HTTPS Gateway

Merchant Specification

Version 2.3.0

This document has been created by the Wirecard AG. Its contents may be changed without prior notice. External web links are provided for information only. Wirecard does not claim liability for access to and correctness of the referenced content.

COPYRIGHT

Printed in Germany / European Union

Version: Version 2.3.0

Last Update : 14-Nov-11

TRADEMARKS

CONTACT INFORMATION

For questions relating to this document please contact:

Wirecard Technologies AG

Einsteinring 35

D-85609 Aschheim

Germany

phone: +49 89 4424 1640

e-mail: support@wirecard.com

Table of Contents

1 General

1.1 Audience

This specification is primarily intended for merchants connecting to the Wirecard platform via the HTTPS gateway. It is probably the most central specification of the processing documentation.It describes the basics of HTTPS and outlines why this secure version of the HyperText Transfer Protocol should be used in the exchange of sensitive payment data. Finally the document discusses how HTTPS access can be configured in XML request messages over.

1.2 Document Conventions

This document uses the following conventions:

- Monospace/Courier font for code and code listings, file names, commands, path names, directory names, Hypertext Markup Language (HTML) tags, and any text that must be typed on the screen.
- *Italic* font to represent placeholder parameters (variables) that should be replaced with an actual value, or items that require *emphasis*..
- Brackets ([]) are used to enclose optional parameters.
- Slashes (/) to separate path directories.
- External web links are provided for information only. Wirecard does not claim liability for access to and correctness of the referenced content.

1.3 Software Requirements

- Internet connection supporting HTTPS (or alternatively on a dedicated TCP/IP connection)
- Working knowledge of XML
- Basic knowledge of PHP, Java, Perl, C++ or any other supported programming language
- SSLv3 or TLSv1 supporting 128-bit encryption (or stronger) to meet PCI requirements.

1.4 References

- Card Processing – Merchant Specification
- EFT Processing – Merchant Specification
- Card & Account Management – Customer Specification
- 3-D Secure Card Processing – Merchant Specification
- Risk Management – Merchant Specification
- Uniform Resource Identifier (URI): Generic Syntax (RFC 2396)
- UTF-8, a transformation format of ISO 10646 (RFC 2279)

1.5 Revision History

This specification is periodically updated to document all design changes made to EFT processing interface. With each revision a new entry is added to the table below, including the date of and the reason for the version change. Additionally, vertical revision bars are placed in the margins to indicate the changes in the text.

Date	Version	Comments
2007-08-06	2.0.0	Chapter 3 restructured. New section on Card Management (SCP) added.
2007-11-06	2.1.0	Notes added to the PHP, Perl and Java script examples in Section 3.2 (Payment Gateway) indicating that the XML transaction request is added for demonstration only.
2008-12-01	2.2.0	Section 3.4 (Card & Account Management) updated and document adapted to new layout
2011-11-14	2.3.0	Changed Risk Management test-system URL

1.6 Character Set

The character set suggested for setting up HTTPS connectivity is UTF-8, a compressed version of unicode that uses a single byte for the ASCII 0-127 characters. A plain ASCII string is therefore also a valid UTF-8 string. UTF-8 is the default character set in XML specifications. It supports most international character sets.

1.6.1 Different Character Sets

It is important that the character set defined in your XML request message is identical to the one set on your server. If you are using umlauts (special characters) in your XML request, any discrepancy in the character sets (UTF-8 vs ISO-8859-1) will result in system errors.

For example, if your server is set to ISO-8859-1 (Western European Character set) but you decide to use UTF-8 in your XML scripts, the following error message is generated

```
<?xml version='1.0' encoding='UTF-8'?>
<WIRECARD_BXML xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='wirecard.xsd'>
    <W_RESPONSE>
      <ERROR>
        <Type>SYSTEM_ERROR</Type>
        <Number>10001</Number>
        <Message>Wrong character set specified.</Message>
        <Advice>Please set the xml encoding to ISO-8859-1 or UTF
        8.</Advice>
      </ERROR>
    </W_RESPONSE>
</WIRECARD_BXML>
```

2 Basics

2.1 What is XML

XML, the eXtensible Markup Language, is designed to carry data in predefined tags. It does not replace HTML, the HyperText Markup Language, but complements it. XML picks up where HTML leaves off. It allows specific markup to be created for specific data. HTML does not clearly distinguish between different markups. Markup for look and links gets mixed in with data. If you change the look of HTML markups, embedded links may be lost. If you change the HTML markup for links, the look may be lost. This is not the case with XML. Unlike HTML, the eXtensible Markup Language is not only flexible but also easy to customize and maintain.

All Wirecard system messages are XML v1.0 documents. They can be implemented using a variety of source code and scripts like Java, Perl, C++, or PHP. The formats of the XML request and response messages are defined as Document Type Definitions (DTDs) and XSD schema. A DTD or schema specifies a set of rules for the format of an XML document. Each request and response message includes a reference to it's associated DTD / Schema when encoded in XML. The included DTD reference is represented as a relative URI (Uniform Resource Identifier, which is the DTD name only) to allow different clients and servers to store DTDs in different locations. When receiving an XML request, the recipient must resolve the relative URI pointing to a location where the DTD is located within the recipient's application context. Wirecard platform messages conforms to the constraints of its corresponding DTD.

2.2 What is HTTPS

HTTPS is the secure version of HTTP, the **H**yper**T**ext **T**ransfer **P**rotocol. It is not to be confused with S-HTTP, a security-enhanced version of HTTP developed and proposed as a standard by EIT. HTTPS is a web protocol developed to encrypt and decrypt user requests as well as the data that is returned by the web server. The communication between web server and client is based on certificates which are described in Section 2.3.

HTTPS runs on the open, non-proprietary protocol Secure Socket Layer (SSL) at a standard 128-bit encryption. It is therefore sometimes also referred to as the HTTP over SSL. HTTPS is a sublayer of the regular HTTP application layer. While HTTP uses port 80, HTTPS uses port 443 in its interactions with the lower TCP/IP layer. HTTPS and SSL support X.509 digital certificates which means that a user can authenticate the sender.

When you visit an online shopping site, you will typically connect to the merchants web server using HTTPS. The web server will invoke an order form that starts with *https://.* When you click "Send" to submit your order, your browser's HTTPS layer will encrypt it. The recipient server will acknowledge your request with an encrypted response. It is returned with an https:// URL and decrypted by your browser's HTTPS sublayer. Without SSL encryption, digital information travels across networks in full view. It is as if your bank sends you a postcard with your account balance and credit card use.

2.3 What is a Certificate

To guard the confidentiality of sensitive data sent over the Internet, any HTTPS connection requires the use of certificates. A certificate is the information with which a trusted third party called Certification Authority (CA) authenticates (certifies) and digitally signs a public key (see Section 2.3.1). A certificate identifies client and server and enhances the traditional security of HTTP using remote login with plain username and password. In combination with HTTPS and the underlying SSL protocol, a certificate verifies the authenticity of the remote machine and provides encryption of the information exchanged between client and server.

2.3.1 Public Key Encryption

Servers communicating over the Internet should ideally protect all incoming and outgoing data from prying eyes. To prevent interception and tampering of confidential data by phishing attacks and spoof sites, merchants should secure their online transactions. This is typically done by public key encryption, the most common and secure technology in use today.

Public key encryption is based on a randomly generated symmetric session key and two separate yet mathematically related asymmetric keys, a public key and private key. When a connection between client and server is established, the one-time session key is used to secure (encrypt and decrypt) the data transmitted in each session. It hashes the message meaning it jumbles it to unreadable format. Because the same session key is used for both encryption and decryption, the hashed message and the session key which is sent along with the message is secured with the recipient's asymmetric public key. As the name suggests, the public key is publicly available. It is signed, stored and shared by the trusted CA as a certificate and is used to encrypt the session key. Using the corresponding private key (which is used only by key owner), the incoming data is decrypted.

In line with the Internet security standard developed by the Internet Engineering Task Force (IETF), each Wirecard server has a unique certificate offering a 1024-bit RSA encryption. The certificate verifies and confirms the authenticity of the Wirecard application server, proving to you that the remote machine you are trying to connect to is actually hosted by Wirecard AG. As a professional merchant you are advised to use and check certificates when you process transactions online, receive/send personal and sensitive information, and wish to inspire trust in those who communicate with you. When you send a transaction request to the Wirecard application server over an HTTPS connection, your server encrypts the message with the CAcertified public key from Wirecard. When we receive your request message, we will verify the authenticity of your message and decrypt the data with the private key of our server.

2.3.2 Root Certificate

A root certificate is a top-level certificate issued by the Certification Authority to verify the integrity of other certificates chained hierarchically below, such as web or transaction server certificates. Web browser applications like Internet Explorer, Mozilla, Safari or Opera typically come with a pre-installed set of most common root certificates. When an online shopper connects to an SSL-secured website from his computer, the browser validates the server certificate of the online merchant with the locally stored root certificate of the trusted CA. If the root certificate has expired or it is not included in the version of the browser application used (because it has been issued by an unfamiliar CA) the connection setup fails and a Browser Alerts message appears.

A company connecting from their transaction server to that of a business partner across the Internet should always ensure that their transaction server is running a valid root certificate which can verify their partner's server certificate. If the server does not have a valid root certificate installed, it cannot establish an SSL connection to the remote server of the business partner. We therefore recommend that Wirecard customers regularly check the validity of the root certificates on their servers and update them as described in the next section.

2.3.3 Certificate Check

The Wirecard certificate is validated and issued by the Certification Authority *TC TrustCenter*. When you access the Wirecard application server through a web browser for the first time, the browser will check the certificate by contacting the certificate directory of the issuing CA. As a trusted CA, *TC TrustCenter* is automatically referenced in the browser's default list of CAs. In the unlikely event that the browser does not recognize the referenced certificate (name and validity) or the issuing CA, you will be alerted to the security risk. Depending on the type of web browser in use you can choose to view the certificate, accept it temporarily or permanently, continue without a certificate check or abort the connection set-up completely. To update a certificate manually, see Installing or Updating Certificates below.

Browser Alerts

When you connect through a browser to a remote server, the browser verifies that the server certificate has been issued by a trusted CA. If the certificate is unknown, a warning text will appear (see below). In the popup which appears you can view the unknown certificate and decide if you wish to reject it or accept it temporarily or permanently

If you are setting up an HTTPS connection in a command line there will be no browser feature checking the certificate of the remote server for you. In this case you will have to verify manually if the certificate exists in the server's certificate registry and if it is still valid. Depending on the operating system of your machine and the programming language used to establish the connection, there are different ways to check the availability and validity of a certificate. If the one in question is missing, it is quite simple to add it manually using a web browser or alternatively a proprietary program and save it to your server's central certificate registry (see the next section for more details).

Java

When establishing HTTPS connectivity using Java, you normally do not need to do a manual certificate check or install the Wirecard certificate. Java typically comes with root certificates of known CAs. When you connect to the Wirecard application server, your Java Virtual Machine (JVM) verifies our server by automatically connecting to the root certificate of the issuing CA. If your server is connected to the Internet and it is possible to reach the root certificate of the CA issuing the Wirecard certificate, you do not need to do anything. However, if you are, for example, pointing to a server on a network segment that is not connected to the Internet and therefore cannot verify the root certificate of the referenced CA, you need to install the Wirecard certificate manually to the keystore of your machine to be able to set up a secure communication with the other server. Similarly, you need to install any certificate that you receive in any other way, e.g on CD, by email.

Installing or Updating Certificates

There are different ways to download and update the Wirecard certificate. The simplest is to use a browser. When you are on a secure web page (starting with *https://*), like the ACM login (https://acm.wirecard.com/general/login/login.html), double-click the padlock ( ) or key ( ) symbol in the task bar at the bottom of the browser page. The certificate to this page is displayed in a popup window. Click the Install button and follow all further on-screen instruction. For example, using the Internet Explorer, select *Tools -> Internet Options -> Content*. Click *Certificates*, select the Wirecard certificate from the list of available certificates and press export to save it in a base64-encoded X.509 format (.cer file) to your hard drive.

Now you can import the certificate to the keystore of your local machine using the `keytool –import` command in your Software Development Kit (SDK). As each JVM instance can have a separate keystore, the Wirecard certificate must be installed in the keystore that is referenced by those application (e.g. Tomcat) which you use to connect to the Wirecard server. If you are connecting from more that one JVM instance, make sure that the instances and the applications reference the same keystore, or otherwise you will need to copy the certificate to each keystore separately.

PHP

If you are running PHP on your server, you have the option to activate or deactivate the SSL certificate check in your script, using the cURL option `curl_setopt` as described below. cURL is the PHP syntax extension which is required to establish any HTTPS connectivity.

It is highly recommended that your PHP script is configured to check all SSL certificates. If your PHP script does not check SSL certificates this may be because this option is deactivated. You can activate the certificate check with the follwoing set option:

```
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, TRUE);
```

Certificate Installation on MS Windows:

To install a CA root certificate on your server using Windows, use the following command:

```
curl_setopt ($ch,CURLOPT_CAINFO,"c:\\php\\tcclass3-2011.pem");
```

Certificate Installation on Unix:

To install a CA root certificate on your server using Unix, use the following command:

```
command:curl_setopt($ch,CURLOPT_CAINFO,"/opt/php/tcclass3-2011.pem");
```

where `/opt/php/` are optional subfolder names.

2.3.4 Certificate Download

You can download the latest TC TrustCenter X.509 CA certificate at:

http://www.trustcenter.de/certservices/cacerts/tcclass3-2011.pem

3 Gateway Access

Having full access to the gateway is crucial for any merchant intending to post transaction to the Wirecard system for realtime processing. The HTTPS Gateway is a flexible and secure communication interface allowing merchants to send and receive payment requests and responses between their network and the Wirecard platform. To verify if transactions and risk checks are correctly processed and card issued, Wirecard has set up the dedicated test environment. For more details please refer to:

- Payment Management Gateway (Section 3.2)
- Risk Management Gateway (Section 3.3)
- Card & Account Management Gateway (Section 3.4)
- Live System Gateway (Section 3.5)

3.1 Authentication

Only authorized users can access the HTTPS gateway interface. To gain access, a user must authenticate himself with a unique identifier (ID) and a password. The user ID is derived from the configuration number of the user's business case while the password is a randomly generated alphanumeric code. If the entered user ID and password match and are successfully validated by the system, the gateway will forward the request.

Since HTTP is a stateless protocol meaning the server does not store and remember any information about a request once it is processed, the client application needs to resend the username and password with each request by including an authorization header with basic authentication data.

Please note that the username and password must be Base64-encoded. For more details about Base64, see the *Request for Comment* RFC 2617

3.1.1 Example

A company agent with the user ID "Aladdin" and password "open sesame" would use the following garbled HTTPS header data:

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

3.2 Payment Management Gateway

Any merchant planning to integrate the Wirecard platform can test the integration on a dedicated test gateway. It is basically identical to the live HTTPS gateway with the exception that none of the submitted payment requests actually trigger a movement of moneys.

As part of the Wirecard quality assurance, merchants are requested to perform several tests on the test gateway in cooperation with the Wirecard support organization prior to connecting to the live HTTPS Gateway. This is to ensure a smooth and flawless communication and transaction data flow between the integrating company and Wirecard.

3.2.1 Test Site Access

The following are your access data to the payment test gateway. Please implement the data as shown in the examples below.

- URL for HTTPS : **https://c3-test.wirecard.com/secure/ssl-gateway**, where *c3-test.wirecard.com* is the host and *secure/ssl-gateway* the path.
- Username: **56500**
- Password: **TestXAPTER**
- Business Case Signature: **5650**
- Content-Type: **text/xml**

3.2.2 Test Site Access for Card & Account Management

For Card & Account / SCP Customers who wish to access the Card & Account Management test system are required to send the IP-address of their computer to our Technical Support at support@wirecard.com

- Username: (will be sent by email)
- Password: (will be sent by email)

3.2.3 Test Requirements

- Transfer data only in plain text.
- Use only test credit card number 4200000000000000.
- Enter “text/xml” as content type in the POST request header.
- Configure the firewall settings of your server for POST/GET message exchange.

3.2.4 PHP Example

The script used in the example below requires PHP version 4.3.0 or higher and must be compiled in OpenSSL. If you do not run the required PHP version, you have to use Curl/clib PHP extensions to set up the HTTPS services. The script will not work with Curl.

The XML transaction request shown in the follwoing PHP example is a mere placeholder and has been added for demo purposes only. Please refer to the C*ard Processing* specification for up-to-date and correct card transaction examples.

Access Data

```
<?php
$host = "c3-test.wirecard.com";
$port = 443;
$path = "/secure/ssl-gateway";
$login = "56500";
$pass = "TestXAPTER";
```

XML - Request

```
$poststring = "<?xml version='1.0' encoding='UTF-8'?>
    <WIRECARD_BXML xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
    xsi:noNamespaceSchemaLocation='wirecard.xsd'>
    <W_REQUEST>
    <W_JOB>
      <JobID>job 2</JobID>
      <BusinessCaseSignature>56500</BusinessCaseSignature>
      <FNC_CC_TRANSACTION>
           <FunctionID>WireCard Test</FunctionID>
           <CC_TRANSACTION>
               <TransactionID>2</TransactionID>
               <Amount>100</Amount>
               <Currency>USD</Currency>
               <CountryCode>US</CountryCode>
               <RECURRING_TRANSACTION>
               <Type>Single</Type>
               </RECURRING_TRANSACTION>
               <CREDIT_CARD_DATA>
                  <CreditCardNumber>4200000000000000</CreditCardNumber>
                  <CVC2>000</CVC2>
                  <ExpirationYear>2010</ExpirationYear>
                  <ExpirationMonth>01</ExpirationMonth>
                  <CardHolderName>Wirecard Test</CardHolderName>
               </CREDIT_CARD_DATA>
               <CONTACT_DATA>
                  <IPAddress>127.0.0.1</IPAddress>
               </CONTACT_DATA>
               <CORPTRUSTCENTER_DATA>
               <ADDRESS>
                  <Address1></Address1>
                  <City></City>
                  <ZipCode></ZipCode>
                  <State></State>
                  <Country></Country>
                  <Phone></Phone>
                  <Email>support@wirecard.com</Email>
               </ADDRESS>
                </CORPTRUSTCENTER_DATA>
             </CC_TRANSACTION>
          </FNC_CC_TRANSACTION>
        </W_JOB>
    </W_REQUEST>
</WIRECARD_BXML>";
```

POST request header

base64 encoding

```
$fp = fsockopen("ssl://".$host, $port, $errno, $errstr, 5);
if(!$fp){
  //error; tell us
  echo "Error: $errstr ($errno)\n";
}else{
//send the server request
 fputs($fp, "POST $path HTTP/1.0\r\n");
  fputs($fp, "Host: $host\r\n");
  fputs($fp,"Content-type: text/xml\r\n");
  fputs($fp, "Content-length: ".strlen($poststring)."\r\n");
  fputs($fp, "Authorization:
  Basic ".base64_encode($login.":".$pass."\r\n"));
  fputs($fp, "Connection: close\r\n");
  fputs($fp,"\r\n");
  fputs($fp, $poststring . "\r\n\r\n");
// prepare for reading the response
stream_set_timeout($fp,30);
         // here we save the response body - XML response from WireCard
$output = "";
// here we store the HTTP headers
$headers= "";
// temp. variable for detecting the end of HTTP headers.
$is_header = 1;
while(!feof($fp)) {
  $buffer = fgets($fp, 128);
  // fgets on SSL socket
  if ($buffer == FALSE) {
  break;
  }
  if (!$is_header) {
  $output .= $buffer;
  }
  if ($buffer == "\r\n") {
  $is_header = 0;
  }
  if ($is_header) {
  $headers .= $buffer;
  }
}
//close fp - we are done with it
fclose($fp);

// print the results in Web Browser - and convert all special
// characters (like <,>,...) to HTML entities
echo "<PRE>\n";
echo htmlentities($headers);
echo "\n";
echo htmlentities($output);
echo "</PRE>\n";
}
?>
```

3.2.5 Perl Example

The following is an example of a Perl script for connecting to the Wirecard test environment over HTTPS.

The XML transaction request shown in the follwoing Perl example is a mere placeholder and has been added for demo purposes only. Please refer to the C*ard Processing* specification for up-to-date and correct card transaction examples.

Access Data

```
#!/usr/bin/perl -w
use strict;
my $host = "c3-test.wirecard.com";
my $port = "443";
my $user = "56500";
my $passwd = "TestXAPTER";
# Create a user agent object
use LWP::UserAgent;
my $ua = new LWP::UserAgent;
# Set timeout in seconds (e.g. 30 = 30 seconds)
$ua->timeout (30);
$ua->agent("AgentName/0.1 " . $ua->agent);
# Create a request
my $req = new HTTP::Request POST => "https://$host:$port/secure/ssl-gateway";
$req->content_type('text/xml');
```

XML request

```
my $content = "<?xml version='1.0' encoding='UTF-8'?>
<WIRECARD_BXML xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='wirecard.xsd'>
<W_REQUEST>
<W_JOB>
     <JobID>job 2</JobID>
     <BusinessCaseSignature>56500</BusinessCaseSignature>
     <FNC_CC_TRANSACTION>
       <FunctionID>WireCard Test</FunctionID>
       <CC_TRANSACTION>
         <TransactionID>2</TransactionID>
         <Amount>100</Amount>
         <Currency>USD</Currency>
         <CountryCode>US</CountryCode>
         <RECURRING_TRANSACTION>
           <Type>Single</Type>
         </RECURRING_TRANSACTION>
         <CREDIT_CARD_DATA>
           <CreditCardNumber>4200000000000000</CreditCardNumber>
           <CVC2>000</CVC2>
           <ExpirationYear>2010</ExpirationYear>
           <ExpirationMonth>01</ExpirationMonth>
           <CardHolderName>Wirecard Test</CardHolderName>
         </CREDIT_CARD_DATA>
         <CONTACT_DATA>
           <IPAddress>127.0.0.1</IPAddress>
         </CONTACT_DATA>
         <CORPTRUSTCENTER_DATA>
```

XML request

```
                    <ADDRESS>
                    <Address1></Address1>
                    <City></City>
                    <ZipCode></ZipCode>
                    <State></State>
                    <Country></Country>
                    <Phone></Phone>
                    <Email>support\@wirecard.com</Email>
            </ADDRESS>
                </CORPTRUSTCENTER_DATA>
            </CC_TRANSACTION>
        </FNC_CC_TRANSACTION>
</W_JOB>
</W_REQUEST>
</WIRECARD_BXML>";
```

POST request

```
$req->content($content);
# Add credentials for basic authorization
$ua->credentials("$host:$port",'WireCard Gateway',$user, $passwd);
# Pass request to the user agent and get a response back
my $res = $ua->request($req);
# Print the request for debbuging
print "HTTP Request:\n".$req->as_string()."\n";
# Check the outcome of the response
if ($res->is_success)
{print "HTTP Response:\n".$res->as_string()."\n"
} else{
print "Content-type:text/html\n\n";
print($res->error_as_HTML."\n");
# communication failure
}
```

3.2.6 Java Example

The following example code is to demonstrate the connectivity to the Wirecard system. It presents a simple view of how an XML issuing request is posted embedded in a Java script. For details on how to set up a connection to the Wirecard gateway using Java, refer to the Jakarta HTTP Client library. If you would like to know more about how to generate XML requests, please see the JAXB2 implementation guidelines.

The Java source code presented below is intended for demonstration purposes only! It should not be used as is when establishing a connection using Java. Please refer to the Card Processing specification for complete and current card transaction examples.

Example Code:

```
import java.net.*;
import java.io.OutputStream;
import java.io.InputStream;
import java.io.ByteArrayOutputStream;

public class SampleXML_GW
{
        private URL url;
        private static final int TIMEOUT = 30000;
        public SampleXML_GW()
        {
           try
           {
               // Select the username/login you received from Wirecard
               String user = "MyUserName";
               // Select the password you received from Wirecard
               String passwd = "MyPassword";
               Authenticator.setDefault(new BasicAuthenticator(user,passwd));
               // in real applications you might need to force Basic
               auth per request.
url = new URL("https://c3-test.wirecard.com:443/secure/sslgateway");
               System.out.println(url);
       }
       catch (MalformedURLException e)
       {
               throw new RuntimeException(e);
       }
}
/**
* Set timeout in milliseconds.
* Set headers of the request (Method, Content-Type, Content-Length).
* @param content
* @param contentType
* @throws ProtocolException
*/
private void setPostRequest(HttpURLConnection urlc, byte[]
content, String contentType) throws ProtocolException
{
   urlc.setConnectTimeout(TIMEOUT);
   urlc.setRequestProperty("Content-Type", contentType);
   urlc.setRequestProperty("Content-Length", String.valueOf
   (content.getBytes().length)); urlc.setRequestMethod("POST");
}
public static void main(String[] args)
{
```

Access data

POST request

XML request

```
   String xmldata = new StringBuffer().append(" <?xml
   version='1.0' encoding='UTF-8'?> ")
.append(" <WIRECARD_BXML
          xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
          xsi:noNamespaceSchemaLocation='wirecard.xsd'> ")
.append(" <W_REQUEST> ")
.append(" <W_JOB> ")
.append(" <JobID>job 2</JobID> ")
.append(" <BusinessCaseSignature>56500</BusinessCaseSignature> ")
.append(" <FNC_CC_TRANSACTION> ")
.append(" <FunctionID>WireCard Test</FunctionID> ")
.append(" <CC_TRANSACTION> ")
.append(" <TransactionID>2</TransactionID> ")
.append(" <Amount>100</Amount> ")
.append(" <Currency>USD</Currency> ")
.append(" <CountryCode>US</CountryCode> ")
.append(" <RECURRING_TRANSACTION> ")
.append(" <Type>Single</Type> ")
.append(" </RECURRING_TRANSACTION> ")
.append(" <CREDIT_CARD_DATA> ")
.append(" <CreditCardNumber>4200000000000000</CreditCardNumber> ")
.append(" <CVC2>000</CVC2> ")
.append(" <ExpirationYear>2010</ExpirationYear> ")
.append(" <ExpirationMonth>01</ExpirationMonth> ")
.append(" <CardHolderName>Wirecard Test</CardHolderName> ")
.append(" </CREDIT_CARD_DATA> ")
.append(" <CONTACT_DATA> ")
.append(" <IPAddress>127.0.0.1</IPAddress> ")
.append(" </CONTACT_DATA> ")
.append(" <CORPTRUSTCENTER_DATA> ")
.append(" <ADDRESS> ")
.append(" <Address1></Address1> ")
.append(" <City></City> ")
.append(" <ZipCode></ZipCode> ")
.append(" <State></State> ")
.append(" <Country></Country> ")
.append(" <Phone></Phone> ")
.append(" <Email>support@wirecard.com</Email> ")
.append(" </ADDRESS> ")
.append(" </CORPTRUSTCENTER_DATA> ")
.append(" </CC_TRANSACTION> ")
.append(" </FNC_CC_TRANSACTION> ")
.append(" </W_JOB> ")
.append(" </W_REQUEST> ")
.append(" </WIRECARD_BXML> ").toString();
```

Java script

```
System.out.println ("---------------------------------------");
System.out.println(xmldata);
System.out.println("---------------------------------------");
System.out.println(new SampleXML_GW().send(xmldata.getBytes(),
"text/xml; charset=\"utf-8\""));
}
private String send(byte[] content, String contentType)
{
try
{
HttpURLConnection con = (HttpURLConnection)
url.openConnection();
if(!con.getDoOutput())
con.setDoOutput(true);
setPostRequest(con,content,contentType);
OutputStream out = con.getOutputStream();
out.write(content);
out.flush();
int responseCode = con.getResponseCode();
if (responseCode != HttpURLConnection.HTTP_OK)
System.out.println("*** SERVER RETURNED RESPONSE CODE
OTHER THAN HTTP_OK ***");
InputStream input = con.getInputStream();
ByteArrayOutputStream stream = new
ByteArrayOutputStream();
byte[] buff = new byte[1024];
int i = 0;
while ((i = input.read(buff)) > 0)
{
stream.write(buff,0,i);
}
con.disconnect();
// in case responseCode != HttpURLConnection.HTTP_OK
this might not be XML
return new String(stream.toByteArray(),"UTF-8");
}
catch (java.io.IOException e)
{
throw new RuntimeException(e);
// TODO : throw an application checked exception and hadle it
}
}
/**
 * Authentication
 */
public static class BasicAuthenticator extends Authenticator
{
protected String username;
protected String password;
public BasicAuthenticator(String username, String password)
{
this.username = username;
this.password = password;
}
protected PasswordAuthentication
getPasswordAuthentication()
{
return new PasswordAuthentication(username,
password.toCharArray());
}
}
```

3.3 Risk Management Gateway

The following are your access data to the risk management test gateway. Please bear in mind that the link, user ID and password below will connect you to a risk management test site designated to validate consumer data only (no payment). If you wish to test payment transaction requests including risk management functionality, please use the payment test site (see above).

3.3.1 Test Site Access

1. URL for HTTPS: **https://r3-test.wirecard.com/risk/verify**, where

   c3-test.wirecard.com is the host and *risk/verify* the path
2. Username: **000000315B095834**
3. Password: **WVxeNLqd8**
4. Business Case Signature: **000000315B09578C**
5. Content-Type: **text/xml**

3.4 Card & Account Management Gateway

Connecting to the Card & Account Management gateway, customers can issue one-time payment cards under the term and conditions of the Supplier and Commissions Payment (SCP) program.

3.4.1 Test Access

Customers who wish to access the Card & Account Management (CAM) system for testing purposes are required to contact Wirecard Technical Support at support@wirecard.com. To be able to access the CAM system, customers must provide Wirecard with the IP address or IP address range of those client machines posting the issuing requests to the CAM system.

Upon completion of the setup process, the customer will receive an individual username and password by email which he can now use to access the CAM system at: **https://c3.wirecard.com/issuer/client**

3.4.2 Request Examples

For examples of Card & Account Management requests messages and how to post them to the CAM system, please see the customer specification *Card & Account Management*.

3.5 Live System Gateway

The Live Gateway is available to merchants who have been registered in the Wirecard system and who have successfully connected to the Wirecard test environment several times.

Merchants who meet these requirements will be sent the access data to the live Wirecard processing environment.

If you have any questions, please contact your sales representative or our technical support at support@wirecard.com.

www.wirecard.com