "<Password>test</password>" _
      & "<Action>SALE</Action>" _
      & "<Account>5454545454545454</Account>" _
      & "<ExpDate>0512</ExpDate>" _
      & "<Amount>12.12</Amount>" _
      & "</Trans>" _
      & "</MonetraTrans>"

Public Sub Main()
      Dim xmlHttp As New MSXML2.ServerXMLHTTP40
      Dim xmlDom As New DOMDocument40
      Dim xmlResp As String
      Dim retval As Integer
      Dim xmlDoc As New MSXML2.DOMDocument40
      Dim oNode As IXMLDOMNode
      Dim oNodes As IXMLDOMNodeList
      Dim pNode As IXMLDOMNode
      Dim attr As IXMLDOMAttribute
      Dim output As String
      Dim Status As String

      xmlHttp.open "Post", strURL, False
```

```
' Some newer versions of MSXML2 seem to not default to ignoring
'   all SSL errors as MSXML2 v4.0 did.  Set that option here as
'   most people don't implement fully signed certificates in
'   Monetra:
' Must set SXH_OPTION_IGNORE_SERVER_SSL_CERT_ERROR_FLAGS (2)
'   to SXH_SERVER_CERT_IGNORE_ALL_SERVER_ERRORS (13056)
xmlHttp.setOption 2, 13056 ' ignore SSL errors
xmlHttp.setRequestHeader "Content-Type", "text/xml"
xmlHttp.send strXML
xmlResp = xmlHttp.responseXML.xml

Set xmlHttp = Nothing

xmlDom.loadXML (xmlResp)

' check return code
Set oNode = xmlDom.selectSingleNode( _
      "//MonetraResp/DataTransferStatus" _
      )

If (Not oNode Is Nothing) Then
      Set attr = oNode.Attributes.Item(0)
      If (attr.Name = "code") Then
            Status = attr.Value
            output = output + "Status : " + _
                  Status + vbNewLine + vbNewLine
            End If
      End If

      ' process response(s)
      Set oNodes = xmlDom.selectNodes("//MonetraResp/Resp")
      If (Not oNodes Is Nothing) Then
      For i = 0 To oNodes.length - 1
            Set oNode = oNodes.nextNode
            If (oNode.nodeName = "Resp") Then
                  ' find the "identifier"
                  Set attr = oNode.Attributes.Item(0)
                  If (attr.Name = "identifier") Then
                        output = output + "Response " + _
                              attr.Value + vbNewLine
                  End If
                  ' find the "ttid"
                  Set pNode = oNode.selectSingleNode("ttid")
                  If (Not pNode Is Nothing) Then
                        output = output + "  TTID: " + _
                              pNode.nodeTypedValue + vbNewLine
                  End If
                  ' find the return codes for this transaction
                  Set pNode = oNode.selectSingleNode("code")
                  If (Not pNode Is Nothing) Then
                        output = output + "  Code: " + _
                              pNode.nodeTypedValue + vbNewLine
                  End If
```

```
                        Set pNode = oNode.selectSingleNode( _
                                    "msoft_code" _
                                    )
                        If (Not pNode Is Nothing) Then
                              output = output + "  msoft_code: " + _
                                     pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode( _
                                    "phard_code" _
                                    )
                        If (Not pNode Is Nothing) Then
                              output = output + "  phard_code: " + _
                                     pNode.nodeTypedValue + vbNewLine
                        End If
                        ' process other parameters
                        Set pNode = oNode.selectSingleNode("verbiage")
                        If (Not pNode Is Nothing) Then
                              output = output + "  Verbiage: " + _
                                      pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode("auth")
                        If (Not pNode Is Nothing) Then
                              output = output + "  AUTH: " + _
                                      pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode("batch")
                        If (Not pNode Is Nothing) Then
                              output = output + "  Batch: " + _
                                      pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode("item")
                        If (Not pNode Is Nothing) Then
                              output = output + "  Item: " + _
                                      pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode("raw_code")
                        If (Not pNode Is Nothing) Then
                              output = output + "  Raw Code: " + _
                                     pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode("raw_avs")
                        If (Not pNode Is Nothing) Then
                              output = output + "  Raw AVS: " + _
                                      pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode("timestamp")
                        If (Not pNode Is Nothing) Then
                              output = output + "  Timestamp: " + _
                                      pNode.nodeTypedValue + vbNewLine
                        End If
                        Set pNode = oNode.selectSingleNode("pclevel")
                        If (Not pNode Is Nothing) Then
                              output = output + "  PCLevel: " + _
```

```
                                        pNode.nodeTypedValue + vbNewLine
                        End If
                        output = output + vbNewLine
                    End If
                Next
            End If
            ' output "output" as appropriate
End Sub
```

3.2.6 Cold Fusion

The Cold Fusion example makes use of only stock tags and functions. It has been tested primarily on Cold Fusion MX 7, but should work all the way back to Cold Fusion 4 (which introduced the *<cfhttp>* tag).

To test this example, you need to name the code snippet below 'monetra_demo.cfm', and place it in your Cold Fusion web root/directory/folder. You may then point a web-browser at your server and go to the URL where the monetra_demo.cfm resides.

Please note that if you have issues with 'https://' addresses, and you are using a self-signed certificate with Monetra, you will need to import your CA certificate so that ColdFusion will not disallow the connection. You can follow the 'Enable ColdFusion for SSL communication' section of this guide:
http://www.adobe.com/cfusion/knowledgebase/index.cfm?id=tn_19139#enableCF

```
<!--- Set These Variables --->
<cfscript>
myurl='http://testbox.monetra.com:8555';
username='vitale';
password='test';
</cfscript>

<html>
<head><title>CF Monetra Demo</title></head>
<body>
<cfif IsDefined("Form.submit") is not TRUE>
      <!--- Output form to be filled out --->
      <H1>Monetra DEMO</H1>
      <hr>
      <B>Please Enter the transaction details for this test:</B><BR>
      <form name="Form" action="monetra_demo.cfm" method="POST">
            Amount:
            <Input name="amount" type="text" size="9" maxlength="9">
            <br>
            Card Number:
            <Input name="account" type="text" size="30" maxlength="30">
            <br>
            Expiration (MMYY):
            <Input name="expdate" type="text" size="4" maxlength="4">
            <br>
            Billing Street:
            <Input name="street" type="text" size="30" maxlength="30">
            <br>
            Billing Zip:
            <Input name="zip" type="text" size="9" maxlength="9">
            <br>
            CV:
            <Input name="cv" type="text" size="4" maxlength="4">
            <br>
            <Input type="Submit" name="submit" value="Process">
```

```
      </form>
      <br>
<cfelse>
      <!--- Set all parameters for a monetra transaction --->
      <cfset MonetraTrans = StructNew()>
      <cfset val = StructInsert(MonetraTrans, "username", username)>
      <cfset val = StructInsert(MonetraTrans, "password", password)>
      <cfset val = StructInsert(MonetraTrans, "action", "sale")>
      <cfset val = StructInsert(MonetraTrans, "amount", amount)>
      <cfset val = StructInsert(MonetraTrans, "account", account)>
      <cfset val = StructInsert(MonetraTrans, "expdate", expdate)>
      <cfset val = StructInsert(MonetraTrans, "street", street)>
      <cfset val = StructInsert(MonetraTrans, "zip", zip)>
      <cfset val = StructInsert(MonetraTrans, "cvv2", cv)>

      <!--- Fairly generic way to turn a MonetraTrans structure
            into XML --->
      <cfset outXML = "<MonetraTrans>#Chr(10)##Chr(13)#">
      <cfset outXML = "#outXML#<Trans identifier='1'>">
      <cfset outXML = "#outXML##Chr(10)##Chr(13)#">
      <cfloop collection="#MonetraTrans#" item="x">
            <cfset outXML = "#outXML#<#x#>#MonetraTrans[x]#</#x#>">
            <cfset outXML = "#outXML##Chr(10)##Chr(13)#">
      </cfloop>
      <cfset outXML = "#outXML#</Trans>#Chr(10)##Chr(13)#">
      <cfset outXML = "#outXML#</MonetraTrans>#Chr(10)##Chr(13)#">

      <cfoutput>
            Sending XML to #myurl#:
            #HTMLCodeFormat(outXML)#<BR>
      </cfoutput>
      <!--- Send XML message to Monetra --->
      <cfhttp url="#myurl#" method="post" result="xml_response">
            <cfhttpparam type="XML" name="XmlDoc" value="#outXML#">
      </cfhttp>
      <cfoutput>
            XML Response:
            #HTMLCodeFormat(xml_response.FileContent)#<BR>
      </cfoutput>
      <!--- PARSE Response --->
      <cfset MonetraResp = XmlParse(xml_response.FileContent)>
      <cfset RespStruct = MonetraResp["MonetraResp"]>
      <cfset TransferStatus = RespStruct["DataTransferStatus"]>
      <cfset dataTransferCode = TransferStatus.XmlAttributes["code"]>
      <cfoutput>
            <BR><BR><B>
            =====PARSED RESPONSE OUTPUT=====
            </B><BR><BR>
      </cfoutput>
      <cfif dataTransferCode is "SUCCESS">
            <!--- Loop through response(s) --->
            <cfloop collection="#RespStruct#" item="i">
                  <cfif i is "Resp">
```

```
                        <cfset Resp=RespStruct[i]>
                        <cfoutput>
                         <B>RESPONSE id
                         #Resp.XmlAttributes["identifier"]#</B>
                         <BR>
                        </cfoutput>
                        <!--- Loop through each element of the
                              response --->
                        <cfloop collection="#Resp#" item="j">
                              <cfoutput>
                              #j# = #Resp[j]#<BR>
                              </cfoutput>
                        </cfloop>
                  </cfif>
            </cfloop>
      <cfelse>
            <cfoutput>
                  Bad Transaction: #dataTransferCode#
            </cfoutput>
      </cfif>
      <cfoutput>
            <BR><B>======END RESPONSE OUTPUT======</B><BR>
      </cfoutput>
</cfif>
</body>
</html>
```