First Data is now fiserv.

Web Service API

Integration Guide

Version 2020-4 (IPG)

Integration Guide Web Service API

Version 2020-4 (IPG)

Contents

Getting Support

There are different manuals available for First Data's eCommerce solutions. This Integration Guide will be the most helpful for integrating the Web Service API for usage with our distribution channels in Europe, Asia, Australia, Latin America and Africa.

For information about settings, customisation, reports and how to process transactions manually (by keying in the information) please refer to the User Guide Virtual Terminal.

If you have read the documentation and cannot find the answer to your question, please contact your local support team.

Information for merchants with existing Web Service API integration using the Java client to connect to the web service:

- The implementation of the IPGApiClient and some signatures of methods of this class have been changed due to a change from appache http client 3.x to appache http client 4.x
- Transaction classes and transaction factory have not been changed
- If the previous IPGApiClient works in your environment, you can continue to use it.

1 Introduction

The Web Service API is an Application Programming Interface which allows you to connect your application with the First Data Gateway. In this way, your application is able to submit payment transactions without any user interference.

Please note that if you store or process cardholder data within your own application, you must ensure that your system components are compliant with the Data Security Standard of the Payment Card Industry (PCI DSS). Depending on your transaction volume, an assessment by a Qualified Security Assessor may be mandatory to declare your compliance status.

From a technical point of view, this API is a Web Service offering one remote operation for performing transactions. The three core advantages of this design can be summarized as follows:

- **Platform independence:** Communicating with the Web Service API means that your application must only be capable of sending and receiving SOAP messages. There are no requirements tied to a specific platform, since the Web Service technology builds on a set of open standards. In short, you are free to choose any technology you want (e.g. J2EE, .NET, PHP, ASP, etc.) for making your application capable of communicating with the Web Service API.
- **Easy integration:** Communicating with a Web Service is simple – your application has to build a SOAP request message encoding your transaction, send it via HTTPS to the Web Service and wait for a SOAP response message which contains your transaction's status report. Since SOAP and HTTP are designed to be lightweight protocols, building requests and responses becomes a straightforward task. Furthermore, you rarely have to do this manually, since there are plenty of libraries available in almost every technology. In general, building a SOAP request and handling the response is reduced to a few lines of code.
- **Security:** All communication between your application and the Web Service API is TLS-encrypted. This is established by your application holding a client certificate which identifies it uniquely at the Web Service. In the same way, the First Data Gateway holds a server certificate which your application may check for making sure that it speaks to our Web Service API. Finally, your application has to do a basic authentication (user name / password) before being allowed to communicate with the Web Service. In this way, the users who are authorised to communicate with the First Data Gateway are identified. These two security mechanisms guarantee that the transaction data sent to First Data both stays private and is identified as transaction data that your application has committed and belongs to no one else.

While this represents just a short summary of the Web Service API's features, the focus of this guide lies on integrating the First Data Gateway functionality into your application. A detailed description, explaining how this is done step by step, is presented in this guide.

2 Artefacts You Need

Supporting a high degree of security requires several artefacts you need for communicating securely with the Web Service API. Since these artefacts are referenced throughout the remainder of this guide, the following checklist shall provide an overview enabling you to make sure that you have received the whole set when registering your application for the First Data Gateway:

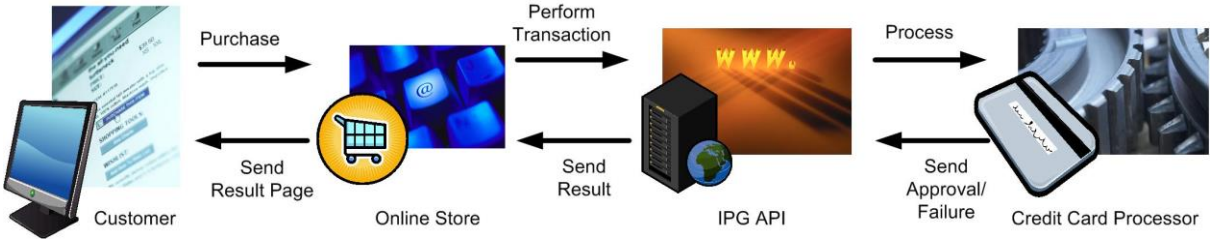
- *Store ID:* Your store ID (e.g. 10012345678) which is required for the basic authentication.
- *User ID:* The user ID denoting the user who is allowed to access the Web Service API, e.g. 1. Again, this is required for the basic authentication.
- *Password:* The password required for the basic authentication.
- *Client Certificate p12 File:* The client certificate and private key stored in a p12 file having the naming scheme WS*storeID*._.*userID*.p12, e.g. in case of the above store ID / user ID examples, this would be WS101._.007.p12. This file is used for authenticating the client at the Gateway. For connecting with Java you need a ks-File, e.g.: WS10012345678._.1.ks.
- *Client Certificate Installation Password:* The password which is required to access the p12 file (containing the client certificate and private key file).
- *Client Certificate Private Key:* The private key of the client certificate stored in a key file having the naming scheme WS*storeID*._.*userID*.key, e.g. in case of the above store ID / user ID examples, this would be WS10012345678._.1.key. Some tools which support you in setting up your application for using the Web Service API require the private key in this format when doing the client authentication at the Gateway.
- *Client Certificate Private Key Password:* This password protects the private key of the client certificate. This password is needed to access the private key file ("*Client Certificate Private Key*") It follows the naming scheme ckp_*creationTimestamp*. For instance, this might be ckp_1193927132.
- *Client Certificate PEM File:* The list of client certificates stored in a PEM file having the naming scheme WS*storeID*._.*userID*.pem, e.g. in case of the above store ID / user ID examples, this would be WS10012345678._.1.pem. Some tools which support you in setting up your application for using the Gateway require this file instead of the p12 file described above.
- *Trust Anchor as concatenated PEM File (tlstrust.pem):* The file contains a list of client certificates you should trust to establish a trusted connection to the running the Web Service API. A Concatenated list of PEM-formatted certificates allow easy installation for Apache Webservers or PHP.*Trust Anchor as Java Keystore File (truststore.jsk):* The file contains a list of client certificates you should trust to establish a trusted connection to the server running the Web Service API. This format is can easily support Java-based integrations
- *Trust Anchor as PKCS#7 File (tlstrust.p7b):* This file contains a list of CA certificates you should trust to establish a trusted connection to the server running the Web Service API. PKCS#7 Files allow the easy installation of multiple certificate for example within Microsoft Windows.

If you should be planning to handle multiple Store IDs through your integration, we can issue a special API User and Client Certificate for you that you can use across all your Stores. When you submit transactions from that API user, you do not need to vary the API User Name as the API User is the same for all your Stores. You will need to include the Store ID in each transaction request in that case.

3 How the API works

The following section describes the API by means of a credit card transaction. The process for other payment types is similar.

In most cases, a customer starts the overall communication process by buying goods or services with her credit card **in your online store**. Following this, your store sends a credit card transaction (mostly in order to capture the customer's funds) via the Web Service API. Having received the transaction, the First Data Gateway forwards it to the credit card processor for authorisation. Based on the result, an approval or error is returned to your online store. This means that all communication and processing details are covered by the First Data Gateway and you only have to know how to communicate with this Web Service.



The Web Service Standard defines such an interface by using the Web Service Definition Language (WSDL). A WSDL file defining the Web Service API for the First Data Gateway can be found at:

https://test.ipg-online.com/ipgapi/services/order.wsdl

Note that you will have to supply your client certificate and your credentials, when viewing or requesting the file e.g. in a Web browser. For instance, in case you want to view the WSDL file in Microsoft's Internet Explorer running on Microsoft Windows XP, you first have to install your client certificate, and then call the above URL. This is done by executing the following steps:

1. Open the folder in which you have saved your client certificate p12 file.
2. Double-click the client certificate p12 file.
3. Click *Next*. Check the file name (which should be already set to the path of your client certificate p12 file) and click *Next*.
4. Provide the Client Certificate Installation Password and click *Next*.
5. Choose the option *Automatically select the certificate store based on the type of certificate* and click *Next*. This will place the certificate in your personal certificate store (more precisely in the local Windows user's personal certificate store).
6. Check the displayed settings and click *Finish*. Your client certificate is now installed.
7. Now, open a Microsoft Internet Explorer window and provide the above URL in the address field.
8. After requesting the URL, the server will ask your browser to supply the client certificate to making sure that it is talking to your application correctly. Since you have installed the certificate in the previous steps, it is transferred to the server without prompting you for any input (i.e. you will not notice this process). Then, the First Data Gateway sends its server certificate (identifying it uniquely) to you. This certificate is verified against pre-installed certificates of your browser. Again, this is done automatically without prompting you for any input. Now, a secure connection is established and all data transferred between your application and the Web Service API is TLS-encrypted. Please note, that only TLS secured communication over standard HTTPS TCP port 443 is accepted.
9. Next, you will be prompted to supply your credentials for authorisation. As user name you have to provide your store ID and user ID encoded in the format WS*storeID*._.*userID* (unless you manage multiple Stores through your integration). For instance, assuming your store ID is 101, your user ID 007, and your password myPW, you have to supply WS101._.007 in the user name field and myPW in the password field. Note that your credentials are encrypted before being passed to the server due to the TLS connection established in the steps above. Then, click *OK*.

10. The Web Service API WSDL file is displayed.

In short, the WSDL file defines the operations offered by the Web Service, their input and return parameters, and how these operations can be invoked. In case of the First Data Gateway Web Service API, it defines only one operation (IPGApiOrder) callable by sending a SOAP HTTP request to the following URL:

https://test.ipg-online.com/ipgapi/services

This operation takes an XML-encoded transaction as input and returns an XML-encoded response. Note that it is not necessary to understand how the WSDL file is composed for using the First Data Gateway. The following chapters will guide you in setting up your store for building and performing custom credit card transactions.

However, in case you are using third-party tools supporting you in setting up your store for accessing the Web Service API, you might have to supply the URL where the WSDL file can be found. In a similar way as described above, you have to tell your Web Service tool, that the communication is TLS-enabled, requiring you to provide your client certificate and accept the server certificate as a trusted one. Furthermore, you have to supply your credentials. How all is done heavily depends on your Web Service tool. Hence, check the tool's documentation for details.

4 Sending transactions to the gateway

The purpose of this chapter is to give you a basic understanding of the steps to be taken when committing transactions to the First Data Gateway. It describes what happens if a customer pays with her credit card in an online store using the Web Service API for committing transactions.

- The customer clicks on the *Pay* button in the online store.
- The online store displays a form asking the customer to provide her credit card number and the expiry month and year.
- The customer types in these three fields and submits the data to the online store (i. e. purchases the goods).
- The online store receives the data and builds an XML document encoding a *Sale* transaction which includes the data provided by the customer and the total amount to be paid by the customer.
- After building the XML *Sale* transaction, the online store wraps it in a SOAP message which describes the Web Service operation to be called with the transaction XML being passed as a parameter.
- Having built the SOAP message, the online store prepares it for being transferred over the Internet by packing its content into an HTTPS POST request. Furthermore, the store sets the HTTP headers, especially its credentials (note that the credentials are the same as the ones you have to provide for viewing the WSDL file).
- Now, the store establishes an TLS connection by providing the client and server certificate. Please note, that only TLS secured communication over standard HTTPS TCP port 443 is accepted.
- Then, the online store sends the HTTPS request to the Web Service API and waits for an HTTP response.
- The Web Service API receives the HTTPS request and parses out the authorization information provided by the store in the HTTP headers.
- Having authorized the store to use the First Data Gateway, the SOAP message contained in the HTTP request body is parsed out. This triggers the Web Service operation handling the transaction processing to run.

- The Gateway then performs the transaction processing, builds an XML response document, wraps it in a SOAP message, and sends this SOAP message back to the client in the body of an HTTPS response.
- Receiving this HTTPS response wakes up the store which reads out the SOAP message and response XML document being part of it.
- Depending on the data contained in the XML response document an approval page is sent back to the customer in case of a successful transaction, otherwise an error page is returned.
- The approval or error page is displayed.

While this example describes the case of a *Sale* transaction, other transactions basically follow the same process.

Summarising the scenario, your application has to perform the following steps in order to commit credit card transactions and analyze the result:

- Build an XML document encoding your transactions
- Wrap that XML document in a SOAP request message
- Build an HTTPS POST request with the information identifying your store provided in the HTTP header and the SOAP request message in the body
- Establish an TLS connection between your application and the Web Service API
- Send the HTTPS POST request to the First Data Gateway and receive the response
- Read the SOAP response message out of the HTTPS response body
- Analyse the XML response document contained in the SOAP response message

These seven steps are described in the following chapters. They guide you through the process of setting up your application for performing custom credit card transactions.

5 Building Transactions in XML]

This chapter describes how the different transaction types can be built in XML. As the above example scenario has outlined, a transaction is first encoded in an XML document which is then wrapped as payload in a SOAP message. That means the XML-encoded transaction represents the parameter passed to the Web Service API operation.

Note that there exists a variety of Web Service tools supporting you in the generation of client stubs which might free you of the necessity to deal with raw XML. However, a basic understanding of the XML format is crucial in order to build correct transactions regardless of the available tool support. Hence, it is recommended to become familiar with the XML format used by the Web Service API for encoding transactions.

5.1 Credit/Debit Card transactions

Regardless of the transaction type, the basic XML document structure of a credit/debit card transaction is as follows:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>...</v1:CreditCardTxType>
            <v1:CreditCardData>...</v1:CreditCardData>
            <v1:Payment>...</v1:Payment>
            <v1:TransactionDetails>...</v1:TransactionDetails>
            <v1:Billing>...</v1:Billing>
            <v1:Shipping>...</v1:Shipping>
```

```
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

The element CreditCardDataTXType is mandatory for all credit card transactions. The other elements depend on the transaction type. The transaction content is type-specific. The elements in XML structure must be kept in the same order as shown in examples, otherwise the *OrderRequest* will fail.

For XML-tags related to Card Present transactions with a chip reader and PIN entry device please refer to the xsd's in the Appendix of this document.

5.1.1 Sale

The following XML document represents an example of a *Sale* transaction using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>sale</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111*******1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
            <v1:Payment>
                  <v1:ChargeTotal>19.95</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

The following XML document represents an example of a Sale transaction for API users handling multiple Store IDs:

```
<ipgapi:IPGApiOrderRequest
            xmlns:ipgapi='http://ipg-online.com/ipgapi/schemas/ipgapi'
            xmlns:v1='http://ipg-online.com/ipgapi/schemas/v1'>
          <v1:Transaction>
            <v1:CreditCardTxType>
              <v1:StoreId>1234567890</v1:StoreId>
              <v1:Type>sale</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
              <v1:CardNumber>4111******1111</v1:CardNumber>
              <v1:ExpMonth>12</v1:ExpMonth>
              <v1:ExpYear>20</v1:ExpYear>
              <v1:CardCodeValue>XXX</v1:CardCodeValue>
            </v1:CreditCardData>
            <v1:Payment>
              <v1:ChargeTotal>15.00</v1:ChargeTotal>
              <v1:Currency>978</v1:Currency>
            </v1:Payment>
```

```
            <v1:TransactionDetails>
              <v1:OrderId>12-34-56</v1:OrderId>
              <v1:MerchantTransactionId>AB500500</v1:MerchantTransactionId>
              <v1:TransactionOrigin>ECI</v1:TransactionOrigin>
              <v1:DynamicMerchantName>MyWebsite</v1:DynamicMerchantName>
            </v1:TransactionDetails>
           <v1:Billing>
              <v1:Zip>0001</v1:Zip>
            </v1:Billing>
            </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

5.1.2 Pre-Authorisation

The following XML document represents an example of a *PreAuth* transaction using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>preAuth</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111*******1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
            <v1:Payment>
                  <v1:ChargeTotal>100.00</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

5.1.3 Post-Authorisation

The following XML document represents an example of a *PostAuth* transaction using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>postAuth</v1:Type>
            </v1:CreditCardTxType>
            <v1:Payment>
                  <v1:ChargeTotal>59.00</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
            <v1:TransactionDetails>
                  <v1:OrderId>
                        703d2723-99b6-4559-8c6d-797488e8977
                  </v1:OrderId>
            </v1:TransactionDetails>
```

```
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

In case your system is not aware of the payment method that has been used for the original Pre-Authorisation transaction, the Post-Authorisation can be performed using any TxType which supports Post-Authorisations. The gateway will then select the correct payment method based on the referenced Order ID.

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

5.1.4 ForceTicket

The following XML document represents an example of a *ForceTicket* transaction using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>forceTicket</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111*******1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
            <v1:Payment>
                  <v1:ChargeTotal>59.00</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
            <v1:TransactionDetails>
                  <v1:ReferenceNumber>123456</v1:ReferenceNumber>
            </v1:TransactionDetails>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

5.1.5 Return

The following XML document represents an example of a *Return* transaction using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>return</v1:Type>
            </v1:CreditCardTxType>
            <v1:Payment>
                  <v1:ChargeTotal>19.00</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
            <v1:TransactionDetails>
                  <v1:OrderId>
                        62e3b5df-2911-4e89-8356-1e49302b1807
```

```
                <v1:OrderId>
            </v1:TransactionDetails>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

In case your system is not aware of the payment method that has been used for the original transaction, the Return can be performed using any TxType which supports Returns. The gateway will then select the correct payment method based on the referenced Order ID.

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

5.1.6 Credit

Please note that Credit is a transaction type that requires special user permissions.

The following XML document represents an example of a *Credit* transaction using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>credit</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111*******1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
            <v1:Payment>
                  <v1:ChargeTotal>50.00</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

5.1.7 Void

The following XML document represents an example of a *Void* transaction using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>void</v1:Type>
            </v1:CreditCardTxType>
            <v1:TransactionDetails>
                  <v1:IpgTransactionId>1234567890</v1:IpgTransactionId>
            </v1:TransactionDetails>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

For referencing to the transaction that shall be voided, this example uses the parameter IpgTransactionId. If you have assigned a transaction ID (**MerchantTransactionId**) in the original transaction, you can alternatively submit this ID as **ReferencedMerchantTransactionId** instead..
In case your system is not aware of the payment method that has been used for the original transaction, the Void can be performed using any TxType which supports Voids. The gateway will then select the correct payment method based on the referenced Order ID and TDate.

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

5.1.8 Recurring Sale (Merchant-triggered)

The following XML document represents an example of a first *Sale* transaction of a series of recurring payments:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>sale</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111*******1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
      <ns2:recurringType>FIRST</ns2:recurringType>
            <v1:Payment>
                  <v1:ChargeTotal>19.95</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

Subsequent transactions in a series need to be flagged like this:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>sale</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111*******1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
      <ns2:recurringType>REPEAT</ns2:recurringType>
            <v1:Payment>
                  <v1:ChargeTotal>19.95</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

Please see chapter Recurring Payments (Scheduler) for the alternative option to let the gateway automatically trigger recurring transactions.

5.2 MasterPass™

MasterPass is MasterCard's digital wallet solution that allows consumers to store payment, billing and shipping details for a fast, convenient, and secure checkout experience at a merchant's website.

The easiest way to make use of MasterPass is to use our Connect solution. However in case you prefer to integrate directly to MasterPass and manage the authentication and wallet process yourself, you can submit the *Wallet ID* and *Wallet Type* within your request for a credit card transaction.

Please see details for a direct integration with MasterPass here:
https://developer.mastercard.com/portal/display/api/MasterPass+-+Merchant+Checkout+Services+-+Documentation

The following XML document represents an example of a *Sale* transaction with MasterPass using the minimum set of elements:

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
      <v1:Transaction>
            <v1:CreditCardTxType>
                  <v1:Type>sale</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111********1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
      <v1:Wallet>
            <v1:WalletType>MASTERPASS</ns2:WalletType>
            <v1:WalletID>101</ns2:WalletID>
      </v1:Wallet>
      <v1:Payment>
            <v1:ChargeTotal>19.95</v1:ChargeTotal>
            <v1:Currency>978</v1:Currency>
      </v1:Payment>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

5.3 PayPal

5.3.1 Post-Authorisation Payment Transaction

After a payment authorisation for PayPal has been submitted via the Gateway's Connect interface, the Web Service API can be used to perform post-authorisation payments.

The following XML document represents an example of a *PostAuth* transaction using the minimum set of elements:

```
<ns5:IPGApiOrderRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/v1">
      <ns4:Transaction>
            <ns4:PayPalTxType>
                  <ns4:Type>postAuth</ns4:Type>
            </ns4:PayPalTxType>
            <ns4:Payment>
                  <ns4:ChargeTotal>1</ns4:ChargeTotal>
                  <ns4:Currency>EUR</ns4:Currency>
            </ns4:Payment>
```

```
            <ns4:TransactionDetails>
                  <ns4:OrderId>
                        C-32121f4d-852f-4f48-8095-8585b917c079
                  </ns4:OrderId>
            </ns4:TransactionDetails>
      </ns4:Transaction>
</ns5:IPGApiOrderRequest>
```

See chapter **XML-Tag overview** for a detailed description of all elements used in the above example as well as further optional elements.

5.3.2 Recurring Payment Transaction

The recurring payments for PayPal can be executed via the Connect solution. You have to submit a *SALE* transaction request with the corresponding parameters to install the recurring payments. The first transaction is always conducted immediately along with the request.

The subsequent transactions are executed by the Gateway's scheduler, via the API Web Service, as defined during the initial *SALE* transaction with the instalation.

5.3.3 Return

The following XML document represents an example of a *Return* transaction using the minimum set of elements:

```
<ns5:IPGApiOrderRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/v1">
      <ns4:Transaction>
            <ns4:PayPalTxType>
                  <ns4:Type>return</ns4:Type>
            </ns4:PayPalTxType>
            <ns4:Payment>
                  <ns4:ChargeTotal>0.4</ns4:ChargeTotal>
                  <ns4:Currency>EUR</ns4:Currency>
            </ns4:Payment>
            <ns4:TransactionDetails>
                  <ns4:OrderId>
                        C-32121f4d-852f-4f48-8095-8585b917c079
                  </ns4:OrderId>
            </ns4:TransactionDetails>
      </ns4:Transaction>
</ns5:IPGApiOrderRequest>
```

In case your system is not aware of the payment method that has been used for the original transaction, the Return can be performed using any TxType which supports Returns. The gateway will then select the correct payment method based on the referenced Order ID.

5.3.4 Void

The following XML document represents an example of a *Void* transaction using the minimum set of elements:

```
<ns5:IPGApiOrderRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/v1">
      <ns4:Transaction>
            <ns4:PayPalTxType>
```

```
            <ns4:Type>void</ns4:Type>
        </ns4:PayPalTxType>
        <ns4:TransactionDetails>
            <v1:IpgTransactionId>1234567890</v1:IpgTransactionId>
        </ns4:TransactionDetails>
    </ns4:Transaction>
</ns5:IPGApiOrderRequest>
```

For referencing to the transaction that shall be voided, this example uses the parameter IpgTransactionId. If you have assigned a transaction ID (**MerchantTransactionId**) in the original transaction, you can alternatively submit this ID as **ReferencedMerchantTransactionId** instead of sending a TDate.
In case your system is not aware of the payment method that has been used for the original transaction, the Void can be performed using any TxType which supports Voids. The gateway will then select the correct payment method based on the referenced Transaction ID.

5.3.5 Credit

Please note that Credit is a transaction type that requires special user permissions.

The following XML document represents an example of a *Credit* transaction using the minimum set of elements:

```
<ns5:IPGApiOrderRequest
        xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
        xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
        xmlns:ns4="http://ipg-online.com/ipgapi/schemas/v1">
    <ns4:Transaction>
        <ns4:PayPalTxType>
            <ns4:Type>credit</ns4:Type>
        </ns4:PayPalTxType>
        <ns4:Payment>
            <ns4:ChargeTotal>1</ns4:ChargeTotal>
            <ns4:Currency>EUR</ns4:Currency>
        </ns4:Payment>
        <ns4:Billing>
            <ns4:Email>x@y.zz</ns4:Email>
        </ns4:Billing>
    </ns4:Transaction>
</ns5:IPGApiOrderRequest>
```

Unlike with other payment methods, PayPal transactions contain no payment data like a card number. Therefore this transaction requires the resgistered email address of the recipient of the payment. This email address must be submitted in the field ns4:Billing/ns4:Email.

5.4 Generic Transaction Type for Voids and Returns

The Tag *SubsequentTransaction* allows you to submit Voids and Refunds independently from which payment method had been used for the original payment transaction.

You can initiate such transactions by referencing to a previous transaction using one of the following options:

- IPG Transaction ID
- Merchant Transaction ID

The following XML document represents an example of a **Void** transaction using the minimum set of elements for IPG Transaction ID:

```
<ns5:IPGApiOrderRequest
```

```
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns2:SubsequentTransaction>
            <ns2:IpgTransactionId>1234567890</ns2:IpgTransactionId>
                  <ns2:Options>
                        <ns2:StoreId>120995000</ns2:StoreId>
                  </ns2:Options>
                  <ns2:TransactionType>VOID</ns2:TransactionType>
            </ns2:SubsequentTransaction>
      </ns5:IPGApiOrderRequest>
```

The following XML document represents an example of a **Void** transaction using the minimum set of elements for Merchant Transaction ID:

```
<ns5:IPGApiOrderRequest
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
            <ns2:SubsequentTransaction>
                  <ns2:ReferencedMerchantTransactionId>ITID-
000380</ns2:ReferencedMerchantTransactionId>
                  <ns2:Options>
                        <ns2:StoreId>44036000750</ns2:StoreId>
                  </ns2:Options>
                  <ns2:TransactionType>VOID</ns2:TransactionType>
            </ns2:SubsequentTransaction>
      </ns5:IPGApiOrderRequest>
```

The following XML document represents an example of a **Return** transaction using the minimum set of elements for IPG Transaction ID:

```
<ns5:IPGApiOrderRequest
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
            <ns2:SubsequentTransaction>
                  <ns2:IpgTransactionId>123456789</ns2:IpgTransactionId>
                  <ns2:Options>
                        <ns2:StoreId>120995000</ns2:StoreId>
                  </ns2:Options>
                  <ns2:TransactionType>RETURN</ns2:TransactionType>
                  <ns2:Payment>
                        <ns2:ChargeTotal>1.00</ns2:ChargeTotal>
                        <ns2:Currency>978</ns2:Currency>
                  </ns2:Payment>
            </ns2:SubsequentTransaction>
      </ns5:IPGApiOrderRequest>
```

The following XML document represents an example of a **Return** transaction using the minimum set of elements for Merchant Transaction ID:

```
<ns5:IPGApiOrderRequest
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
            <ns2:SubsequentTransaction>
                  <ns2:ReferencedMerchantTransactionId>ITID-
000380</ns2:ReferencedMerchantTransactionId>
                  <ns2:Options>
```

```
                        <ns2:StoreId>44036000750</ns2:StoreId>
                    </ns2:Options>
                    <ns2:TransactionType>RETURN</ns2:TransactionType>
                    <ns2:Payment>
                        <ns2:ChargeTotal>1.00</ns2:ChargeTotal>
                        <ns2:Currency>978</ns2:Currency>
                    </ns2:Payment>
                </ns2:SubsequentTransaction>
        </ns5:IPGApiOrderRequest>
```

6 Additional Web Service actions

6.1 Initiate Clearing

Clearing for transactions can be initiated via the Web Service similar to a payment transaction:

```
<ipgapi:IPGApiActionRequest
            xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
        <a1:Action>
            </a1:InitiateClearing>
        </a1:Action>
</ipgapi:IPGApiActionRequest>
```

Clearing will will be executed directly. If clearing was not successful for at least one terminal, the gateway will send “false” in the response.

```
<ipgapi:IPGApiActionResponse
        xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
        xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1"

        xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1">

        <ipgapi:successfully>false</ipgapi:successfully>

</ipgapi:IPGApiActionResponse>
```

6.2 Inquiry Order

The action *InquiryOrder* allows you to get details about previously processed transactions of a specific order. You therefore need to submit the corresponding Order ID:

```
<ns4:IPGApiActionRequest
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
        <ns2:Action>
            <ns2:InquiryOrder>
                <ns2:OrderId>
                    b5b7fb49-3310-4212-9103-5da8bd026600
                </ns2:OrderId>
            </ns2:InquiryOrder>
        </ns2:Action>
</ns4:IPGApiActionRequest>
```

The result contains information about all transactions belonging to the corresponding Order ID:

```
<?xml version="1.0" encoding="UTF-8"?><ipgapi:IPGApiActionResponse
xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
```

```
xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
    <ipgapi:successfully>true</ipgapi:successfully>
    <ipgapi:OrderId>b5b7fb49-3310-4212-9103-5da8bd026600</ipgapi:OrderId>
    <v1:Billing/>
    <v1:Shipping/>
    <a1:TransactionValues>
        <v1:CreditCardTxType>
            <v1:Type>sale</v1:Type>
        </v1:CreditCardTxType>
        <v1:CreditCardData>
            <v1:CardNumber>4501*****8992</v1:CardNumber>
            <v1:ExpMonth>11</v1:ExpMonth>
            <v1:ExpYear>17</v1:ExpYear>
            <v1:Brand>VISA</v1:Brand>
        </v1:CreditCardData>
        <v1:Payment>
            <v1:ChargeTotal>350.05</v1:ChargeTotal>
            <v1:Currency>826</v1:Currency>
        </v1:Payment>
        <v1:TransactionDetails>
            <v1:Comments>AS400</v1:Comments>
            <v1:InvoiceNumber>551294633441</v1:InvoiceNumber>
            <v1:OrderId>b5b7fb49-3310-4212-9103-5da8bd026600</v1:OrderId>
            <v1:Ip>194.127.72.6</v1:Ip>
            <v1:TDate>1450091856</v1:TDate>
            <v1:TransactionOrigin>MOTO</v1:TransactionOrigin>
        </v1:TransactionDetails>
        <ipgapi:IPGApiOrderResponse>

<ipgapi:ApprovalCode>Y:015722:0795783078:PPXM:2062</ipgapi:ApprovalCode>
            <ipgapi:AVSResponse>PPX</ipgapi:AVSResponse>
            <ipgapi:Brand>VISA</ipgapi:Brand>
            <ipgapi:Country>GBR</ipgapi:Country>
            <ipgapi:OrderId> b5b7fb49-3310-4212-9103-
5da8bd026600</ipgapi:OrderId>
            <ipgapi:PaymentType>CREDITCARD</ipgapi:PaymentType>
<ipgapi:ProcessorApprovalCode>015722</ipgapi:ProcessorApprovalCode>
            <ipgapi:ProcessorCCVResponse>M</ipgapi:ProcessorCCVResponse>
            <ipgapi:ReferencedTDate>1450091856</ipgapi:ReferencedTDate>
            <ipgapi:TDate>1450091856</ipgapi:TDate>
            <ipgapi:TDateFormatted>2015.12.14 12:17:36
(CET)</ipgapi:TDateFormatted>
            <ipgapi:TerminalID>80250837</ipgapi:TerminalID>
        </ipgapi:IPGApiOrderResponse>
        <a1:TraceNumber>2062</a1:TraceNumber>
        <a1:TransactionState>CAPTURED</a1:TransactionState>
        <a1:SubmissionComponent>CONNECT</a1:SubmissionComponent>
    </a1:TransactionValues>
</ipgapi:IPGApiActionResponse>
```

6.3 Inquiry Transaction

The action *InquiryTransaction* allows you to get details about a previously processed transaction. You therefore need to either submit the *merchantTransactionId* if you have assigned one or alternatively the *ipgTransactionId:*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ipg="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
```

```
   <soapenv:Header/>
   <soapenv:Body>
      <ipg:IPGApiActionRequest>
         <a1:Action>
            <a1:InquiryTransaction>
               <!--Optional:-->
               <a1:StoreId>12072591</a1:StoreId>
               <!--You have a CHOICE of the next 3 items at this level-->
               <a1:OrderId>C-38fd1bcd-1d67-4248-b9d5-
d30376d92163</a1:OrderId>
               <a1:TDate>1453814407</a1:TDate>
              </a1:InquiryTransaction>
         </a1:Action>
      </ipg:IPGApiActionRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

The response contains the same elements as in the Inquiry Order example above.

6.4 Recurring Payments (Scheduler)

The action *RecurringPayment* allows you to install, modify or cancel periodic payments in a way that subsequent transactions will automatically be triggered by the gateway.

For every recurring transaction, the gateway can send a server-to-server transaction notification to a defined Notification URL. Please contact your local support team to get your URL registered for these notifications.

6.4.1 Install

The following example shows how to install a monthly credit card payment with 12 executions (*InstallmentCount*) in 2011 starting on 15 January 2011.
Please note that the *RecurringStartDate* will be interpreted based on the timezone Europe/Berlin.

```
<ns4:IPGApiActionRequest
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns2:Action>
            <ns2:RecurringPayment>
                  <ns2:Function>install</ns2:Function>
                  <ns2:RecurringPaymentInformation>
                        <ns2:RecurringStartDate>
                              20110115
                        </ns2:RecurringStartDate>
                        <ns2:InstallmentCount>12</ns2:InstallmentCount>
                        <ns2:InstallmentFrequency>
                              1
                        </ns2:InstallmentFrequency>
                        <ns2:InstallmentPeriod>
                              month
                        </ns2:InstallmentPeriod>
                  </ns2:RecurringPaymentInformation>
                  <ns2:CreditCardData>
                        <ns3:CardNumber>4035……4977</ns3:CardNumber>
                        <ns3:ExpMonth>12</ns3:ExpMonth>
                        <ns3:ExpYear>12</ns3:ExpYear>
                        <ns3:CardCodeValue>XXX</ns3:CardCodeValue>
                  </ns2:CreditCardData>
                  <ns3:Payment>
```

```
                        <ns3:ChargeTotal>1</ns3:ChargeTotal>
                        <ns3:Currency>978</ns3:Currency>
                  </ns3:Payment>
            </ns2:RecurringPayment>
      </ns2:Action>
</ns4:IPGApiActionRequest>
```

If you set the *RecurringStartDate* to the actual date, the first payment will immediately be initiated. In this case, the payment data will only be stored for future payments if this first payment was succesful/approved.
A start date in the past is not allowed.

The default value for *TransactionOrigin* is 'ECI'. If you want to change this value, you can submit a different TransactionOrigin tag in the RecurringPayment tag.

6.4.2 Modify

Modifications of an existing Recurring Payment can be initiated using the *Order ID*:

```
<ns4:IPGApiActionRequest
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns2:Action>
            <ns2:RecurringPayment>
                  <ns2:Function>modify</ns2:Function>
                  <ns2:OrderId>
                        e368a525-173f-4f56-9ae2-beb4023a6993
                  </ns2:OrderId>
                  <ns2:RecurringPaymentInformation>
                        <ns2:InstallmentCount>999</ns2:InstallmentCount>
                  </ns2:RecurringPaymentInformation>
            </ns2:RecurringPayment>
      </ns2:Action>
</ns4:IPGApiActionRequest>
```

You only need to include the elements that need to be changed. If you change the credit card number, it is also required to include the expiry date, otherwise you can change the expiry date without specifying the credit card number. If you want to change the amount, you also need to include the currency.

6.4.3 Cancel

To cancel a Recurring Payment, you also use the *Order ID*:

```
<ns4:IPGApiActionRequest
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns2:Action>
            <ns2:RecurringPayment>
                  <ns2:Function>cancel</ns2:Function>
                  <ns2:OrderId>
                        e368a525-173f-4f56-9ae2-beb4023a6993
                  </ns2:OrderId>
            </ns2:RecurringPayment>
      </ns2:Action>
</ns4:IPGApiActionRequest>
```

6.4.4 Test Recurring Payments in test environment

The test system allows you to manually initiate a scheduled payment to test this functionality. This function will not work in live mode.

```
<ns4:IPGApiActionRequest
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns2:Action>
            <ns2:RecurringPayment>
                  <ns2:Function>
                        perform only in test environment
                  </ns2:Function>
                  <ns2:OrderId>
                        A-eab002b9-5889-4082-9cc9-5bc06b8eaa61
                  </ns2:OrderId>
            </ns2:RecurringPayment>
      </ns2:Action>
</ns4:IPGApiActionRequest>
```

6.4.5 Response

The response for a successful instalment, modification or cancellation contains the value ***true*** for the parameter *<ns4:successfully>*:

```
<ns4:IPGApiActionResponse
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns4:successfully>true</ns4:successfully>
      <ns4:OrderId>e368a525-173f-4f56-9ae2-beb4023a6993</ns4:OrderId>
</ns4:IPGApiActionResponse>
```

6.5 External transaction status

Some payment endpoints do not send the final result of a payment transaction within their response. In such cases the Gateway returns an approval code that starts with a question mark (?:…).The action GetExternalTransactionState allows you to request updates on the state of such transactions. You can use *OrderID* + *TDate*, *MerchantTransactionId* or *IpgTransactionId* to reference to a transaction.

6.5.1 Trigger email notifications

The action SendEMailNotification triggers an email notification for a given transaction. The email will be created with the email template that has been configured for your Store.

See the User Guide Virtual Terminal & Online Portal for more information on transaction notifications by email.

```
<ns5:IPGApiActionRequest
            xmlns:ns5=http://ipg-online.com/ipgapi/schemas/ipgapi
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">

      <ns2:Action>
            <ns2:SendEMailNotification>
                  <ns2:OrderId>0/8/15</ns2:OrderId>
                  <ns2:TDate>1250599046</ns2:TDate>
```

```
            </ns2:SendEMailNotification>
      </ns2:Action>
</ns5:IPGApiActionRequest>
```

If the optional parameter Email is not set, the email address of the customer stored with the transaction will be used.

6.5.2 Card Information Inquiry

The function *InquiryCardInformation* allows you to check the brand and function of a card by submitting the card number.

Request:

```
…<a1:InquiryCardInformation>
      <ns2:StoreId>123456789</ns2:StoreId>
      <ns2:CardNumber>5413…0002</ns2:CardNumber>
</a1:InquiryCardInformation>…
```

Response:

```
…<ipgapi:CardInformation>
      <ns2:Brand>MASTERCARD</ns2:Brand>
      <ns2:CardFunction>credit</ns2:CardFunction>
      <ns2:Country>USA</ns2:Country>
      <ns2:Corporate>CORPORATE</ns2:Corporate>
</ipgapi:CardInformation>
</ipgapi:IPGApiActionResponse>
```

6.6 Basket Information and Product Catalogue

6.6.1 Basket information in transaction messages

The following example shows how you can use the basket parameters to document in the transaction what has been sold.

```
<ns5:IPGApiOrderRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns2:Transaction>
            <ns2:CreditCardTxType>
                  <ns2:Type>sale</ns2:Type>
            </ns2:CreditCardTxType>
            <ns2:CreditCardData>
                  <ns2:CardNumber>4035.…4977</ns2:CardNumber>
                  <ns2:ExpMonth>12</ns2:ExpMonth>
                  <ns2:ExpYear>14</ns2:ExpYear>
            </ns2:CreditCardData>
            <ns2:Payment>
                  <ns2:ChargeTotal>1</ns2:ChargeTotal>
                  <ns2:Currency>EUR</ns2:Currency>
            </ns2:Payment>
            <ns2:TransactionDetails>
                  <ns2:OrderId>68d4a595-fd58-4859-83cd-
1ae13962a3ac</ns2:OrderId>
            </ns2:TransactionDetails>
```

```
            <ns2:Basket>
                  <ns2:Item>
                        <ns2:ID>product ID xyz</ns2:ID>
                        <ns2:Description>description of
abc</ns2:Description>
                        <ns2:ChargeTotal>11</ns2:ChargeTotal>
                        <ns2:Currency>EUR</ns2:Currency>
                        <ns2:Quantity>5</ns2:Quantity>
                        <ns2:Option>
                              <ns2:Name>colour</ns2:Option>
                              <ns2:Choice>blue</ns2:Choice>
                        </ns2:Option>
                        <ns2:Option>
                              <ns2:Name>size</ns2:Option>
                              <ns2:Choice>large</ns2:Choice>
                        </ns2:Option>
                  </ns2:Item>
            </ns2:Basket>
      </ns2:Transaction>
</ns5:IPGApiOrderRequest>
```

6.6.2 Setting up a Product Catalogue

You can store basic information about the products you sell in the following way:

```
<ns5:IPGApiActionRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">

      <ns3:Action>
            <ns3:ManageProducts>
                  <ns3:Function>store</ns3:Function>
                  <ns3:Product>
                        <ns3:ProductID>product ID xyz</ns3:ProductID>
                        <ns2:ChargeTotal>2</ns2:ChargeTotal>
                        <ns2:Currency>EUR</ns2:Currency>
                        <ns3:OfferStarts>
                              2014-12-27T13:29:41.000+01:00
                        </ns3:OfferStarts>
                        <ns3:OfferEnds>
                              2015-09-19T14:29:41.000+02:00
                        </ns3:OfferEnds>
                        <ns2:Option>
                              <ns2:Name>colour</ns2:Option>
                              <ns2:Choice>blue</ns2:Choice>
                        </ns2:Option>
                        <ns2:Option>
                              <ns2:Name>size</ns2:Option>
                              <ns2:Choice>large</ns2:Choice>
                        </ns2:Option>
                  </ns3:Product>
            </ns3:ManageProducts>
      </ns3:Action>
</ns5:IPGApiActionRequest>
```

OfferStarts and OfferEnds are optional and can be used to restrict the visibility of the related products in custom applications but they will not restrict the possibility of a sale. There are further optional fields Description, OptionName and Name. Please take a look at the a1.xsd in the appendix of this document.

The function display shows the requested product with every characteristics.

```
<ns5:IPGApiActionRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns3:Action>
            <ns3:ManageProducts>
                  <ns3:Function>display</ns3:Function>
                  <ns3:Product>
                        <ns3:ProductID>product ID xyz</ns3:ProductID>
                  </ns3:Product>
            </ns3:ManageProducts>
      </ns3:Action>
</ns5:IPGApiActionRequest>
```

The function delete can be used to set the available stock of a product to zero.

```
<ns5:IPGApiActionRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns3:Action>
            <ns3:ManageProducts>
                  <ns3:Function>delete</ns3:Function>
                  <ns3:Product>
                        <ns3:ProductID>product ID xyz</ns3:ProductID>
                  </ns3:Product>
            </ns3:ManageProducts>
      </ns3:Action>
</ns5:IPGApiActionRequest>
```

6.6.3 Manage Product Stock

For every product stock function, the product ID and given options need to exist in your Product Catalogue.

After you have installed a product, you can fill the product stock with the function add.

```
<ns5:IPGApiActionRequest
            xmlns:ns5=http://ipg-online.com/ipgapi/schemas/ipgapi
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns3:Action>
            <ns3:ManageProductStock>
                  <ns3:Function>add</ns3:Function>
                  <ns3:ProductStock>
                        <ns3:ProductID>product ID xyz</ns3:ProductID>
                        <ns2:Option>
                              <ns2:Name>colour</ns2:Option>
                              <ns2:Choice>blue</ns2:Choice>
                        </ns2:Option>
                        <ns2:Option>
                              <ns2:Name>size</ns2:Option>
                              <ns2:Choice>large</ns2:Choice>
                        </ns2:Option>
                        <ns3:Quantity>13</ns3:Quantity>
                  </ns3:ProductStock>
            </ns3:ManageProductStock>
      </ns3:Action>
```

```
</ns5:IPGApiActionRequest>
```

The function substract works in the same way, but will only change the quantity, if the difference will not be negative. If you want to set the quantity to zero you can use the function delete described above.

6.6.4 Sale transactions using product stock

After you have set up the product stock, you can use it to verify if there are enough items on stock for a transaction. A succesful transaction will then substract the quantity. If the product stock contains less than the requested quantity, the transaction will be rejected without any changes to the product stock.

To use this function, add **<ns2:ProductStock>check</ns2:ProductStock>** to Basket.

```
<ns5:IPGApiOrderRequest
            xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns2:Transaction>
            <ns2:CreditCardTxType>
                  <ns2:Type>sale</ns2:Type>
            </ns2:CreditCardTxType>
            <ns2:CreditCardData>
                  <ns2:CardNumber>4035....4977</ns2:CardNumber>
                  <ns2:ExpMonth>12</ns2:ExpMonth>
                  <ns2:ExpYear>14</ns2:ExpYear>
            </ns2:CreditCardData>
            <ns2:Payment>
                  <ns2:ChargeTotal>1</ns2:ChargeTotal>
                  <ns2:Currency>EUR</ns2:Currency>
            </ns2:Payment>
            <ns2:TransactionDetails>
                  <ns2:OrderId>68d4a595-fd58-4859-83cd-
1ae13962a3ac</ns2:OrderId>
            </ns2:TransactionDetails>
            <ns2:Basket>
                  <ns2:ProductStock>check</ns2:ProductStock>
                  <ns2:Item>
                        <ns2:ID>product ID xyz</ns2:ID>
                        <ns2:Description>description of
abc</ns2:Description>
                        <ns2:ChargeTotal>11</ns2:ChargeTotal>
                        <ns2:Currency>EUR</ns2:Currency>
                        <ns2:Quantity>5</ns2:Quantity>
                        <ns2:Option>
                              <ns2:Name>colour</ns2:Option>
                              <ns2:Choice>blue</ns2:Choice>
                        </ns2:Option>
                        <ns2:Option>
                              <ns2:Name>size</ns2:Option>
                              <ns2:Choice>large</ns2:Choice>
                        </ns2:Option>
                  </ns2:Item>
            </ns2:Basket>
      </ns2:Transaction>
</ns5:IPGApiOrderRequest>
```

7 Data Vault

With the Data Vault product option you can store sensitive cardholder data in an encrypted

database in First Data's data centre to use it for subsequent transactions without the need to store this data within your own systems.
If you have ordered this product option, the Web Service API offers you the following functions.

See further possibilities with the Data Vault product in the Integration Guide for the Connect solution.

7.1 Token Type Options

The type of token can be defined with the optional element *TokenType,* which can have 2 possible values : "ONETIME" or "MULTIPAY".

The default value (when no token type gets submitted) is MULTIPAY.

One time token (that are only valid for a specific time span) is an option for merchants, which work with tokens for every transaction, no matter if the consumer registers or prefers to check out as a "guest".
The following XML document represents an example of a request with included element *TokenType* = *MULTIPAY*:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns4:IPGApiOrderRequest
xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns2:Transaction>
            <ns2:CreditCardTxType>
                  <ns2:StoreId>330995001</ns2:StoreId>
                  <ns2:Type>sale</ns2:Type>
            </ns2:CreditCardTxType>
            <ns2:CreditCardData>
                  <ns2:CardNumber>4035*****4977</ns2:CardNumber>
                  <ns2:ExpMonth>12</ns2:ExpMonth>
                  <ns2:ExpYear>28</ns2:ExpYear>
                  <ns2:CardCodeValue>XXX</ns2:CardCodeValue>
            </ns2:CreditCardData>
            <ns2:Payment>
                  <ns2:ChargeTotal>27.2</ns2:ChargeTotal>
                  <ns2:Currency>INR</ns2:Currency>
                  <ns2:TokenType>MULTIPAY</ns2:TokenType>
            </ns2:Payment>
            <ns2:TransactionDetails>
                  <ns2:TransactionOrigin>ECI</ns2:TransactionOrigin>
            </ns2:TransactionDetails>
      </ns2:Transaction>
</ns4:IPGApiOrderRequest>
```

The following XML document represents an example of a request with included element *TokenType* = *ONETIME*:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns4:IPGApiOrderRequest
xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns2:Transaction>
            <ns2:CreditCardTxType>
                  <ns2:StoreId>330995001</ns2:StoreId>
                  <ns2:Type>sale</ns2:Type>
```

```
            </ns2:CreditCardTxType>
            <ns2:CreditCardData>
                  <ns2:CardNumber>4035*****4977</ns2:CardNumber>
                  <ns2:ExpMonth>12</ns2:ExpMonth>
                  <ns2:ExpYear>28</ns2:ExpYear>
                  <ns2:CardCodeValue>XXX</ns2:CardCodeValue>
            </ns2:CreditCardData>
            <ns2:Payment>
                  <ns2:ChargeTotal>27.2</ns2:ChargeTotal>
                  <ns2:Currency>INR</ns2:Currency>
                  <ns2:TokenType>ONETIME</ns2:TokenType>
            </ns2:Payment>
            <ns2:TransactionDetails>
                  <ns2:TransactionOrigin>ECI</ns2:TransactionOrigin>
            </ns2:TransactionDetails>
      </ns2:Transaction>
</ns4:IPGApiOrderRequest>
```

For merchants, which do not wish to define the token themselves, but want it to be generated and returned, the element *AssignToken* should be set to 'true' and no HostedDataId needs to be sent in that case.

The following XML document represents an example of a request for getting the token generated by Gateway, with the element AssignToken = true:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header/>
   <soap:Body>
      <ns5:IPGApiOrderRequest
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
    <ns2:Transaction>
        <ns2:CreditCardTxType>
            <ns2:StoreId>2209905999</ns2:StoreId>
            <ns2:Type>sale</ns2:Type>
        </ns2:CreditCardTxType>
        <ns2:CreditCardData>
            <ns2:CardNumber>4257******0111</ns2:CardNumber>
            <ns2:ExpMonth>12</ns2:ExpMonth>
            <ns2:ExpYear>17</ns2:ExpYear>
            <ns2:CardCodeValue>XXX</ns2:CardCodeValue>
        </ns2:CreditCardData>
        <ns2:Payment>
            <ns2:ChargeTotal>700.00</ns2:ChargeTotal>
            <ns2:Currency>GBP</ns2:Currency>
            <ns2:AssignToken>true</ns2:AssignToken>
        </ns2:Payment>
        <ns2:TransactionDetails>
            <ns2:TransactionOrigin>MOTO</ns2:TransactionOrigin>
        </ns2:TransactionDetails>
        <ns2:Billing>
            <ns2:Address1>Flat 412a  123 London Rd</ns2:Address1>
            <ns2:City>London</ns2:City>
            <ns2:Zip>CH488AQ</ns2:Zip>
            <ns2:Country>GB</ns2:Country>
        </ns2:Billing>
    </ns2:Transaction>
</ns5:IPGApiOrderRequest>
   </soap:Body>
```

```
</soap:Envelope>
```

The following XML document represents an example of a response with the token generated in the element HostedDataID:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ipgapi:IPGApiOrderResponse
xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1">
         <ipgapi:ApprovalCode>Y:609287:8383366115:PPXP:006295</ipgapi:Appro
valCode>
         <ipgapi:AVSResponse>PPX</ipgapi:AVSResponse>
         <ipgapi:Brand>VISA</ipgapi:Brand>
         <ipgapi:Country>ESP</ipgapi:Country>
         <ipgapi:CommercialServiceProvider>CARDNET</ipgapi:CommercialServic
eProvider>
         <ipgapi:OrderId>A-0dd18b32-bc19-40ea-8173-
80537093b18f</ipgapi:OrderId>
         <ipgapi:IpgTransactionId>8383366115</ipgapi:IpgTransactionId>
         <ipgapi:PaymentType>CREDITCARD</ipgapi:PaymentType>
         <ipgapi:ProcessorApprovalCode>609287</ipgapi:ProcessorApprovalCode
>
         <ipgapi:ProcessorCCVResponse>P</ipgapi:ProcessorCCVResponse>
         <ipgapi:ProcessorReferenceNumber>702514006295</ipgapi:ProcessorRef
erenceNumber>
         <ipgapi:ProcessorResponseCode>00</ipgapi:ProcessorResponseCode>
         <ipgapi:ProcessorResponseMessage>Function performed error-
free</ipgapi:ProcessorResponseMessage>
         <ipgapi:TDate>1485354544</ipgapi:TDate>
         <ipgapi:TDateFormatted>2017.01.25 15:29:04
(MEZ)</ipgapi:TDateFormatted>
         <ipgapi:TerminalID>IPGCNP00</ipgapi:TerminalID>
         <ipgapi:TransactionResult>APPROVED</ipgapi:TransactionResult>
         <ipgapi:TransactionTime>1485354544</ipgapi:TransactionTime>
         <ipgapi:HostedData>
            <ipgapi:HostedDataID>7F98D913-85CF-4B88-B994-
B59CB0D4AEB2</ipgapi:HostedDataID>
         </ipgapi:HostedData>
      </ipgapi:IPGApiOrderResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

7.2 Store or update payment information when performing a transaction

Additionally send the parameter HostedDataID together with the transaction data as a unique identification for the payment information in this transaction. Depending on the payment type, credit card number and expiry date or account number and bank code will be stored under this ID. In cases where the submitted ‚HostedDataID' already exists for your store, the stored payment information will be updated.

```
      <ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
            <v1:Transaction>
```

```
            <v1:CreditCardTxType>
                  <v1:Type>sale</v1:Type>
            </v1:CreditCardTxType>
            <v1:CreditCardData>
                  <v1:CardNumber>4111********1111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
            <v1:Payment>
                  <v1:HostedDataID>
                        HDID customer 1234567
                  </v1:HostedDataID>
                  <v1:ChargeTotal>19.00</v1:ChargeTotal>
                  <v1:Currency>978</v1:Currency>
            </v1:Payment>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

The record is only being stored if the authorisation of the payment transaction is successful and your Store has been setup for this service.

If you want to assign multiple IDs to the same payment information (e.g. because your customer has several contracts or accounts with you where they want to used the same card for payment), you can include the parameter HostedDataID multiple times with different values.

7.3 Store payment information from an approved transaction

Payment information can also be stored referring to a previously approved transaction

```
<ns4:IPGApiActionRequest
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns2:Action>
          <ns2:StoreHostedData>
              <ns2:DataStorageItem>
                  <ns2:OrderId>1234567890</ns2:OrderId>
                  <ns2:HostedDataID>
                        4e72021b-d155-4062-872a-30228c0fe023
                  </ns2:HostedDataID>
              </ns2:DataStorageItem>
          </ns2:StoreHostedData>
      </ns2:Action>
  </ns4:IPGApiActionRequest>
```

This action stores the payment information of the transaction with the order id 1234567890. The transaction must be an approved transaction, otherwise this action fails.

7.4 Initiate payment transactions using stored data

If you stored cardholder information using the Data Vault product, you can perform transactions using the ‚HostedDataID' without the need to pass the credit card or bank account data again.

Please note that it is not allowed to store the card code (in most cases on the back of the card) so that for credit card transactions, the cardholder still needs to enter this value. For the checkout process in your web shop, we recommend that you also store the last four digits of the credit card number on your side and display it when it comes to payment. In that way the cardholder can see which of his maybe several cards has been registered in your shop and will be used for this payment transaction.

```
<ipgapi:IPGApiOrderRequest
```

```
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
            <v1:Transaction>
                  <v1:CreditCardTxType>
                        <v1:Type>sale</v1:Type>
                  </v1:CreditCardTxType>
                  <v1:Payment>
                        <v1:HostedDataID>
                              HDID customer 1234567
                        </v1:HostedDataID>
                        <v1:ChargeTotal>19.00</v1:ChargeTotal>
                        <v1:Currency>978</v1:Currency>
                  </v1:Payment>
            </v1:Transaction>
      </ipgapi:IPGApiOrderRequest>
```

7.5 Store payment information without performing a transaction at the same time

Besides the possibility to store new records when performing a payment transaction, you can store payment information using an Action Request. In that way it is also possible to upload multiple records at once. The following example shows the upload for a record with credit card data. Please note that also in this case, existing records will be updated if the HostedDataID is the same.

Example of Request to store the data under given hostedDataID:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns4:IPGApiActionRequest
xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
            <ns2:Action>
              <ns2:StoreHostedData>
                <ns2:DataStorageItem>
                    <ns2:CreditCardData>
                        <ns3:CardNumber>4035******4977</ns3:CardNumber>
                            <ns3:ExpMonth>12</ns3:ExpMonth>
                            <ns3:ExpYear>22</ns3:ExpYear>
                        </ns2:CreditCardData>
 <ns2:HostedDataID>2a356872-54c7-4d09-800c-0be221e72edb</ns2:HostedDataID>
                    </ns2:DataStorageItem>
                    <ns2:DataStorageItem>
                        <ns2:DE_DirectDebitData>
                            <ns3:BankCode>50010060</ns3:BankCode>
                            <ns3:AccountNumber>32121604</ns3:AccountNumber>
                        </ns2:DE_DirectDebitData>
                        <ns2:HostedDataID>6f6de992-e484-4a68-a520-
5f3a32e46fad</ns2:HostedDataID>
                        <ns2:BillingName>Dummy Owner</ns2:BillingName>
                    </ns2:DataStorageItem>
                </ns2:StoreHostedData>
            </ns2:Action>
        </ns4:IPGApiActionRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The result for a successful storage contains the value ***true*** for the parameter *<ns4:successfully>*:

```
      <ns4:IPGApiActionResponse
```

```
                  xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
                  xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
                  xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
            <ns4:successfully>true</ns4:successfully>
      </ns4:IPGApiActionResponse>
```

In cases where one or more records have not been stored successfully, the corresponding Hosted Data IDs are marked in the result:

```
      <ns4:IPGApiActionResponse
                  xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
                  xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
                  xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
            <ns4:successfully>true</ns4:successfully>
            <ns2:Error Code="SGSDAS-020300">
                  <ns2:ErrorMessage>
                        Could not store the hosted data id:
                        691c7cb3-a752-4d6d-abde-83cad63de258.
                        Reason: An internal error has occured while
                        processing your request
                  </ns2:ErrorMessage>
            </ns2:Error>
      </ns4:IPGApiActionResponse>
```

Example of response in case of missing mandatory parameter:

```
<ipgapi:IPGApiActionResponse
xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1">
      <ipgapi:successfully>true</ipgapi:successfully>
       <a1:Error>
            <a1:ErrorMessage>Billing Name is mandatory while creating
hosted data for direct debit.
      </a1:ErrorMessage>
                  </a1:Error>
      <ipgapi:IPGApiActionResponse>
```

Example of response in case where hosted data has not been stored successfully:

```
<ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>true</ipgapi:successfully>
            <a1:Error>
                <a1:ErrorMessage>hosted data id:
E99F19BB9D4F4503B8908D9F86C183F8. Invalid expiration date: CreditCard
[cardNumber=492181...2311, expirationMonth=3, expirationYear=2020,
trackData=(masked), trackOneData=(masked), trackTwoData=(masked),
cardCodeValue=(len:null, isChipCard=null,
enrichedCreditCard=EnrichedCreditCard [typeString=null, issuername=null,
country=null, binCreditCardTypes=[], creditCardInformationList=[],
creditCardType=null, cardFunction=null, commercialCardType=null]]
</a1:ErrorMessage>
            </a1:Error>
        </ipgapi:IPGApiActionResponse>
```

7.6 Avoid duplicate cardholder data for multiple records

To avoid customers using the same cardholder data for multiple user accounts, the additional tag ***DeclineHostedDataDuplicates*** can be sent along with the request. The valid values for this tag are 'true'/'false'. If the value for this tag is set to 'true' and the cardholder data in the request is already found to be associated with another 'hosteddataid', the transaction will be declined.

7.7 Display stored records

Existing records can be displayed using the action *Display*:

```
<ns4:IPGApiActionRequest
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns3:Action>
            <ns3:StoreHostedData>
                  <ns3:DataStorageItem>
                        <ns3:Function>display</ns3:Function>
                        <ns3:HostedDataID>
                              d56feaaf-2d96-4159-8fd6-887e07fc9052
                        </ns3:HostedDataID>
                  </ns3:DataStorageItem>
            </ns3:StoreHostedData>
      </ns3:Action>
</ns4:IPGApiActionRequest>
```

The response contains the stored information. For security reasons, only the first 6 and last 4 digits of credit card numbers are being sent back.

```
<ns4:IPGApiActionResponse
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns4:successfully>true</ns4:successfully>
      <ns4:DataStorageItem>
            <ns2:CreditCardData>
                  <ns3:CardNumber>4035*****4977</ns3:CardNumber>
                  <ns3:ExpMonth>12</ns3:ExpMonth>
                  <ns3:ExpYear>12</ns3:ExpYear>
            </ns2:CreditCardData>
            <ns2:HostedDataID>
                  d56feaaf-2d96-4159-8fd6-887e07fc9052
            </ns2:HostedDataID>
      </ns4:DataStorageItem>
</ns4:IPGApiActionResponse>
```

If the Hosted Data ID does not exist, the API response indicates an error:

```
<ns4:IPGApiActionResponse
            xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns4:successfully>true</ns4:successfully>
      <ns2:Error Code="SGSDAS-020301">
            <ns2:ErrorMessage>
                  Hosted data id:
```

```
                        6c814261-a843-49fb-bacd-1411d3780286 not found.
                  </ns2:ErrorMessage>
            </ns2:Error>
      </ns4:IPGApiActionResponse>
```

The value successfully contains false, only if the data vault can't determined because the request finished in an error.

7.8 Delete existing records

The action "Delete" allows you to remove data records that are no longer needed:

```
      <ns4:IPGApiActionRequest
                  xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
                  xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
                  xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
      <ns3:Action>
            <ns3:StoreHostedData>
                  <ns3:DataStorageItem>
                        <ns3:Function>delete</ns3:Function>
                        <ns3:HostedDataID>
                              9605c2d1-428c-4de2-940e-4bec4737ab5d
                        </ns3:HostedDataID>
                  </ns3:DataStorageItem>
            </ns3:StoreHostedData>
      </ns3:Action>
      </ns4:IPGApiActionRequest>
```

A successful deletion will be confirmed with the following response:

```
      <ns4:IPGApiActionResponse
                  xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
                  xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
                  xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
      <ns4:successfully>true</ns4:successfully>
   </ns4:IPGApiActionResponse>
```

8 Dynamic Currency Conversion (Global Choice™) and Dynamic Pricing

With First Data's Global Choice™, foreign customers have the choice to pay for goods and services purchased online in their home currency when using their Visa or MasterCard credit card for the payment. The currency conversion is quick and eliminates the need for customers to mentally calculate the estimated cost of the purchase in their home currency.

International Visa and MasterCard eCommerce customers can make informed decisions about their online purchases and eradicate any unexpected pricing or foreign exchange conversions on receipt of their monthly statements.

Another option for your foreign customers is to display all pricing within your online store in their home currency using our Dynamic Pricing solution. This solution removes the need for your company to set pricing in any other currency other than your home currency.

If your Store has been activated for one of these product options, you can use this Web Service API to request the currency exchange rates for such transactions.

8.1 Exchange rate requests for Global Choice™

The following example shows a request to the Web Service API to request a card-related exchange rate.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns5:IPGApiActionRequest xmlns:ns5="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:ns2="http://ipg-
online.com/ipgapi/schemas/v1" xmlns:ns3="http://ipg-
online.com/ipgapi/schemas/a1">
            <ns3:Action>
                <ns3:RequestCardRateForDCC>
                    <ns3:StoreId>110994125</ns3:StoreId>
                    <ns3:BIN>402939</ns3:BIN>
                    <ns3:BaseAmount>100.5</ns3:BaseAmount>
                </ns3:RequestCardRateForDCC>
            </ns3:Action>
        </ns5:IPGApiActionRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A successful respose is shown in the following answer:

- The status is given by <ipgapi:successfully>true</ipgapi:successfully>
- The response is wrapped within <ipgapi:CardRateForDCC>

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>true</ipgapi:successfully>
            <ipgapi:CardRateForDCC>
                <v1:InquiryRateId>49150</v1:InquiryRateId>
                <a1:ForeignCurrencyCode>978</a1:ForeignCurrencyCode>
                <a1:ForeignAmount>130.33</a1:ForeignAmount>
                <a1:ExchangeRate>1.2968</a1:ExchangeRate>
                <a1:DccOffered>true</a1:DccOffered>
                <a1:ExpirationTimestamp>2015-06-
23T13:46:00.000+02:00</a1:ExpirationTimestamp>
                <a1:MarginRatePercentage>3.0000</a1:MarginRatePercentage>
                <a1:ExchangeRateSourceName>REUTERS WHOLESALE
INTERBANK</a1:ExchangeRateSourceName>
                <a1:ExchangeRateSourceTimestamp>2014-07-
14T12:46:00.000+02:00</a1:ExchangeRateSourceTimestamp>
            </ipgapi:CardRateForDCC>
        </ipgapi:IPGApiActionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.2 Exchange rate requests for Dynamic Pricing

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns5:IPGApiActionRequest xmlns:ns5="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:ns2="http://ipg-
online.com/ipgapi/schemas/v1" xmlns:ns3="http://ipg-
online.com/ipgapi/schemas/a1">
            <ns3:Action>
                <ns3:RequestMerchantRateForDynamicPricing>
                    <ns3:StoreId>110994125</ns3:StoreId>
                    <ns3:ForeignCurrency>826</ns3:ForeignCurrency>
                    <ns3:BaseAmount>100.5</ns3:BaseAmount>
                </ns3:RequestMerchantRateForDynamicPricing>
            </ns3:Action>
        </ns5:IPGApiActionRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A successful respose is shown in the following answer from IPG:

- The status is given by <ipgapi:successfully>true</ipgapi:successfully>
- The response is wrapped within <ipgapi:MerchantRateForDynamicPricing>

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>true</ipgapi:successfully>
            <ipgapi:MerchantRateForDynamicPricing>
                <v1:InquiryRateId>49150</v1:InquiryRateId>
                <a1:ForeignCurrencyCode>978</a1:ForeignCurrencyCode>
                <a1:ForeignAmount>130.33</a1:ForeignAmount>
                <a1:ExchangeRate>1.2968</a1:ExchangeRate>
                <a1:DccOffered>true</a1:DccOffered>
                <a1:ExpirationTimestamp>2015-06-
23T13:46:00.000+02:00</a1:ExpirationTimestamp>
                <a1:MarginRatePercentage>3.0000</a1:MarginRatePercentage>
                <a1:ExchangeRateSourceName>REUTERS WHOLESALE
INTERBANK</a1:ExchangeRateSourceName>
                <a1:ExchangeRateSourceTimestamp>2014-07-
14T12:46:00.000+02:00</a1:ExchangeRateSourceTimestamp>
            </ipgapi:MerchantRateForDynamicPricing>
        </ipgapi:IPGApiActionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.3 Exchange rate responses

All rate responses share the same XML data-type, they are just wrapped in different parent-tags.

Common fields for all requests are the following:

- Success-status: given with `<ipgapi:successfully>true</ipgapi:successfully>`
- The ID of the request, given with `<v1:InquiryRateId>49150</v1:InquiryRateId>`

The latter InquiryRateId is later to be used to reference the rate request, when performing a transaction with a converted transaction amount.

8.4 Conversion offering

A rate request with an offering returned is shown with the following example.

- The offering is denoted with <a1:DccOffered>true</a1:DccOffered>
- Each offering has associated timestamps, given as xml:date-time.
  - Source time <a1:ExchangeRateSourceTimestamp>2014-07-14T12:46:00.000+02:00</a1:ExchangeRateSourceTimestamp>
  - Expiration time <a1:ExpirationTimestamp>2015-06-23T13:46:00.000+02:00</a1:ExpirationTimestamp>
- The source of the curreny-conversion is shown by <a1:ExchangeRateSourceName>REUTERS WHOLESALE INTERBANK</a1:ExchangeRateSourceName>
- Finally, the currency conversion results are given by the following fields
  - Foreign currency: <a1:ForeignCurrencyCode>978</a1:ForeignCurrencyCode>
  - Foreign amount: <a1:ForeignAmount>130.33</a1:ForeignAmount>
  - Exchange rate: <a1:ExchangeRate>1.2968</a1:ExchangeRate>

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>true</ipgapi:successfully>
            <ipgapi:CardRateForDCC>
                <v1:InquiryRateId>49150</v1:InquiryRateId>
                <a1:ForeignCurrencyCode>978</a1:ForeignCurrencyCode>
                <a1:ForeignAmount>130.33</a1:ForeignAmount>
                <a1:ExchangeRate>1.2968</a1:ExchangeRate>
                <a1:DccOffered>true</a1:DccOffered>
                <a1:ExpirationTimestamp>2015-06-
23T13:46:00.000+02:00</a1:ExpirationTimestamp>
                <a1:MarginRatePercentage>3.0000</a1:MarginRatePercentage>
                <a1:ExchangeRateSourceName>REUTERS WHOLESALE
INTERBANK</a1:ExchangeRateSourceName>
                <a1:ExchangeRateSourceTimestamp>2014-07-
14T12:46:00.000+02:00</a1:ExchangeRateSourceTimestamp>
            </ipgapi:CardRateForDCC>
        </ipgapi:IPGApiActionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.5 Declined rate request

A rate request with a declined offering is shown with the following example.

- The declined offering is denoted with <a1:DccOffered>false</a1:DccOffered>
- Also for declined offerings an ID is returned: <v1:InquiryRateId>4051</v1:InquiryRateId>

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
```

```
        <ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>true</ipgapi:successfully>
            <ipgapi:MerchantRateForDynamicPricing>
                <v1:InquiryRateId>4051</v1:InquiryRateId>
                <a1:DccOffered>false</a1:DccOffered>
            </ipgapi:MerchantRateForDynamicPricing>
        </ipgapi:IPGApiActionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.6 Failed rate request

A rate request which couldn't be processed successfully is shown by the following example:

- Failure-status: given with <ipgapi:successfully>false</ipgapi:successfully>
- The error-element:
  - The error-code by the Code attribute: <a1:Error Code="SGS-27440">
  - The human readable message:
    <a1:ErrorMessage>no amount given</a1:ErrorMessage>

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>false</ipgapi:successfully>
            <a1:Error Code="SGS-27440">
                <a1:ErrorMessage>no amount given</a1:ErrorMessage>
            </a1:Error>
        </ipgapi:IPGApiActionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.7 Global Choice™ transactions

For compliance reasons First Data's Global Choice can only be offered on transactions that take place in full at that time (e.g. Sale, Refund) and not on any delayed settlement (e.g. pre/post auth, recurring) due to the fluctuation of the rate of exchange.

Performing transactions with a converted amount involves the following steps

1. Perform a rate request as described in the sections above.
2. Use the returned InquiryRateId to reference the conversion in the payment transaction message. Use the field DccApplied to denote whether the user has chosen to use the proposed conversion or not.

Please note that an InquiryRateId may be used **only once**. After each transaction request, whether successful or not, regardless of the dccApplied setting used, a new rate has to be requested. Re-using a conversion-rate will result in an error message CORE-DCC-10, since the rate-inquiry is already associated with another transaction.

8.7.1 Step 1: Rate request

The Global Choice™ card-rate-request is shown here to give a complete example:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns5:IPGApiActionRequest xmlns:ns5="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:ns2="http://ipg-
online.com/ipgapi/schemas/v1" xmlns:ns3="http://ipg-
online.com/ipgapi/schemas/a1">
            <ns3:Action>
                <ns3:RequestCardRateForDCC>
                    <ns3:StoreId>110994125</ns3:StoreId>
                    <ns3:BIN>419681</ns3:BIN>
                    <ns3:BaseAmount>202.02</ns3:BaseAmount>
                </ns3:RequestCardRateForDCC>
            </ns3:Action>
        </ns5:IPGApiActionRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The order to be used in a later transaction, the request has to be

- Successful: <ipgapi:successfully>true</ipgapi:successfully>
- With a returned conversion offering <a1:DccOffered>true</a1:DccOffered>
- Not expired <a1:ExpirationTimestamp>2015-06-23T12:46:00.000+02:00</a1:ExpirationTimestamp>

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>true</ipgapi:successfully>
            <ipgapi:CardRateForDCC>
                <v1:InquiryRateId>8391</v1:InquiryRateId>
                <a1:ForeignCurrencyCode>978</a1:ForeignCurrencyCode>
                <a1:ForeignAmount>261.98</a1:ForeignAmount>
                <a1:ExchangeRate>1.2968</a1:ExchangeRate>
                <a1:DccOffered>true</a1:DccOffered>
                <a1:ExpirationTimestamp>2015-06-
23T12:46:00.000+02:00</a1:ExpirationTimestamp>
                <a1:MarginRatePercentage>3.0000</a1:MarginRatePercentage>
                <a1:ExchangeRateSourceName>REUTERS WHOLESALE
INTERBANK</a1:ExchangeRateSourceName>
                <a1:ExchangeRateSourceTimestamp>2014-07-
14T12:46:00.000+02:00</a1:ExchangeRateSourceTimestamp>
            </ipgapi:CardRateForDCC>
        </ipgapi:IPGApiActionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.7.2 Step 2: Using the conversion rate for the payment transaction

The Global Choice™ feature is selected by the element <ns2:InquiryRateReference>.

- The rate-id is used to reference the conversion rate: <ns2:InquiryRateId>8391</ns2:InquiryRateId>

- The users choice whether to apply the proposed rate is specified by: <ns2:DccApplied>true</ns2:DccApplied>

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns5:IPGApiOrderRequest xmlns:ns5="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:ns2="http://ipg-
online.com/ipgapi/schemas/v1" xmlns:ns3="http://ipg-
online.com/ipgapi/schemas/a1">
            <ns2:Transaction>
                <ns2:CreditCardTxType>
                    <ns2:StoreId>110994125</ns2:StoreId>
                    <ns2:Type>return</ns2:Type>
                </ns2:CreditCardTxType>
                <ns2:Payment>
                    <ns2:ChargeTotal>202.02</ns2:ChargeTotal>
                    <ns2:Currency>826</ns2:Currency>
                </ns2:Payment>
                <ns2:TransactionDetails>
                    <ns2:OrderId>API-Test 7dcb3590-2fa7-4702-afab-
adfd34390620 DCCTest::testSaleReturnDCC(110)</ns2:OrderId>
                    <ns2:InquiryRateReference>
                        <ns2:InquiryRateId>8391</ns2:InquiryRateId>
                        <ns2:DccApplied>true</ns2:DccApplied>
                    </ns2:InquiryRateReference>
                </ns2:TransactionDetails>
            </ns2:Transaction>
        </ns5:IPGApiOrderRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For completeness the successful response is also shown here.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiOrderResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">

<ipgapi:ApprovalCode>Y:000000:0014746213:PPXM:0000</ipgapi:ApprovalCode>
            <ipgapi:AVSResponse>PPX</ipgapi:AVSResponse>
            <ipgapi:Brand>VISA</ipgapi:Brand>
            <ipgapi:Country>MLT</ipgapi:Country>

<ipgapi:CommercialServiceProvider>BOSMS</ipgapi:CommercialServiceProvider>
            <ipgapi:OrderId>API-Test 7dcb3590-2fa7-4702-afab-adfd34390620
DCCTest::testSaleReturnDCC(110)</ipgapi:OrderId>
            <ipgapi:PaymentType>CREDITCARD</ipgapi:PaymentType>

<ipgapi:ProcessorApprovalCode>000000</ipgapi:ProcessorApprovalCode>
            <ipgapi:ProcessorCCVResponse>M</ipgapi:ProcessorCCVResponse>
            <ipgapi:ProcessorResponseCode>00</ipgapi:ProcessorResponseCode>

<ipgapi:ProcessorResponseMessage>Authorised</ipgapi:ProcessorResponseMessag
e>
```

```
            <ipgapi:ReferencedTDate>1407154820</ipgapi:ReferencedTDate>
            <ipgapi:TDate>1407154821</ipgapi:TDate>
            <ipgapi:TDateFormatted>2014.08.04 14:20:21
(CEST)</ipgapi:TDateFormatted>
            <ipgapi:TerminalID>80000012</ipgapi:TerminalID>
            <ipgapi:TransactionResult>APPROVED</ipgapi:TransactionResult>
            <ipgapi:TransactionTime>1407154821</ipgapi:TransactionTime>
        </ipgapi:IPGApiOrderResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

9 Payment URL

Payment URL is a functionality that allows you to provide a link to your customers (e.g. in an email invoice, WhatsApp message, SMS, QR code, etc.) which then takes the customer to a First Data-hosted page to securely make the payment with their preferred payment method, whenever convenient for them.

This is especially useful in scenarios where goods get paid after delivery, where no goods get shipped at all (e.g. final payment for trips that have been booked months ago) or for the payment of monthly bills.

You can also implement this functionality for unsuccessful purchases where the original payment transaction has been declined so that you can proactively give your customer a second chance to make their purchase.

The First Data Gateway provides

- The capability to request a Payment URL (link) for a specific amount through this Web Service API
- A hosted payment page where the customer can select the preferred payment method (based on the payment methods that are activated for your account) and make the payment
- A hosted result page that tells the customer if the payment was successful or not, including a Retry button where the customer can chose a different payment method in case the transaction was not successful
- Support for the specific fields that are required for Visa transactions with MCC 6012 in the UK

9.1 Payment URL creation

The request for a Payment URL includes transaction type, amount and currency as well as the language that shall be used on the payment page that will be shown to the customer after accessing the URL.

The URL request stays valid for 182 days (182 * 24 * 3600 seconds) + 1 day (on which the URL was generated).
A merchant can override these settings by setting 'Expiration element' to desired value, which is an expiration date in unix timestamp (in seconds, while IPG calculates it in milliseconds), this value shall be calculated by a merchant himself.

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns5:IPGApiActionRequest
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
            <ns2:Action>
```

```
                <ns2:CreatePaymentURL>
                    <ns2:Transaction>
                        <ns3:PaymentUrlTxType>
                            <ns3:StoreId>120995000</ns3:StoreId>
                            <ns3:Type>sale</ns3:Type>
                        </ns3:PaymentUrlTxType>
                        <ns3:Payment>
                            <ns3:ChargeTotal>13.99</ns3:ChargeTotal>
                            <ns3:Currency>EUR</ns3:Currency>
                        </ns3:Payment>
                        <ns3:TransactionDetails/>
                        <ns3:ClientLocale>
                            <ns3:Language>en</ns3:Language>
                            <ns3:Country>GB</ns3:Country>
                        </ns3:ClientLocale>
                    </ns2:Transaction>
                </ns2:CreatePaymentURL>
            </ns2:Action>
        </ns5:IPGApiActionRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The Response contains the Payment URL:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiActionResponse xmlns:ipgapi="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:a1="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:v1="http://ipg-
online.com/ipgapi/schemas/v1">
            <ipgapi:successfully>true</ipgapi:successfully>
            <ipgapi:OrderId>A-a22cd17f-0e50-4541-9404-159aa62815f0</
            ipgapi:OrderId>
            <ipgapi:TransactionId>88963651</ipgapi:TransactionId>
            <ipgapi:paymentUrl>https://test.ipg-
online.com/connect/gateway/processing?storename=120995000&amp;oid=A-
6d6f02ee-1020-4935-a8fd-e8d34e0ace03&amp;paymentUrlId=efc0d59b-7128-4d36-
ba7d-0f7b642fd9ea</ipgapi:paymentUrl>
        </ipgapi:IPGApiActionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

When the customer completed the payment transaction, the gateway can send a server-to-server transaction notification to a defined Notification URL. Please contact your local support team to get your URL registered for these notifications.

9.2 Payment URL deletion

For cases, when you need to prevent your customers to make a payment twice, you can use “DeletePaymentURL” feature.

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns5:IPGApiActionRequest
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
```

```
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
<ns2:Action>
                <ns2:DeletePaymentURL>
                  <ns2:StoreId>120995000</ns2:StoreId>
                  <ns2:PaymentUrlID>e2fd0144-7644-4a5e-
9e72-71cfa14c37ff</ns2:PaymentUrlID>
                </ns2:DeletePaymentURL>
            </ns2:Action>
        </ns5:IPGApiActionRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

When processing a Payment URL an additional check ensures the Payment URL has not been voided, and if it has, the URL will lead the customer to a screen that explains that the URL is no longer valid.

9.3 Payment URL custom text

For cases where you would like to add a free text to be shown above the payment options on the page that the consumer will see when going to the URL for making the payment, you can submit an element `hostedPaymentPageText` in your request to our Gateway:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns5:IPGApiActionRequest
xmlns:ns5="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/v1">
<ns2:Action>
      <ns2:CreatePaymentURL>
            <ns2:Transaction>
                  <ns3:PaymentUrlTxType>
                        <ns3:StoreId>120995000</ns3:StoreId>
                        <ns3:Type>sale</ns3:Type>
                  </ns3:PaymentUrlTxType>
                  <ns3:Payment>
                        <ns3:Currency>EUR</ns3:Currency>
                  </ns3:Payment>
                  <ns3:TransactionDetails/>
                        <ns3:ClientLocale>
                              <ns3:Language>en</ns3:Language>
                              <ns3:Country>GB</ns3:Country>
                        </ns3:ClientLocale>
            </ns2:Transaction>
            <ns2:hostedPaymentPageText>This is a sample text
            </ns2:hostedPaymentPageText>
      </ns2:CreatePaymentURL>
</ns2:Action>
```

10 3-D Secure Authentication

10.1 3-D Secure authentication (3DS 1.0)

3D Secure is an authentication mechanism designed to reduce fraud and chargebacks in relation to Card-Not-Present transactions.

With our Connect solution (see separate Integration Guide Connect), we can manage the required flows for the authentication process for you. If you should however prefer to handle this process and the required redirections yourself, the Web Service API allows you to make single API calls for the required steps:

1. You make an API call to verify if the cardholder is enrolled to participate in a 3D Secure program
2. For the cases where the cardholder is enrolled, you redirect your customer to the card issuer's Access Control Server (ACS) using the URL that you received in the respose to your verification request
3. You receive the payer authentication response from the card issuer which includes encoded confirmation of the authentication status and send this information in a second API call so that we can verify the signature, decode it and provide you with the result of the authentication
4. You finally trigger the financial transaction (Sale or Pre-Authorisation), referencing to the obtained authentication with a Transaction ID

API call for Step 1

To verify if the card has been enrolled you need to submit a verification request with an `AuthenticateTransaction` parameter set to "true" and TxType = payerAuth.
The following represents an example of a Verification Request (VEReq) with TxType=payerAuth:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns4:IPGApiOrderRequest
xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
            <ns2:Transaction>
                <ns2:CreditCardTxType>
                    <ns2:StoreId>120995000</ns2:StoreId>
                    <ns2:Type>payerAuth</ns2:Type>
                </ns2:CreditCardTxType>
                <ns2:CreditCardData>
                    <ns2:CardNumber>5426*****4979</ns2:CardNumber>
                    <ns2:ExpMonth>12</ns2:ExpMonth>
                    <ns2:ExpYear>24</ns2:ExpYear>
                    <ns2:CardCodeValue>XXX</ns2:CardCodeValue>
                </ns2:CreditCardData>
                <ns2:CreditCard3DSecure>
            <ns2:AuthenticateTransaction>true</ns2:AuthenticateTransaction>
                </ns2:CreditCard3DSecure>
                <ns2:Payment>
                    <ns2:ChargeTotal>13.99</ns2:ChargeTotal>
                    <ns2:Currency>978</ns2:Currency>
                </ns2:Payment>
            </ns2:Transaction>
        </ns4:IPGApiOrderRequest>
```

```
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Our integrated Merchant Plug-in (MPI) then checks the card participation from the 3D Secure directory and returns the redirection URL of the card issuer's Access Control Server (ACS).

If the card is enrolled in 3D Secure, the response to the verification request should contain the following key values:

- PaReq: The Payer Authentication Request, required to initiate the authentication
- ACS URL: The target of 3D Secure redirection
- Term URL: The URL, that the ACS should send the outcome to in your application
- MD : Merchant Data which have to be sent to ACS URL

The following represents an example of a VEReq response:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiOrderResponse
xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1">
            <ipgapi:ApprovalCode>?:waiting 3dsecure</ipgapi:ApprovalCode>
            <ipgapi:Brand>MASTERCARD</ipgapi:Brand>
<ipgapi:CommercialServiceProvider>TELECASH</ipgapi:CommercialServiceProvide
r>
            <ipgapi:OrderId>A-4b9804e6410b84475809e59e1b26</ipgapi:OrderId>
            <ipgapi:IpgTransactionId>8383394827</ipgapi:IpgTransactionId>
            <ipgapi:PaymentType>CREDITCARD</ipgapi:PaymentType>
            <ipgapi:TDate>1493130774</ipgapi:TDate>
            <ipgapi:TDateFormatted>2017.04.25
16:32:54(CEST)</ipgapi:TDateFormatted>
            <ipgapi:TransactionTime>1493130774</ipgapi:TransactionTime>
            <ipgapi:Secure3DResponse>
                <v1:Secure3DVerificationResponse>
                    <v1:VerificationRedirectResponse>
                        <v1:AcsURL>https://3ds-
acs.test.modirum.com/mdpayacs/pareq</v1:AcsURL>
      <v1:PaReq> c7fb83b8ag...73t4a827t4af8738a</v1:PaReq>
<v1:TermUrl>https://www.mywebshop.com/process3dSecure/</v1:TermUrl>
                        <v1:MD>MD1234....sdfk</v1:MD>
                    </v1:VerificationRedirectResponse>
                </v1:Secure3DVerificationResponse>
            </ipgapi:Secure3DResponse>
        </ipgapi:IPGApiOrderResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

API call for Step 3

After you have redirected the cardholder for authentication and have received the payer authentication response from the card issuer, you submit the PARes and MD in your second call to our API. The transaction type must have the same value as in API call for Step 1 message example.

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
```

```
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns4:IPGApiOrderRequest
xmlns:ns4="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/a1">
            <ns2:Transaction>
                <ns2:CreditCardTxType>
                    <ns2:StoreId>120995000</ns2:StoreId>
                    <ns2:Type>payerAuth</ns2:Type>
                </ns2:CreditCardTxType>
                <ns2:CreditCard3DSecure>
                    <ns2:Secure3DRequest>
                        <ns2:Secure3DAuthenticationRequest>
                            <ns2:AcsResponse>
                              <ns2:MD>MDasdadA5809e59e1b263b4aa9</ns2:MD>
                              <ns2:PaRes>eJzVWNeyq8iS…83IBmfhg</ns2:PaRes>
                            </ns2:AcsResponse>
                        </ns2:Secure3DAuthenticationRequest>
                    </ns2:Secure3DRequest>
                </ns2:CreditCard3DSecure>
                <ns2:Payment/>
                <ns2:TransactionDetails>
                    <ns2:IpgTransactionId>8383394827</ns2:IpgTransactionId>
                </ns2:TransactionDetails>
            </ns2:Transaction>
        </ns4:IPGApiOrderRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Our Gateway verifies the response and provides the result back to you, including the data required as the part of authorization request.

The following represents the example of an Authentication Response:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ipgapi:IPGApiOrderResponse
xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1">
            <ipgapi:ApprovalCode>Y:ECI2/5:Authenticated</ipgapi:ApprovalCod
e>
            <ipgapi:Brand>MASTERCARD</ipgapi:Brand>
            <ipgapi:CommercialServiceProvider>TELECASH</ipgapi:CommercialSe
rviceProvider>
            <ipgapi:OrderId>A-123456789</ipgapi:OrderId>
            <ipgapi:IpgTransactionId>8383394827</ipgapi:IpgTransactionId>
            <ipgapi:PaymentType>CREDITCARD</ipgapi:PaymentType>
            <ipgapi:TDate>1493137253</ipgapi:TDate>
            <ipgapi:TDateFormatted>2017.04.25 18:20:53
(CEST)</ipgapi:TDateFormatted>
            <ipgapi:TransactionResult>APPROVED</ipgapi:TransactionResult>
            <ipgapi:TransactionTime>1493137253</ipgapi:TransactionTime>
            <ipgapi:Secure3DResponse>
                <v1:ResponseCode3dSecure>1</v1:ResponseCode3dSecure>
            </ipgapi:Secure3DResponse>
        </ipgapi:IPGApiOrderResponse>
    </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

API call for Step 4

The following represents an example of a *Sale* transaction initiated after previously authenticated request:

```
<soap:Envelope xmlns:Soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns3:IPGApiOrderRequest
xmlns="http://ipg-online.com/ipgapi/schemas/a1"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/ipgapi"
      <ns2:Transaction>
        <ns2:CreditCardTxType>
          <ns2:StoreId>120995000</ns2:StoreId>
          <ns2:Type>sale</ns2:Type>
        </ns2:CreditCardTxType>
        <ns2:TransactionDetails>
                <ns2:IpgTransactionId>8383394827</ns2:IpgTransactionId>
        </ns2:TransactionDetails>
      </ns2:Transaction>
    </ns3:IPGApiOrderRequest>
  </soap:Body>
</soap:Envelope>
```

If you should have your own Merchant Plug-in (MPI) for 3D Secure or use a 3rd party provider for this, you can alternatively submit the result of the authentication process in your Sale or Pre-Authorisation transaction message to the Web Service API. See *CreditCard3DSecure* elements in the XML-Tag overview chapter of this document.

In principle, it may occur that 3D Secure authentications cannot be processed successfully for technical reasons. If one of the systems involved in the authentication process is temporarily not responding, the payment transaction will be processed as a “regular” eCommerce transaction (ECI 7). **A liability shift to the card issuer for possible chargebacks is not warranted in this case**. If you prefer that such transactions shall not be processed at all, our technical support team can block them for your Store on request.

10.2 EMV 3-D Secure authentication (3DS 2.0)

The new EMV 3-D Secure protocol (also known as 3DS 2.0) specification has been developed for the benefit of the entire industry to collaboratively develop the next generation of 3-D Secure protocol. The new version promotes frictionless consumer authentication and enables consumers to authenticate themselves with their card issuer when making card-not-present e-commerce purchases.

Due to continuous development and changes demanded by payment schemes and issuers the integration guide for EMV 3DS protocol has been maintained separately.
Detailed description and examples of the flows can be found on Gateway’s online portal:
https://docs.firstdata.com/org/gateway/node/476

11 Purchasing cards

Purchasing Cards offer businesses the ability to allow their employees to purchase items with a credit card while providing additional information on sales tax, customer code etc. When providing specific details on the payment being made with a Purchasing card favourable addendum interchange rates are applied.

There are three levels of details required for Purchasing Cards:

- Level I — The first level is the standard transaction data; no enhanced data is required at this level.
- Level II — The second level requires that data such as tax amount and customer code be supplied in addition to the standard transaction date. (Visa only have a level II option)
- Level III — The third level allows a merchant to pass a detailed accounting of goods and services purchased to the buyer. All the data for Level I and Level II must also be passed to participate in Level III. (Visa and Mastercard).

PurchaseCard element can contain contain 0-100 *LineItemData* elements.
Detailed description of all *PurchaseCard* elements can be found in the XML-Tag overview chapter of this document.

The following represents an example of a purchasing card L3 transaction including a single *LineItemData* element and mandatory fields:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns4:IPGApiOrderRequest xmlns:ns4="http://ipg-
online.com/ipgapi/schemas/ipgapi" xmlns:ns2="http://ipg-
online.com/ipgapi/schemas/a1" xmlns:ns3="http://ipg-
online.com/ipgapi/schemas/v1">
            <ns3:Transaction>
                <ns3:CreditCardTxType>
                    <ns3:StoreId>110995100</ns3:StoreId>
                    <ns3:Type>sale</ns3:Type>
                </ns3:CreditCardTxType>
                <ns3:CreditCardData>
                    <ns3:CardNumber>4035*****4977</ns3:CardNumber>
                    <ns3:ExpMonth>12</ns3:ExpMonth>
                    <ns3:ExpYear>18</ns3:ExpYear>
                    <ns3:CardCodeValue>XXX</ns3:CardCodeValue>
                </ns3:CreditCardData>
                <ns3:Payment>
                    <ns3:ChargeTotal>23</ns3:ChargeTotal>
                    <ns3:Currency>GBP</ns3:Currency>
                </ns3:Payment>
                <ns3:TransactionDetails>
                    <ns3:PurchaseCard>
                        <ns3:CustomerReferenceID>9632587410</ns3:CustomerRe
ferenceID>
                        <ns3:SupplierInvoiceNumber>321456987</ns3:SupplierI
nvoiceNumber>
                        <ns3:SupplierVATRegistrationNumber>GB18150620</ns3:
SupplierVATRegistrationNumber>
                        <ns3:LineItemData>
                            <ns3:CommodityCode>0</ns3:CommodityCode>
                            <ns3:Description>DIRECT MARKETING
PURCH</ns3:Description>
                            <ns3:Quantity>200000</ns3:Quantity>
                            <ns3:UnitOfMeasure>TPR</ns3:UnitOfMeasure>
```

```
                            <ns3:UnitPrice>1200</ns3:UnitPrice>
                            <ns3:LineItemTotal>1200</ns3:LineItemTotal>
                        </ns3:LineItemData>
                    </ns3:PurchaseCard>
                </ns3:TransactionDetails>
            </ns3:Transaction>
        </ns4:IPGApiOrderRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If *LineItemData* element were removed, the example above would represent a purchasing card level II transaction.

The following represents an example of a purchasing card Level III transaction including multipule *LineItemData* elements with all possible fields populated:

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:IPGApiOrderRequest
xmlns:ns3="http://ipg-online.com/ipgapi/schemas/ipgapi"
xmlns:ns2="http://ipg-online.com/ipgapi/schemas/v1"
xmlns:ns4="http://ipg-online.com/ipgapi/schemas/a1">
            <ns2:Transaction>
                <ns2:CreditCardTxType>
                    <ns2:StoreId>110995100</ns2:StoreId>
                    <ns2:Type>preAuth</ns2:Type>
                </ns2:CreditCardTxType>
                <ns2:CreditCardData>
                    <ns2:CardNumber>4035*****4977</ns2:CardNumber>
                    <ns2:ExpMonth>12</ns2:ExpMonth>
                    <ns2:ExpYear>18</ns2:ExpYear>
                    <ns2:CardCodeValue>XXX</ns2:CardCodeValue>
                </ns2:CreditCardData>
                <ns2:Payment>
                    <ns2:SubTotal>15</ns2:SubTotal>
                    <ns2:ValueAddedTax>4</ns2:ValueAddedTax>
                    <ns2:DeliveryAmount>5</ns2:DeliveryAmount>
                    <ns2:ChargeTotal>24</ns2:ChargeTotal>
                    <ns2:Currency>GBP</ns2:Currency>
                </ns2:Payment>
                <ns2:TransactionDetails>
                    <ns2:PurchaseCard>
      <ns2:CustomerReferenceID>9632587410</ns2:CustomerReferenceID>
      <ns2:SupplierInvoiceNumber>321456987</ns2:SupplierInvoiceNumber>
      <ns2:SupplierVATRegistrationNumber>GB18150620</ns2:SupplierVATRegistr
ationNumber>
                        <ns2:TotalDiscountAmountAndRate>
                            <ns2:Amount>55</ns2:Amount>
                            <ns2:Rate>99.99</ns2:Rate>
                        </ns2:TotalDiscountAmountAndRate>
                        <ns2:VATShippingAmountAndRate>
                            <ns2:Amount>35</ns2:Amount>
                            <ns2:Rate>0.10</ns2:Rate>
                        </ns2:VATShippingAmountAndRate>
                        <ns2:LineItemData>
                            <ns2:CommodityCode>1112</ns2:CommodityCode>
```

```
                            <ns2:ProductCode>22369852147</ns2:ProductCode>
<ns2:Description>DIRECTMARKETINGPURCH</ns2:Description>
                            <ns2:Quantity>200000</ns2:Quantity>
                            <ns2:UnitOfMeasure>TPR</ns2:UnitOfMeasure>
                            <ns2:UnitPrice>1200</ns2:UnitPrice>
                            <ns2:VATAmountAndRate>
                                <ns2:Amount>9999</ns2:Amount>
                                <ns2:Rate>0.1</ns2:Rate>
                            </ns2:VATAmountAndRate>
                            <ns2:DiscountAmountAndRate>
                                <ns2:Amount>13</ns2:Amount>
                                <ns2:Rate>99.99</ns2:Rate>
                            </ns2:DiscountAmountAndRate>
                            <ns2:LineItemTotal>1200</ns2:LineItemTotal>
                        </ns2:LineItemData>
                        <ns2:LineItemData>
                            <ns2:CommodityCode>5647</ns2:CommodityCode>
                            <ns2:ProductCode>22369852148</ns2:ProductCode>
                            <ns2:Description>2-DIRECTMARKETING
PURCH</ns2:Description>
                            <ns2:Quantity>200001</ns2:Quantity>
                            <ns2:UnitOfMeasure>DAY</ns2:UnitOfMeasure>
                            <ns2:UnitPrice>1201</ns2:UnitPrice>
                            <ns2:VATAmountAndRate>
                                <ns2:Amount>9999</ns2:Amount>
                                <ns2:Rate>0.2</ns2:Rate>
                            </ns2:VATAmountAndRate>
                            <ns2:DiscountAmountAndRate>
                                <ns2:Amount>14</ns2:Amount>
                                <ns2:Rate>99.99</ns2:Rate>
                            </ns2:DiscountAmountAndRate>
                            <ns2:LineItemTotal>1202</ns2:LineItemTotal>
                        </ns2:LineItemData>
                        <ns2:LineItemData>
                            <ns2:CommodityCode>575</ns2:CommodityCode>
                            <ns2:ProductCode>22369852149</ns2:ProductCode>
                            <ns2:Description>3-DIRECTMARKETING
PURCH</ns2:Description>
                            <ns2:Quantity>200002</ns2:Quantity>
                            <ns2:UnitOfMeasure>ACR</ns2:UnitOfMeasure>
                            <ns2:UnitPrice>1203</ns2:UnitPrice>
                            <ns2:VATAmountAndRate>
                                <ns2:Amount>9999</ns2:Amount>
                                <ns2:Rate>0.3</ns2:Rate>
                            </ns2:VATAmountAndRate>
                            <ns2:DiscountAmountAndRate>
                                <ns2:Amount>15</ns2:Amount>
                                <ns2:Rate>99.99</ns2:Rate>
                            </ns2:DiscountAmountAndRate>
                            <ns2:LineItemTotal>1204</ns2:LineItemTotal>
                        </ns2:LineItemData>
                    </ns2:PurchaseCard>
                </ns2:TransactionDetails>
            </ns2:Transaction>
        </ns3:IPGApiOrderRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

12 XML-Tag overview

12.1 Overview by transaction type

The following shows which XML-tags need to be submitted for each transaction type as well as which ones can optionally be used. Please only use the fields stated below and also note the order.

For XML-tags related to Card Present transactions with a chip reader and PIN entry device please refer to the xsd's in the Appendix of this document.

Abbreviations:
m: mandatory
o: optional
d: optional with default value
a and **b**: maximum one of the two values (In case you will find a value="a" in the column for a transaction type, it means either all those elements marked with "a" need to be present in the message, or the one marked with "b".)
1: if **a** or **b** is provided optional,
mandatory if **a** and **b** have not been provided
3: mandatory for 3D Secure transactions
s: see details in 3D Secure chapter
f: mandatory for Visa transactions of UK-based Financial Institutions with Merchant Category Code 6012
p: mandatory for split shipment
q: see details in Purchasing cards chapter

Path/ Name	**Credit Card**						
all paths relative to ipgapi:IPGApiOrderRequest/ v1:Transaction	Sale	ForceTicket	PreAuth	PostAuth	Return	Credit	Void
v1:CreditCardTxType/ v1:Type	**m**	**m**	**m**	**m**	**m**	**m**	**m**
v1:CreditCardData/ v1:CardNumber	**a**	**a**	**a**			**a**	
v1:CreditCardData/ v1:ExpMonth	**a**	**a**	**a**			**a**	
v1:CreditCardData/ v1:ExpYear	**a**	**a**	**a**			**a**	
v1:CreditCardData/ v1:CardCodeValue	*o*	*o*	*o*			*o*	
v1:CreditCardData/ v1:TrackData	**b**	**b**	**b**			**b**	
v1:CreditCardData/ v1:Brand	*o*	*o*	*o*			*o*	
v1:CreditCard3DSecure/ v1:VerificationResponse	3	3	3			3	
v1:CreditCard3DSecure/ v1:PayerAuthenticationResponse	s	s	s			s	
v1:CreditCard3DSecure/ v1:DRSPECI	s	s	s			s	
v1:CreditCard3DSecure/ v1:AuthenticationValue	s	s	s			s	

v1:CreditCard3DSecure/ v1:XID	s	s	s			s	
v1:CreditCard3DSecure/ v1:AuthenticateTransaction	s	s	s			s	
v1:CreditCard3DSecure/ v1:Secure3DRequest/ v1: Secure3DAuthenticationRequest/ v1: IVRAuthenticationRequest	s	s	s			s	
v1:CreditCard3DSecure/ v1:Secure3DRequest/ v1:Secure3DAuthenticationRequest/ v1:AcsResponse	s	s	s			s	
v1:CreditCard3DSecure/ v1:Secure3DRequest/ v1: Secure3DVerificationRequest v1: IVRVerificationRequest	s	s	s			s	
v1:CreditCard3DSecure/ v1:Secure3DverificationResponse/ v1:IVRVerificationResponse	s	s	s			s	
v1:CreditCard3DSecure/ v1:Secure3DverificationResponse/ v1:VerificationRedirectResponse	s	s	s			s	
v1:CreditCardData/ v1:Upop	u	u	u			u	
v1:cardFunction/ v1:Type	*o*	*o*	*o*			*o*	
v1:DE_DirectDebitTxType/ v1:Type							
v1:DE_DirectDebitData/ v1:TrackData							
v1:DE_DirectDebitData/ v1:MandateReference							
v1:DE_DirectDebitData/ v1:MandateType							
v1:DE_DirectDebitData/ v1:DateOfMandate							
v1:Payment/ v1:HostedDataID	***1***	***1***	***1***			***1***	
v1:Payment/ v1:HostedDataStoreID	***1***	***1***	***1***			***1***	
v1:Payment/ v1:DeclineHostedDataDuplicates	***1***	***1***	***1***			***1***	
v1:Payment/ v1:numberOfInstallments	*o*						
v1:Payment/ v1:installmentsInterest	*d*						
v1:Payment/ v1:installmentDelayMonths	*o*						
v1:Payment/ v1:SubTotal	*o*	*o*	*o*	*o*	*o*	*o*	
v1:Payment/ v1:ValueAddedTax	*o*	*o*	*o*	*o*	*o*	*o*	
v1:Payment/ v1:localTax	*o*	*o*	*o*	*o*	*o*	*o*	

v1:Payment/ v1:DeliveryAmount	*o*	*o*	*o*	*o*	*o*	*o*	
v1:Payment/ v1:ChargeTotal	**m**	**m**	**m**	**m**	**m**	**m**	
v1:Payment/ v1:Currency	**m**	**m**	**m**	**m**	**m**	**m**	
v1:recurringType	*o*						
v1:WalletType	*o*						
v1:WalletID	*o*						
v1:TransactionDetails/ v1:OrderId	*o*	*o*	*o*	**m**	**m**	*o*	**a**
v1:TransactionDetails/ v1:MerchantTransactionId	*o*	*o*	*o*	*o*	*o*	*o*	*o*
v1:TransactionDetails/ v1:Ip	*o*		*o*			*o*	
v1:TransactionDetails/ v1:ReferenceNumber		**m**					
v1:TransactionDetails/ v1:Tdate							**a**
v1:TransactionDetails/ v1:ReferencedMerchantTransactionId							**b**
v1:TransactionDetails/ v1:TransactionOrigin	*d*		*d*			*d*	
v1:TransactionDetails/ v1:InvoiceNumber	*o*	*o*	*o*	*o*		*o*	
v1:TransactionDetails/ v1:PONumber	*o*	*o*	*o*			*o*	
v1:TransactionDetails/ v1:DynamicMerchantName	*o*	*o*	*o*			*o*	
v1:TransactionDetails/ v1:Comments	*o*	*o*	*o*	*o*	*o*	*o*	*o*
v1:TransactionDetails/ v1:PurchaseCard	*q*	*q*	*q*	*q*	*q*	*q*	*q*
v1:TransactionDetails/ v1:Terminal/ v1:TerminalID	*o*	*o*	*o*			*o*	
v1:TransactionDetails/ v1:InquiryRateReference	*o*	*o*	*o*				
v1:TransactionDetails/ v1:SplitShipment/ v1:SequenceCount			*o*	*o*			
v1:TransactionDetails/ v1:SplitShipment/ v1:FinalShipment				**p**			
v1:Billing/ v1:CustomerID	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Name	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Company	*o*	*o*	*o*			*o*	

v1:Billing/ v1:Address1	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Address2	*o*	*o*	*o*			*o*	
v1:Billing/ v1:City	*o*	*o*	*o*			*o*	
v1:Billing/ v1:State	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Zip	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Country	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Phone	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Fax	*o*	*o*	*o*			*o*	
v1:Billing/ v1:Email	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:Type	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:Name	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:Address1	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:Address2	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:City	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:State	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:Zip	*o*	*o*	*o*			*o*	
v1:Shipping/ v1:Country	*o*	*o*	*o*			*o*	
v1:Basket/ v1:Item/ v1:ID	*o*	*o*	*o*			*o*	
v1:Basket/ v1:Item/ v1:Description	*o*	*o*	*o*			*o*	
v1:Basket/ v1:Item/ v1:SubTotal							
v1:Basket/ v1:Item/ v1:ValueAddedTax							
v1:Basket/ v1:Item/ v1:DeliveryAmount							
v1:Basket/ v1:Item/ v1:ChargeTotal	*o*	*o*	*o*			*o*	
v1:Basket/ v1:Item/ v1:Currency							

v1:Basket/ v1:Item/ v1:Quantity	*o*	*o*	*o*			*o*	
v1:Basket/ v1:Item/ v1:Option/ v1:Name	*o*	*o*	*o*			*o*	
v1:Basket/ v1:Item/ v1:Choice	*o*	*o*	*o*			*o*	
v1:TopUpTxType/ v1:MPCharge/ v1:MNSP							
v1:TopUpTxType/ v1:MPCharge/ v1:MSISDN							
v1:TopUpTxType/ v1:MPCharge/ v1:PaymentType							
v1:ClientLocale/ v1:Language	*d*	*d*	*d*	*d*	*d*	*d*	*d*
v1:ClientLocale/ v1:Country	*d*	*d*	*d*	*d*	*d*	*d*	*d*
v1:MCC6012Details/ v1:BirthDate	**f**	**f**	**f**				
v1:MCC6012Details/ v1:AccountFirst6	**f,a**	**f,a**	**f,a**				
v1:MCC6012Details/ v1:AccountLast4	**f,a**	**f,a**	**f,a**				
v1:MCC6012Details/ v1:AccountNumber	**f,b**	**f,b**	**f,b**				
v1:MCC6012Details/ v1:PostCode	**f**	**f**	**f**				
v1:MCC6012Details/ v1:Surname	**f**	**f**	**f**				

Path/ Name	**PayPal**				**Mobile Top-up**
all paths relative to ipgapi:IPGApiOrderRequest/ v1:Transaction	PostAuth	Return	Credit	Void	MPCharge
v1:CreditCardTxType/ v1:Type					
v1:CreditCardData/ v1:CardNumber					
v1:CreditCardData/ v1:ExpMonth					
v1:CreditCardData/ v1:ExpYear					
v1:CreditCardData/ v1:CardCodeValue					

v1:CreditCardData/ v1:TrackData					
v1:CreditCard3DSecure/ v1:VerificationResponse					
v1:CreditCard3DSecure/ v1:PayerAuthenticationResponse					
v1:CreditCard3DSecure/ v1:AuthenticationValue					
v1:CreditCard3DSecure/ v1:XID					
v1:DE_DirectDebitTxType/ v1:Type					
v1:DE_DirectDebitData/ v1:MandateReference					
v1:DE_DirectDebitData/ v1:MandateType					
v1:DE_DirectDebitData/ v1:TrackData					
v1:PayPalTxType/ v1:Type	**m**	**m**	**m**	**m**	
v1:Payment/ v1:HostedDataID					
v1:Payment/ v1:HostedDataStoreID					
v1:Payment/ v1:DeclineHostedDataDuplicates					
v1:Payment/ v1:SubTotal	*o*	*o*	*o*		*o*
v1:Payment/ v1:ValueAddedTax	*o*	*o*	*o*		*o*
v1:Payment/ v1:DeliveryAmount	*o*	*o*	*o*		*o*
v1:Payment/ v1:ChargeTotal	**m**	**m**	**m**		**m**
v1:Payment/ v1:Currency	**m**	**m**	**m**		**m**
v1:TransactionDetails/ v1:OrderId	**m**	**m**	*o*	**m**	*o*
v1:TransactionDetails/ v1:MerchantTransactionId	*o*	*o*	*o*	*o*	*o*
v1:TransactionDetails/ v1:Ip			*o*		*o*
v1:TransactionDetails/ v1:ReferenceNumber					
v1:TransactionDetails/ v1:Tdate				**a**	
v1:TransactionDetails/ v1:ReferencedMerchantTransactionId				**b**	
v1:TransactionDetails/ v1:TransactionOrigin			*d*		
v1:TransactionDetails/ v1:InvoiceNumber			*o*		*o*

v1:TransactionDetails/ v1:PONumber			*o*		*o*
v1:TransactionDetails/ v1:DynamicMerchantName			*o*		*o*
v1:TransactionDetails/ v1:Comments	*o*	*o*	*o*	*o*	*o*
v1:Billing/ v1:CustomerID			*o*		*o*
v1:Billing/ v1:Name			*o*		*o*
v1:Billing/ v1:Company			*o*		*o*
v1:Billing/ v1:Address1			*o*		*o*
v1:Billing/ v1:Address2			*o*		*o*
v1:Billing/ v1:City			*o*		*o*
v1:Billing/ v1:State			*o*		*o*
v1:Billing/ v1:Zip			*o*		*o*
v1:Billing/ v1:Country			*o*		*o*
v1:Billing/ v1:Phone			*o*		*o*
v1:Billing/ v1:Fax			*o*		*o*
v1:Billing/ v1:Email			**m**		*o*
v1:Shipping/ v1:Type			*o*		
v1:Shipping/ v1:Name			*o*		
v1:Shipping/ v1:Address1			*o*		
v1:Shipping/ v1:Address2			*o*		
v1:Shipping/ v1:City			*o*		
v1:Shipping/ v1:State			*o*		
v1:Shipping/ v1:Zip			*o*		
v1:Shipping/ v1:Country			*o*		
v1:Basket/ v1:Item/ v1:ID			*o*		
v1:Basket/ v1:Item/ v1:Description			*o*		

v1:Basket/ v1:Item/ v1:SubTotal					
v1:Basket/ v1:Item/ v1:ValueAddedTax					
v1:Basket/ v1:Item/ v1:DeliveryAmount					
v1:Basket/ v1:Item/ v1:ChargeTotal			*o*		
v1:Basket/ v1:Item/ v1:Currency					
v1:Basket/ v1:Item/ v1:Quantity			*o*		
v1:Basket/ v1:Item/ v1:Option/ v1:Name			*o*		
v1:Basket/ v1:Item/ v1:Choice			*o*		
v1:TopUpTxType/ v1:MPCharge/ v1:MNSP					**m**
v1:TopUpTxType/ v1:MPCharge/ v1:MSISDN					**m**
v1:TopUpTxType/ v1:MPCharge/ v1:PaymentType					**m**
v1:ClientLocale/ v1:Language	*d*	*d*	*d*	*d*	*d*
v1:ClientLocale/ v1:Country	*d*	*d*	*d*	*d*	*d*

12.2 Description of the XML-Tags

12.2.1 CreditCardTxType

Path/Name	XML Schema type	Description
v1:CreditCardTxType/ v1:Type	xs:string	Stores the transaction type. Possible values are sale, forceTicket, preAuth, postAuth, return, credit and void.

12.2.2 CreditCardData

Path/Name	XML Schema type	Description

v1:CreditCardData/ v1:CardNumber	xs:string	Stores the customer's credit card number. Make sure that the string contains only digits, i.e. passing the number e.g. in the format *xxxx-xxxx-xxxx-xxxx* will result in an error returned by the Web Service API.
v1:CreditCardData/ v1:ExpMonth	xs:string	Stores the expiration month of the customer's credit card. Make sure that the content of this element always contains two digits, i.e. a card expiring in July will have this element with value 07.
v1:CreditCardData/ v1:ExpYear	xs:string	Stores the expiration year of the customer's credit card. The same formatting restrictions as for the v1:ExpMonth element apply here.
v1:CreditCardData/ v1:CardCodeValue	xs:string	Stores the three or four digit card security code (CSC) – sometimes also referred to as card verification value (CVV) or code (CVC) – which is typically printed on the back of the credit card. For information about the benefits of CSC contact support.
v1:CreditCardData/ v1:TrackData	xs:string	Stores the track data of a card when using a card reader instead of keying in card data (can optionally be used instead of transmitting CardNumber, ExpMonth and ExpYear). This field needs to contain at least the concatenated track 1 and 2 data. Track data 3 is optional. The track data must include the track and field separators as they are stored on the card. Example for the track data separator from track data 1 and 2 without the data: %…?;…?
v1:CreditCardData/ v1:TrackData	xs:string	Optional field for the brand of the credit card. If this field is set, the transaction will only be processed if the card number matches the brand.

For XML-tags related to Card Present transactions with a chip reader and PIN entry device please refer to the xsd's in the Appendix of this document.

12.2.3 recurringType

Path/Name	XML Schema type	Description
v1:recurringType	xs:string	This field allows you to flag transactions as recurring. It can be set to FIRST for the first transaction of a series and to REPEAT for the subsequent transactions in a series.

12.2.4 UnscheduledCredentialOnFileType

Path/Name	XML Schema type	Description
v1: UnscheduledCredentialOnFileType	xs:string	This field allows you to flag transactions as Unscheduled Credential On File Type. Currently the valid values are FIRST,

		CARDHOLDER_INITIATED or MERCHANT_INITIATED to advise the scenario if the credential is stored on your side.

12.2.5 Wallet

Path/Name	XML Schema type	Description
v1:Wallet/ v1:WalletType	xs:string	This field allows you to submit the wallet type for transactions that have been initiated through a digital wallet. Currently the valid values are MASTERPASS, APPLE_PAY, ANDROID_PAY
v1:Wallet/ v1:WalletID	xs:string	This field allows you to submit the wallet ID for transactions that have been initiated through a digital wallet.

12.2.6 cardFunction

Path/Name	XML Schema type	Description
v1:cardFunction/ v1:Type	xs:string	This field allows you to indicate the card function in case of combo cards which provide credit and debit functionality on the same card. It can be set to credit or debit.

12.2.7 CreditCard3DSecure

Path/Name	XML Schema type	Description
v1:CreditCard3DSecure/ v1:VerificationResponse	xs:string	Stores the VerificationResponse (VERes) of your Merchant Plug-in.
v1:CreditCard3DSecure/ v1:PayerAuthenticationResponse	xs:string	Stores the PayerAuthenticationResponse (PARes) of your Merchant Plug-in.
v1:CreditCard3DSecure/ v1:DSRPECI	xs:string	To set ECI value for Digital Secure Remote Payments. If you submit this parameter, any values for parameters VerificationResponse and PayerAuthenticationResponse will be ignored.
v1:CreditCard3DSecure/ v1:AuthenticationValue	xs:string	Stores the AuthenticationValue (MasterCard: AAV or VISA: CAAV) of your Merchant Plug-in.
v1:CreditCard3DSecure/ v1:XID	xs:string	Stores the XID of your Merchant Plug-in.
v1:CreditCard3DSecure/ v1:AuthenticateTransaction	xs:boolean	Indicates, if transaction is going to be authenticated as 3DSecure transaction.If a card is enrolled, the response will contain the Verification Redirect Response element
v1:CreditCard3DSecure/ v1:Override3dsCountryExclusion	xs:boolean	Set true, if for this transaction you would like to enforce 3-D Secure authentication, despite this country possibly being exempted from authentication due to the

		merchant configured list of countries where 3-D Secure is not required.
v1:CreditCard3DSecure/ v1:SkipTRA	xs:boolean	Set to true, if for this transaction you would enforce 3-D Secure authentication, despite of the result of Transaction Risk Analysis performed by RiskShield

Please note that these are values you receive from your own Merchant Plug-in for 3D Secure or a solution of a 3D Secure provider. The integrated 3D Secure functionality of the Connect feature can not be used for transactions via the API for technical reasons.

12.2.8 3DSecure Authentication / Verification Redirect Response

Path/Name	XML Schema type	Description
v1:VerificationRedirectResponse v1:AcsURL	xs:string	Represents the target of the 3D Secure redirection
v1:VerificationRedirectResponse v1:PaReq	xs:string	Represents the PAReq data which has to be sent in the "PAReq" attribute to the ACS URL.
v1: VerificationRedirectResponse/ v1:TermUrl	xs:string	Represents the default TermURL, which should be used in order to process the response from the 3D Secure process. In case that a merchant would like to parse the response by himself, he has to specify the "TermUrl" parameter in the form with his custom URL, in which he will process the response and call the API with the response PARes and Merchant Data.
v1: VerificationRedirectResponse/ v1:MD	xs:string	Represents the merchant data which has to be sent in the "MD" attribute to the ACS URL.

12.2.9 3DSecure Authentication / ACS Response

Path/Name	XML Schema type	Description
v1:AcsResponse v1:MD	xs:string	Merchant Data from ACS redirection POST attribute ("MD"attribute). *Please note, that this element might not be sent back by the issuer (ACS) in case of EMV 3DS protocol (3DS 2.0)*
v1:AcsResponse v1:PaRes	xs:string	Represents PARes data from ACS redirection POST attribute ("PARes" attribute).

12.2.10PayPalTxType

Path/Name	XML Schema type	Description
v1:PayPalTxType/ v1:Type	xs:string	Stores the transaction type. Possible values are postAuth, return, credit and void.

12.2.11Payment

Path/Name	XML Schema type	Description
v1:Payment/ v1:HostedDataID	xs:string	Stores the Hosted Data ID for the Data Vault product
v1:Payment/ v1:HostedDataStoreID	xs:string	Stores the Hosted Data ID for the Data Vault product in this store (only as technical user)
v1:Payment/ v1:DeclineHostedDataDuplicates	xs:string	Declines duplicate credit card
v1:Payment/ v1:numberOfInstallments	xs:string	Stores the number of instalments for a Sale transaction if the customer pays the amount in several parts
v1:Payment/ v1:installmentsInterest	xs:string	Indicates, if the installment interest has been applied; possible values "yes" or "no"
v1:Payment/ v1:installmentDelayMonths	xs:string	Represents the number of months the first payment will be delayed; possible values in the range <1; 99>
v1:Payment/ v1:SubTotal	xs:decimal	Stores the Sub Total of an order. If this member is set, then also ChargeTotal has to be set.
v1:Payment/ v1:ValueAddedTax	xs:decimal	Stores the VAT of an order. If this member is set, then also SubTotal has to be set.
v1:Payment/ v1:DeliveryAmount	xs:decimal	Stores the delivery amount of an order. If this member is set, then also SubTotal has to be set.
v1:Payment/ v1:ChargeTotal	xs:double	Stores the transaction amount. Make sure that the number of positions after the decimal point does not exceed 2, e.g. 3.123 would be invalid – however, 3.12, 3.1, and 3 are correct.
v1:Payment/ v1:Currency	xs:string	Stores the currency as a three-digit ISO 4217 value (e. g. 978 for Euro)

12.2.12TransactionDetails

Path/Name	XML Schema type	Description
v1:TransactionDetails/ v1:OrderId	xs:string	Stores the order ID. This must be unique per Store ID. If no Order ID is transmitted, the Gateway will generate one automatically. Note: For cases where you plan to use EMV 3DS Authentication prior to the authoriazation, please use only the following characters in OrderId: A-Z, a-z, 0-9, '-'
v1:TransactionDetails/ v1:MerchantTransactionId	xs:string	Allows you to assign a unique ID for the transaction. This ID can be used to reference to this transactions in a Void

		request (ReferencedMerchantTransactionId) or to retrieve transaction details with the API action InquiryTransaction. Uniqueness needs to be enforced by the merchant.
v1:TransactionDetails/ v1:Ip	xs:string	Stores the customer's IP address which can be used by the Web Service API for fraud detection by IP address. Make sure that you supply the IP in the format *xxx.xxx.xxx.xxx*, e.g. 128.0.10.2 would be a valid IP.
v1:TransactionDetails/ v1:ReferenceNumber	xs:string	Stores the six digit reference number you have received as the result of a successful external authorization (e.g. by phone). The Gateway needs this number for uniquely mapping a *ForceTicket* transaction to a previously performed external authorization.
v1:TransactionDetails/ v1:PurchaseCard	xs:string	Stores the purchasing card Level II and Level III transaction data.
v1:TransactionDetails/ v1:TDate	xs:string	Stores the TDate of the *Sale*, *PostAuth*, *ForceTicket*, *Return*, or *Credit* transaction this *Void* transaction refers to. A TDate value is returned within the response to a successful transaction of one of these five types. When performing a *Void* transaction, you have to pass the TDate in addition to the order ID for uniquely identifying the transaction to be voided. The scenario presented below gives an example.
v1:TransactionDetails/ v1:ReferencedMerchantTransactionId	xs:string	Stores the MerchantTransactionId of the *Sale*, *PostAuth*, *ForceTicket*, *Return*, or *Credit* transaction this *Void* transaction refers to. This can be used as an alternative to TDate if you assigne a MerchantTransactionId in the original transaction request.
v1:TransactionDetails/ v1:TransactionOrigin	xs:string	The source of the transaction. The possible values are ECI (if the order was received via email or Internet), MOTO (mail order / telephone order), MAIL (mail order), PHONE (telephone order) and RETAIL (face to face).
v1:TransactionDetails/ v1:SplitShipment/ v1:SequenceCount	xs:int	Stores the total number of shipments in case of split shipment. Can either be included in the PreAuth or the first PostAuth. A different value in the first PostAuth overwrites the value from the PreAuth.
v1:TransactionDetails/ v1:SplitShipment/ v1:FinalShipment	xs:boolean	Needs to be set to "true" in the final PostAuth of a series of split shipments.
v1:TransactionDetails/ v1:InvoiceNumber	xs:string	Stores the invoice number.
v1:TransactionDetails/ v1:PONumber	xs:string	Stores the purchase order number.

v1:TransactionDetails/ v1:DynamicMerchantName	xs:string	Stores a dynamic merchant name for the cardholder's statement
v1:TransactionDetails/ v1:Comments	xs:string	Stores the comments.
v1:TransactionDetails/ v1:SCAExemptionIndicators	xs:string	Indicates the reason to skip Strong Customer Authentication (SCA), e.g. 3-D Secure with submitting directly an authorization request. For available values and more details see the chapter 13.2.30
v1:TransactionDetails/ v1:HighRiskPurchaseIndicator	xs:boolean	Needs to be set to 'true', for transactions handling a cryptocurrency and initiated from a MCC 6051(Quasi Cash—Merchant) store; or for transactions handling high risk securities initiated from the store with MCC 6211 (Securities—Brokers/ Dealers).
v1:TransactionDetails/ v1:vmid	xs:string	8 characters Visa Merchant Identifier assigned by Visa, required for Trusted Merchant and Delegated Authentication. Can be used only if you are enrolled with Visa's Delegated Authentication program.

12.2.13Purchasing Cards

Path/Name	XML Schema type	Description
v1:PurchaseCard v1:CustomerReferenceID	xs:string (20max)	A reference to a Customer Code/Customer Reference ID
v1:PurchaseCard v1:SupplierInvoiceNumber	xs:string (30max)	A reference to a Purchase Identifier/Merchant related data.
v1:PurchaseCard v1:SupplierVATRegistrationNumber	xs:string (30max)	Represents a Merchant VAT registration/Single Business Reference Number/Merchant Tax ID or Corporation VAT Number
v1:PurchaseCard v1:TotalDiscountAmountAndRate	xs:string	Represents the total discount amount applied to a transaction (i.e. total transaction percentage discounts, fixed transaction amount reductions or summarization of line item discounts).
v1:PurchaseCard v1:VATShippingAmountAndRate	xs:string	Represents the total freight/shipping amount applied to a transaction.
v1:PurchaseCard v1:LineItemData	xs:string	Represents mandatory data for Level III transactions.

12.2.14Purchasing Cards / Line Item Data

Path/Name	XML Schema type	Description
v1:LineItemData v1:CommodityCode	xs:numeric (positive, 4max)	A reference to a commodity code used to classify purchased item
v1:LineItemData v1:ProductCode	xs:string (20max)	A reference to a merchant product identifier, the Universal Product Code (UPC) of purchased item

v1:LineItemData v1:Description	xs:string (30max)	Represents a description of purchased item
v1:LineItemData v1:Quantity	xs:numeric (minInclusive value="1")	Represents a quantity of purchased items.
v1:LineItemData v1:UnitOfMeasure	xs:string (3 max)	Represents a unit of measure of purchased items
v1:LineItemData v1:UnitPrice	xs:decimal	Represents mandatory data for Level III transactions.
v1:LineItemData v1:VATAmountAndRate	xs:decimal	Represents a rate of the VAT amount, e.g. 0.09 (means 9%)
v1:LineItemData v1:DiscountAmountAndRate	xs:decimal	Represents a rate of the discount amount, e.g. 0.09 (means 9%)
v1:LineItemData v1:LineItemTotal	xs:decimal	This field is a calculation of the unit cost multiplied by the quantity and less the discount per line item. The calculation is reflected as: [Unit Cost * Quantity] - Discount per Line Item = Line Item Total.

12.2.15InquiryRateReference

Path/Name	XML Schema type	Description
v1:InquiryRateReference/ v1:InquiryRateId	xs:long	A reference to a rate-inquiry for transactions with Global Choice™or Dynamic Pricing.
v1:InquiryRateReference/ v1:DccApplied	xs:boolean	Specifies whether a cardholder has choosen to accept the proposed currency conversion offering when using Global Choice™.

12.2.16Billing

Path/Name	XML Schema type	Description
v1:Billing/ v1:CustomerID	xs:string	Stores your ID for your customer.
v1:Billing/ v1:Name	xs:string	Stores the customer's name. If provided, it will appear on your transaction reports.
v1:Billing/ v1:Company	xs:string	Stores the customer's company. If provided, it will appear on your transaction reports.
v1:Billing/ v1:Address1	xs:string	Stores the first line of the customer's address. If provided, it will appear on your transaction reports.
v1:Billing/ v1:Address2	xs:string	Stores the second line of the customer's address. If provided, it will appear on your transaction reports.
v1:Billing/ v1:City	xs:string	Stores the customer's city. If provided, it will appear on your transaction reports.
v1:Billing/ v1:State	xs:string	Stores the customer's state. If provided, it will appear on your transaction reports.
v1:Billing/ v1:Zip	xs:string	Stores the customer's zip code. If provided, it will appear on your transaction reports.

v1:Billing/ v1:Country	xs:string	Stores the customer's country. If provided, it will appear on your transaction reports.
v1:Billing/ v1:Phone	xs:string	Stores the customer's phone number. If provided, it will appear on your transaction reports.
v1:Billing/ v1:Fax	xs:string	Stores the customer's fax number. If provided, it will appear on your transaction reports.
v1:Billing/ v1:Email	xs:string	Stores the customer's Email address. If provided, it will appear on your transaction reports. If you are using the email transaction notification feature, this email address will be used for notifications to your customer.

12.2.17Shipping

Path/Name	XML Schema type	Description
v1:Shipping/ v1:Name	xs:string	Stores the name of the recipient. If provided, it will appear on your transaction reports.
v1:Shipping/ v1:Address1	xs:string	Stores the first line of the shipping address. If provided, it will appear on your transaction reports.
v1:Shipping/ v1:Address2	xs:string	Stores the second line of the shipping address. If provided, it will appear on your transaction reports.
v1:Shipping/ v1:City	xs:string	Stores the recipient's city. If provided, it will appear on your transaction reports.
v1:Shipping/ v1:State	xs:string	Stores the recipient's state. If provided, it will appear on your transaction reports.
v1:Shipping/ v1:Zip	xs:string	Stores the recipient's zip code. If provided, it will appear on your transaction reports.
v1:Shipping/ v1:Country	xs:string	Stores the recipient's country. If provided, it will appear on your transaction reports.

12.2.18ClientLocale

Path/Name	XML Schema type	Description
v1:ClientLocale/ v1:Language	xs:string	If you are using the email transaction notification feature, this language will be used for notifications to your customer. Possible values are: de, en, it.
v1:ClientLocale/ v1:Country	xs:string	Specifies the variant of the language. This member can only be set if the language is set. Possible values are: DE, GB, IT. If you do not define a country, a matching country will be chosen.

If you do not submit language information in the transaction, the language settings of your store will be used for the email notifications.

12.2.19RequestCardRateForDCC

Path/Name	XML Schema type	Description
v1:RequestCardRateForDCC/ v1:StoreId	xs:string (max 20)	Your Store ID. The base currency is derived from the Store settings.
v1:RequestCardRateForDCC/ v1:BIN	xs:int	The credit cards' Bank Identifier Number (first 6 digits of credit card number)
v1:RequestCardRateForDCC/ v1:BaseAmount	xs:decimal	The amount to be converted (optional). When no amount is given in the request, no amount will be returned, only the conversion rate.

12.2.20RequestMerchantRateForDynamicPricing

Path/Name	XML Schema type	Description
v1:RequestCardRateForDCC/ v1:StoreId	xs:string (max 20)	Your Store ID. The base currency is derived from the Store settings.
v1:RequestCardRateForDCC/ v1:ForeignCurrency	xs:string	The currency to be converted to. (ISO_4217 format)
v1:RequestCardRateForDCC/ v1:BaseAmount	xs:decimal	The amount to be converted (optional). When no amount is given in the request, no amount will be returned, only the conversion rate.

12.2.21CardRateForDCC and MerchantRateForDynamicPricing

Both elements are of the same xml-type InquiryRateType, therefore their substructure is exactly the same and described only once.

Path/Name	XML Schema type	Description
<InquiryRateType>/ v1: InquiryRateId	xs:long	The Store ID. The base currency is derived from the Store's settings.
<InquiryRateType>/ v1:ForeignCurrencyCode	xs:string	The currency that the amount has been converted to (ISO_4217 format)
<InquiryRateType>/ v1:ForeignAmount	xs:decimal	The converted amount.
<InquiryRateType>/ v1:ExchangeRate	xs:decimal	The exchange rate of the currency conversion
<InquiryRateType>/ v1:DccApplied	xs:boolean	Whether the user accepted the DCC offering or not.
<InquiryRateType>/ v1:DccOffered	xs:boolean	Whether an offering for dynamic currency conversion was extended
<InquiryRateType>/ v1:ExpirationTimestamp	xs:dateTime	Timestamp after which this DCC offering expires
<InquiryRateType>/ v1:MarginRatePercentage	xs:decimal	Optional margin information.
<InquiryRateType>/ v1:ExchangeRateSourceName	xs:string	The source of the currency conversion.
<InquiryRateType>/ v1:ExchangeRateSourceSourceTimestamp	xs:dateTime	The timestamp when the source has done the currency conversion

Note:Instead of <InquiryRateType> substitute either CardRateForDCC or MerchantRateForDynamicPricing

12.2.22MCC 6012 Visa and Mastercard Mandate

For UK-based Financial Institutions with Merchant Category Code 6012, Visa and Mastercard have mandated additional information of the primary recipient of the loan to be included in the authorization message.
If you are a UK 6012 merchant use the following parameters for your transaction request:

Path/Name	XML Schema type	Description
v1:MCC6012Details/ v1:BirthDate	xs:string	Date of birth in format YYYYMMDD
v1:MCC6012Details/ v1:AccountFirst6	xs:string	First 6 digits of recipient PAN (where the primary recipient account is a card)
v1:MCC6012Details/ v1:AccountLast4	xs:string	Last 4 digits of recipient PAN (where the primary recipient account is a card)
v1:MCC6012Details/ v1:AccountNumber	xs:string (max 50)	Recipient account number (where the primary recipient account is not a card)
v1:MCC6012Details/ v1:PostCode	xs:string (max 50)	Post Code
v1:MCC6012Details/ v1:Surname	xs:string (max 100)	Surname

If you are a UK merchant with Merchant Category Code 6051 and 7299, you can optionally use the same MCC6012 parameters in your request for debt repayment transactions.

12.2.23Market Segment Addendum

Card transactions in specific market segments can obtain incentive rates when they include addendum data.
The Web Service API allows you to submit addendum data for the following industries:

Airlines (MCC 3000-3299 or 4511)	v1:AirlineDetails, v1: TravelRoute
Car Rental (MCC 3351-3500, 7512, 7513 or 7519	v1:CarRental
Hotel Lodgings (MCC 3501-3999 or 7011)	v1:HotelLodgings

Please see v1.xsd for details (link in Appendix).

12.2.24SCA Exemptions

Following PSD2 mandate requirements you are able to request an exemption from Strong Customer Authentication (SCA) with including one of the available SCAExemptionIndicators in your transaction request to the Gateway.

v1:SCAExemptionIndicator/ Low Value Exemption	Used for transaction amounts below 30 EUR or respective value in other European currencies.
v1:SCAExemptionIndicator/ Trusted Merchant Exemption	Used for cases where merchant has been flagged as trusted by their customers.

Note: PSD2 mandate is only applicable for European distribution channels.

13

13 Custom Parameters

You can send up to ten additional parameters as individual key-value pairs. The values will be stored so that they can be returned in Inquiry Actions and be visible in the Virtual Terminal's Order Details view.

Please refer to the element *AdditionalRequestParamaters* in the XSD.

14 Building a SOAP Request Message

After building your transaction in XML, a SOAP request message describing the Web Service operation call, you wish to perform, has to be created. That means while the XML-encoded transaction you have established as described in the previous chapter represents the operation argument, the SOAP request message encodes the actual operation call.
Building such a SOAP request message is a rather straightforward task. The complete SOAP message wrapping the XML-*Sale*-transaction looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <ipgapi:IPGApiOrderRequest
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
                  <v1:Transaction>
                        <v1:CreditCardTxType>
                              <v1:Type>sale</v1:Type>
                        </v1:CreditCardTxType>
                        <v1:CreditCardData>
                              <v1:CardNumber>
                                    4111********1111
                              </v1:CardNumber>
                              <v1:ExpMonth>12</v1:ExpMonth>
                              <v1:ExpYear>07</v1:ExpYear>
                        </v1:CreditCardData>
                        <v1:Payment>
                              <v1:ChargeTotal>19.00</v1:ChargeTotal>
                              <v1:Currency>978</v1:Currency>
                        </v1:Payment>
                  </v1:Transaction>
            </ipgapi:IPGApiOrderRequest>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In short, the SOAP request message contains a SOAP envelope consisting of a header and a body. While no specific header entries are required for calling the Web Service, the SOAP body takes the transaction XML document as sub element as shown above. Note that there are no further requirements for transactions of a type other than Sale. That means the general format of the SOAP request message regardless of the actual transaction type is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <ipgapi:IPGApiOrderRequest
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1">
                  <v1:Transaction>
                        <!-- transaction content -->
```

```
                </v1:Transaction>
            </ipgapi:IPGApiOrderRequest>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Finally, you may have noticed that there are no specific entries describing which Web Service operation to call. In fact, the First Data Gateway automatically maps the ipgapi:IPGApiOrderRequest element to the corresponding Web Service operation.

15 Reading the SOAP Response Message

The SOAP response message may be understood as the Web Service operation result. Hence, processing the SOAP request message may have either resulted in a SOAP response message in the success case (i.e. the return parameter) or a SOAP fault message in case of a failure (i.e. the thrown exception). Both SOAP message types are contained in the body of the HTTP response message.

15.1 SOAP Response Message

A SOAP response message is received as the result to the credit card processor (started by the First Data gateway) having approved your transaction. It always has the following scheme:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
                  <!-- transaction result -->
            </ipgapi:IPGApiOrderResponse>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If you have send an Action, you get an ipgapi:IPGApiActionResponse.

Again, no headers are defined. The SOAP body contains the actual transaction result contained in the ipgapi:IPGApiOrderResponse or ipgapi:IPGApiOrderRequest element. Its sub elements and their meanings are presented in the next chapter. However, in order to provide a quick example, an approved *Sale* transaction is wrapped in a SOAP message similar to the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
                  <ipgapi:CommercialServiceProvider>
                        BNLP
                  </ipgapi:CommercialServiceProvider>
                  <ipgapi:TransactionTime>
                        1192111687392
                  </ipgapi:TransactionTime>
                  <ipgapi:ProcessorReferenceNumber>
                        3105
                  </ipgapi:ProcessorReferenceNumber>
                  <ipgapi:ProcessorResponseMessage>
                        Function performed error-free
                  </ipgapi:ProcessorResponseMessage>
                  <ipgapi:ErrorMessage />
                  <ipgapi:OrderId>
                        62e3b5df-2911-4e89-8356-1e49302b1807
```

```
                    </ipgapi:OrderId>
                    <ipgapi:ApprovalCode>
                          Y:440368:0000057177:PPXM:0043364291
                    </ipgapi:ApprovalCode>
                    <ipgapi:AVSResponse>PPX</ipgapi:AVSResponse>
                    <ipgapi:TDate>1192140473</ipgapi:TDate>
                    <ipgapi:TransactionResult>
                          APPROVED
                    </ipgapi:TransactionResult>
                    <ipgapi:TerminalID>123456</ipgapi:TerminalID>
                    <ipgapi:ProcessorResponseCode>
                          00
                    </ipgapi:ProcessorResponseCode>
                    <ipgapi:ProcessorApprovalCode>
                          440368
                    </ipgapi:ProcessorApprovalCode>
                    <ipgapi:ProcessorReceiptNumber>
                          4291
                    </ipgapi:ProcessorReceiptNumber>
                    <ipgapi:ProcessorTraceNumber>
                          004336
                    </ipgapi:ProcessorTraceNumber>
              </ipgapi:IPGApiOrderResponse>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

15.2 SOAP Fault Message

In general, a SOAP fault message returned by the Web Service API has the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <SOAP-ENV:Fault>
                  <faultcode>SOAP-ENV:Client</faultcode>
                  <faultstring xml:lang="en-US">
                        <!-- fault message -->
                  </faultstring>
                  <detail>
                        <!-- fault message -->
                  </detail>
            </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Basically, the faultstring element carries the fault type. According to the fault type, the other elements are set. Note that not all of the above shown elements have to occur within the SOAP-ENV:Fault element. Which elements exist for which fault type is described in the upcoming sections.

15.3 SOAP-ENV:Server

In general, this fault type indicates that the Web Service has failed to process your transaction due to an internal system error. If you receive this as response, please contact our support team to resolve the problem.

An *InternalException* always looks like the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <SOAP-ENV:Fault>
                  <faultcode>SOAP-ENV:Server</faultcode>
                  <faultstring xml:lang="en-US">
                        unexpected error
                  </faultstring>
            </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP fault message elements – relative to the SOAP-ENV:Envelope/SOAP-ENV:Body/SOAP-ENV:Fault element – are set as follows:

Path/Name	XML Schema type	Description
faultcode	xs:string	This element is always set to SOAP-ENV:Server, indicating that the fault cause is due to the system underlying the API having failed.
faultstring	xs:string	This element always carries the following fault string: unexpected error

15.4 SOAP-ENV:Client

15.4.1 MerchantException

This fault type occurs if the Gateway can trace back the error to your store having passed incorrect information. This may have one of the following reasons:

1. Your store is registered as being closed. In case you will receive this information despite your store being registered as open, please contact support.
2. The store ID / user ID combination you have provided for HTTPS authorization is syntactically incorrect.
3. The XML does not match the schema.

A *MerchantException* always looks as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <SOAP-ENV:Fault>
                  <faultcode>SOAP-ENV:Client</faultcode>
                  <faultstring xml:lang="en-US">
                        MerchantException
                  </faultstring>
                  <detail>
                        <!-- detailed explanation. -->
                  </detail>
            </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP fault message elements – relative to the SOAP-ENV:Envelope/SOAP-ENV:Body/SOAP-ENV:Fault element – are set as follows:

Path/Name	XML Schema type	Description

faultcode	xs:string	This element is always set to SOAP-ENV:Client
faultstring	xs:string	This element is always set to MerchantException
detail/reason	xs:string	Minimum one reason

See section Merchant Exceptions in the Appendix for detailed analysis of errors.

15.4.2 ProcessingException

A fault of this type is raised whenever the Gateway has detected an error while processing your transaction. The difference to the other fault types is that the transaction passed the check against the xsd.

A *ProcessingException* always looks as shown below:

```
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body>
            <SOAP-ENV:Fault>
                  <faultcode>SOAP-ENV:Client</faultcode>
                  <faultstring xml:lang="en-US">
                        ProcessingException: Processing the request
                        resulted in an error - see SOAP details for more
                        information
                  </faultstring>
                  <detail>
                        <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="https://ipg-online.com/ipgapi/schemes/ipgapi">
                              <ipgapi:CommercialServiceProvider>
                                    BNLP
                              </ipgapi:CommercialServiceProvider>
                              <ipgapi:TransactionTime>
                                    1192111156423
                              </ipgapi:TransactionTime>
                              <ipgapi:ProcessorReferenceNumber />
                              <ipgapi:ProcessorResponseMessage>
                                    Card expiry date exceeded
                              </ipgapi:ProcessorResponseMessage>
                              <ipgapi:ErrorMessage>
                                    SGS-000033: Card expiry date exceeded
                              </ipgapi:ErrorMessage>
                              <ipgapi:OrderId>
                                    62e3b5df-2911-4e89-8356-1e49302b1807
                              </ipgapi:OrderId>
                              <ipgapi:ApprovalCode />
                              <ipgapi:AVSResponse />
                              <ipgapi:TDate>1192139943</ipgapi:TDate>
                              <ipgapi:TransactionResult>
                                    FAILED
                              </ipgapi:TransactionResult>
                              <ipgapi:TerminalID>123456</ipgapi:TerminalID>
                              <ipgapi:ProcessorResponseCode/>
                              <ipgapi:ProcessorApprovalCode />
                              <ipgapi:ProcessorReceiptNumber />
                              <ipgapi:ProcessorTraceNumber />
                        </ipgapi:IPGApiOrderResponse>
                  </detail>
            </SOAP-ENV:Fault>
```

```
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP fault message elements – relative to the SOAP-ENV:Envelope/SOAP-ENV:Body/SOAP-ENV:Fault element – are set as described below.

Path/Name	XML Schema type	Description
faultcode	xs:string	This element is always set to SOAP-ENV:Client, indicating that the fault cause is likely to be found in invalid transaction data having been passed.
faultstring	xs:string	This element always carries the following fault string: ProcessingException
detail/ ipgapi:IPGApiOrderResponse	Composite element	This element contains the error. Since there are numerous causes for raising such an exception, the next chapter will give an overview by explaining the data contained in this element.

See section Processing Exceptions in the Appendix for detailed analysis of errors.

16 Analysing the Transaction Result

16.1 Transaction Approval

The SOAP message wrapping a transaction approval has been presented in the previous chapter together with an example. The transaction status report generated by the Gateway is contained in the ipgapi:IPGApiOrderResponse element and can be understood as the data returned by the Web Service operation. In the following, its elements – relative to the ipgapi:IPGApiOrderResponse super element – are described. Note that always the full set of elements is contained in the response – however, some elements might be empty.

Path/Name	XML Schema type	Description
ipgapi: CommercialServiceProvider	xs:string	Indicates your provider.
ipgapi:TransactionTime	xs:string	The time stamp which is set by the Gateway before returning the transaction approval.
ipgapi: ProcessorReferenceNumber	xs:string	In some cases, this element might be empty. It stores a number allowing the credit card processor to refer to this transaction. You do not need to provide this number in any further transaction. However, have that number ready, in case you detect any problems with your transaction and you want to contact support.
ipgapi: ProcessorResponseMessage	xs:string	In case of an approval, this element contains either contains the response message provided by the authorisation system (e.g. an auth

		code) or in case there is no such message, the string: Function performed error-free
ipgapi: ProcessorResponseCode	xs:string	The response code from the credit card processor
ipgapi:ErrorMessage	xs:string	This element is empty in case of an approval.
ipgapi:OrderId	xs:string	This element contains the order ID. For *Sale*, *PreAuth*, *ForceTicket*, and *Credit* transactions, a new order ID is returned. For *PostAuth*, *Return*, and *Void* transactions, supply this number in the v1:OrderId element for making clear to which transaction you refer. The ipgapi:OrderId element of a transaction approval to a *PostAuth*, *Return*, or *Void* transaction simply returns the order ID, such a transaction has referred to.
ipgapi:ApprovalCode	xs:string	Stores the approval code the transaction processor has created for this transaction. You do not need to provide this code in any further transaction. However, have that number ready, in case you detect any problems with your transaction and you want to contact support.
ipgapi:AVSResponse	xs:string	Returns the address verification system (AVS) response.
ipgapi:TDate	xs:string	Stores the TDate you have to supply when voiding this transaction (which is only possible for *Sale* and *PostAuth* transactions). In this case, pass its value in the v1:TDate element of the *Void* transaction you want to build.
ipgapi:TransactionResult	xs:string	Stores the transaction result which is always set to APPROVED in case of an approval or WAITING in case the final result is not yet clear and will be updated at a later point.
ipgapi:TerminalID	xs:string	The Terminal ID used for this transaction.
ipgapi:PaymentType	xs:string	The payment type used for this transaction.
ipgapi:Brand	xs:string	The brand of the card used for this transaction.
ipgapi:ConvenienceFee	xs:decimal	The Convenience fee value, returned in the response if you have this feature configured and this is applicable for Sale transaction.
ipgapi:Country	xs:string	The country where the card has been issued that has been used for this transaction.

ipgapi:SchemeTransactionId	xs:string	Returned in the response by Mastercard for stored credentials transactions.

16.2 Transaction Failure

As shown in the previous chapter, a SOAP fault message, resulting from the credit card processor having failed to process your transaction, contains an ipgapi:IPGApiOrderResponse element passed as child of a SOAP detail element. Note that its sub elements are exactly the same as in the transaction approval case. Their meaning in the failure case is described below:

Path/Name	XML Schema type	Description
ipgapi: CommercialServiceProvider	xs:string	Indicates your provider.
ipgapi:TransactionTime	xs:string	The time stamp which is set by the Gateway before returning the transaction failure. The format is Unix time (https://en.wikipedia.org/wiki/Unix_time).
ipgapi: ProcessorReferenceNumber	xs:string	In some cases, this element might be empty. Stores a number allowing the credit card processor to refer to this transaction. You do not need to provide this number in any further transactions. However, have that number ready, in case you detect any problems with your transaction and you want to contact support.
ipgapi: ProcessorResponseMessage	xs:string	Stores the error message the credit card processor has returned. For instance, in case of an expired credit card this might be: Card expiry date exceeded
ipgapi: ProcessorResponseCode	xs:string	The response code from the credit card processor
ipgapi: ProcessorApprovalCode	xs:string	The approval code from the credit card processor
ipgapi: ProcessorReceiptNumber	xs:string	The receipt number from the credit card processor
ipgapi: ProcessorTraceNumber	xs:string	The trace number from the credit card processor
ipgapi:ErrorMessage	xs:string	Stores the error message returned by the Gateway. It is always encoded in the format *SGS-XXXXXX: Message* with *XXXXXX* being a six digit error code and *Message* describing the error (this description might be different from the processor response message). For instance, in the above example the error message SGS-000033: Card expiry date exceeded is returned. Make sure to have the error code and message ready when contacting support.

ipgapi:OrderId	xs:string	Stores the order ID. In contrast to an approval, this order ID is never required for any further transaction, but needed for tracing the cause of the error. Hence, make sure to have it ready when contacting support.
ipgapi:ApprovalCode	xs:string	This element is empty in case of a transaction failure.
ipgapi:AVSResponse	xs:string	Returns the address verification system (AVS) response.
ipgapi:TDate	xs:string	Stores the TDate. Similar to the order ID, the TDate is never required for any further transaction, but needed for tracing the error cause. Hence, make sure to have it ready when contacting support.
ipgapi:TransactionResult	xs:string	In the failure case, there are three possible values: • DECLINED • FRAUD • FAILED DECLINED is returned in case the credit card processor does not accept the transaction, e.g. when finding the customer's funds not to be sufficient. FRAUD is returned in case a fraud attempt is assumed by the Gateway. If an internal gateway error should occur, the returned value is FAILED.
ipgapi:TerminalID	xs:string	The Terminal ID used for this transaction.

17 Building an HTTPS POST Request

Building an HTTPS POST request is a task you rarely have to do "by hand". There are plenty of tools and libraries supporting you in the composition of HTTPS requests. Mostly, the required functionality for doing this task is contained in the standard set of libraries coming with the technological environment in which you develop your online store.
Since all of these libraries slightly differ in their usage, no general building process can be described. In order to illustrate the basic concepts, the following chapters will give examples showing how to build a valid HTTPS request in PHP and ASP. In general, the set of parameters you have to provide for building a valid HTTPS request in whatever technology is as follows:

Parameter	Value	Description
URL	https:// test.ipg-online.com/ ipgapi/services	This is the full URL of the Web Service API – depending on the functionality you use for building HTTP requests, you might have to split this URL into host and service and provide this information in the appropriate HTTP request headers. Please note, that only TLS secured communication over standard HTTPS TCP port 443 is accepted.

Content-Type	text/xml	This is an additional HTTP header needed to be set. This is due to the SOAP request message being encoded in XML and passed as content in the HTTP POST request body.
Authorization	Type: Basic Username: WS*storeID*._.*userID* Password: *yourPassword*	Your store is identified at the Gateway by checking these credentials. In order to use the Web Service API, you have to provide your store ID, user ID, and password as the content of an HTTP *Basic* authorization header. For instance, if your store ID is 101, your user ID 007, and your password myPW, the authorization user name is WS101._.007. The complete HTTP authorization header would be: Authorization: Basic V1MxMDEuXy4wMDc6bXlQVw== Note that the latter string is the base 64 encoding result of the string WS101._.007:myPW.
HTTP Body	SOAP request XML	The HTTP POST request body takes the SOAP request message

Please note, that the Gateway is using GSLB (Global Server Load Balancing) solution to route traffic to different locations. By default, DNS returns IP address of a primary datacenter. This may change during planned maintenance or unplanned outage – in such case a different IP is returned, pointing to DR (disaster recovery) location. It is therefore critical, that you respect IP address and TTL returned by DNS. Please consider that while setting up firewalls, proxy whitelists etc.

17.1 PHP

Doing HTTP communication in PHP is mostly accomplished with the aid of cURL which is shipped both as library and command line tool. In newer PHP versions, cURL is already included as extension which has to be “activated”, thus making the cURL functionality available in any PHP script. While this is a rather straightforward task in case your Web server operates on Microsoft Windows, it might require to compile PHP on Unix/Linux machines. Therefore, you might consider to call the cURL command line tool from your PHP script instead of using the cURL extension. Both variants are considered in the following beginning with the usage of the cURL extension in PHP 5.2.4 running on a Windows machine.

17.1.1 Using the cURL PHP Extension

Mostly, activating the cURL extension in PHP 5.2.4 simply requires to uncomment the following line in your *php.ini* configuration file:

```
;extension=php_curl.dll
```

Note that other PHP versions might require other actions in order to enable cURL support in PHP. Refer to your PHP documentation for more information. After activating cURL, an HTTP request with the above parameters is set up with the following PHP statements:

```
<?php
// storing the SOAP message in a variable – note that the plain XML code
// is passed here as string for reasons of simplicity, however, it is
// certainly a good practice to build the XML e.g. with DOM – furthermore,
// when using special characters, you should make sure that the XML string
// gets UTF-8 encoded (which is not done here):
$body = "<SOAP-ENV:Envelope ...>...</SOAP-ENV:Envelope>";
// initializing cURL with the IPG API URL:
```

```
$ch = curl_init("https://test.ipg-online.com/ipgapi/services");
// setting the request type to POST:
curl_setopt($ch, CURLOPT_POST, 1);
// setting the content type:
curl_setopt($ch, CURLOPT_HTTPHEADER, array("Content-Type: text/xml"));
// setting the authorization method to BASIC:
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
// supplying your credentials:
curl_setopt($ch, CURLOPT_USERPWD, "WS101._.007:myPW");
// filling the request body with your SOAP message:
curl_setopt($ch, CURLOPT_POSTFIELDS, $body);
...
?>
```

Setting the security options which are necessary for enabling TLS communication will be discussed in the next chapter extending the above script.

17.1.2 Using the cURL Command Line Tool

For the reasons described above, you might consider using the cURL command line tool instead of the extension. Using the tool does not require any PHP configuration efforts – your PHP script simply has to call the executable with a set of parameters. Since the security settings are postponed to the next chapter, the following script only shows how to set up the standard HTTP parameters, i.e. the script is extended with the TLS parameters in the next chapter.

```
<?php
// storing the SOAP message in a variable – note that you have to escape
// " and \n, since the latter makes the command line tool fail,
// furthermore note that the plain XML code is passed here as string
// for reasons of simplicity, however, it is certainly a good practice
// to build the XML e.g. with DOM – finally, when using special
// characters, you should make sure that the XML string gets UTF-8 encoded
// (which is not done here):
$body = "<SOAP-ENV:Envelope ...>...</SOAP-ENV:Envelope>";
// setting the path to the cURL command line tool – adapt this path to the
// path where you have saved the cURL binaries:
$path = "C:\curl\curl.exe";
// setting the IPG API URL:
$apiUrl = " https://test.ipg-online.com/ipgapi/services";
// setting the content type:
$contentType = " --header \"Content-Type: text/xml\"";
// setting the authorization method to BASIC and supplying
// your credentials:
$user = " --basic --user WS101._.007:myPW";
// setting the request body with your SOAP message – this automatically
// marks the request as POST:
$data = " --data \"".$body."\"".
...
?>
```

17.2 ASP

There are multiple ways of building an HTTP request in ASP. However, in the following, the usage of WinHTTP 5.1 is described as it ships with Windows Server 2003 and Windows XP SP2. Furthermore, only a few lines of code are required in order to set up a valid HTTP request. Note that the following code fragment is written in JavaScript. Using VB Script instead does not fundamentally change the shown statements.

```
<%@ language="javascript"%>
<html>...<body>
<%
// storing the SOAP message in a variable – note that the plain XML code
```

```
// is passed here as string for reasons of simplicity, however, it is
// certainly a good practice to build the XML e.g. with DOM – furthermore,
// when using special characters, you should make sure that the XML string
// gets UTF-8 encoded (which is not done here):
var body = "<SOAP-ENV:Envelope ...>...</SOAP-ENV:Envelope>";
// constructing the request object:
var request = Server.createObject("WinHttp.WinHttpRequest.5.1");
// initializing the request object with the HTTP method POST
// and the IPG API URL:
request.open("POST", "https://test.ipg-online.com/ipgapi/services");
// setting the content type:
request.setRequestHeader("Content-Type", "text/xml");
// setting the credentials:
request.setCredentials("WS10036000750._.1001", "testinger", 0);
...
%>
</body></html>
```

Note that the above script is extended in the next chapter by setting the security options which are required for establishing the TLS channel.

18 Establishing a TLS connection

Before sending the HTTP request built in the previous chapter, a secure communication channel has to be established, guaranteeing both that all data is passed encrypted and that the client (your application) and server (running the Web Service API) can be sure of communicating with each other and no one else.
Please note, that only TLS secured communication over standard HTTPS TCP port 443 is accepted.

Both are achieved by establishing an TLS connection with the client and server exchanging certificates. A certificate identifies a communication party uniquely. Basically, this process works as follows:

1. TLS: The client requests access to www.ipg-online.com
2. TLS: The server presents its certificate to the client
3. TLS: The client verifies the server's certificate (optional)
4. TLS: The server asks the client for a client certificate
5. TLS: The client sends its certificate to the server
6. TLS: The server verifies the client's credentials
7. TLS: If successful, the server establishes TLS tunnel to www.ipg-online.com and all the data exchanged between parties is encrypted.
8. HTTP: Start HTTP and request the URL part: /ipgapi/services […]

Following this process, your application has to do two things: First, start the communication by sending its client certificate. Second, verify the received server certificate. How this is accomplished differs from platform to platform. However, in order to illustrate the basic concepts, the PHP and ASP scripts started in the previous chapter will be continued by extending them with the relevant statements necessary for setting up a TLS connection.

18.1 PHP

Picking up the distinction between using either the PHP cURL extension or the command line tool, the following two sections will continue the two different ways of enabling secure HTTP communication. However, regardless of which approach you intend to use, you will be confronted with one special feature of cURL: cURL requires the client certificate to be passed as PEM file with the Client Certificate Private Key passed in an extra file. Finally, the Client Certificate Private Key password has to be supplied. Simply spoken, the PEM file contains the list of client certificates with all information necessary for allowing the server to identify the client. The private key is not really necessary for this kind of communication. However, it is crucial for making cURL work.

18.1.1 Using the PHP cURL Extension

Building on the script started in the previous chapter, the parameters which are necessary for establishing an TLS connection with cURL are set in the following statements:

```
<?php
...
// telling cURL to verify the server certificate:
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 1);
// setting the path where cURL can find the certificate to verify the
// received server certificate against:
curl_setopt($ch, CURLOPT_CAINFO, "C:\certs\tlstrust.pem");
// setting the path where cURL can find the client certificate:
curl_setopt($ch, CURLOPT_SSLCERT, "C:\certs\WS101._.007.pem");
// setting the path where cURL can find the client certificate's
// private key:
curl_setopt($ch, CURLOPT_SSLKEY, "C:\certs\WS101._.007.key");
// setting the key password:
curl_setopt($ch, CURLOPT_SSLKEYPASSWD, "ckp_1193927132");
...
?>
```

Note that this script is extended in the next chapter by the statements doing the actual HTTP request.

18.1.2 Using the cURL Command Line Tool

Building on the script started in the previous chapter, the statements which initialize the TLS parameters passed to the cURL command line tool are as follows:

```
<?php
...
// setting the path where cURL can find the certificate to verify the
// received server certificate against:
$serverCert = " --cacert C:\certs\tlstrust.pem";
// setting the path where cURL can find the client certificate:
$clientCert = " --cert C:\certs\WS101._.007.pem";
// setting the path where cURL can find the client certificate's
// private key:
$clientKey = " --key C:\certs\WS101._.007.key";
// setting the key password:
$keyPW = " --pass ckp_1193927132";
...
?>
```

Note that this script is extended in the next chapter by the statements doing the actual HTTP request.

18.2 ASP

For making the above TLS initialization process work, ASP requires both the client and the server certificate to be present in certificate stores. In other words, before ASP can communicate via TLS, both certificates have to be installed first. The following steps which assume ASP running on Microsoft IIS 5.1 under Windows XP, will guide you through this set up process:

1. Click Start, click *Run...*, type *mmc* and click *OK*.
2. Open the *File* menu, select *Add/Remove Snap-In*.
3. Click *Add*.
4. Under *Snap-In* choose *Certificates* and click *Add*.
5. You will be prompted to select the account for which you want to manage the certificates. Since IIS uses the computer account, choose *Computer Account* and click *Next*.
6. Choose *Local Computer* and click *Finish*.
7. Click *Close* and then *OK*.
8. Expand the *Certificates (Local Computer)* tree - the client certificate will be installed in the *Personal* folder.

9. Therefore, right click the *Certificates* folder, select *All Tasks*, click *Import...* – this will open the Certificate Import Wizard.
10. Click *Next*. Choose your client certificate p12 file and click *Next*.
11. Provide the client certificate installation password and click *Next*.
12. Select *Place all certificates in the following store* and browse for the *Personal* folder if not yet displayed. Click *Next*.
13. Check the displayed settings and click *Finish*. Your client certificate is now installed in the local computer's personal certificates store. Here, IIS (running ASP) can lookup the client certificate when communicating with another server via HTTP.
14. Now, the server certificate has to be installed in the *Trusted Root Certification Authorities* store. The certificates in this store are used for verification whenever receiving a certificate from a server. That means the Web Service API server certificate has to be installed here. In this way, IIS is able to verify the server certificate received when contacting the Web Service. Therefore, choose *Trusted Root Certification Authorities* from the *Certificates (Local Computer)* tree open the sub folder *Certificates*.
15. Right click the *Certificates* folder, select *All Tasks*, click *Import...* – this will open the Certificate Import Wizard again.
16. Click *Next*. Choose the Trust Anchor PKCS#7 file and click *Next*.
17. Select *Place all certificates in the following store* and browse for the *Trusted Root Certification Authorities* folder if not yet displayed. You should trust all client certificates listed to establish a trusted connection to the server. Click *Next*.
18. Check the displayed settings and click *Finish*. The server certificate is now installed in the local computer's trusted certificates store. Here, IIS can lookup the server certificate for verification against the Web Service API server certificate received during the TLS setup process.

After installing both certificates one could assume that the environment allowing ASP to communicate via TLS is set up. However, there is still one thing which makes the communication fail: IIS – running your ASP – has a Windows user which does not have the necessary rights to access the client certificate private key. Although accessing the private key is not really necessary for establishing the TLS connection to the Gateway, the IIS user needs access rights for running the authentication process in ASP. For granting rights to a user, Microsoft provides the *WinHttpCertCfg.exe* tool you can download for free under:

http://www.microsoft.com/downloads/details.aspx?familyid=c42e27ac-3409-40e9-8667-c748e422833f&displaylang=en

After installing the tool, open a command prompt, switch to the directory where you have installed the tool, and type in the following line for granting access to the IIS user:

winhttpcertcfg -g -c LOCAL_MACHINE\My -s WS101._.007 -a IWAM_MyMachine

LOCAL_MACHINE\My determines the key store where the personal certificates for the local machine account are stored. After installing the client certificate in the personal certificates store as described above, the client certificate can be found under this path, so there is no need to provide another path. WS101._.007 is the name of the client certificate. You have to adapt this name to the name of your client certificate. Therefore, check the name displayed for the client certificate in the mmc console after installing it as described above. Finally, IWAM_MyMachine denotes the IIS user name. Note that IIS 5.1 uses IWAM_MachineName by default. That means if your machine has the name IISServerMachine, the IIS user will be called IWAM_IISServerMachine. Note that other IIS versions might use a different naming scheme. If you do not know your machine name or IIS user name, check the IIS documentation and contact your administrator.

Now you are ready to use TLS in your ASP code. The code extending the ASP script started in the previous chapter is reduced to only one additional statement which tells WinHTTP which client certificate to send (and where to find it) when contacting the First Data Gateway:

```
<%@ language="javascript"%>
<html>...<body>
<%
...
```

```
// setting the path where the client certificate to send can be found:
request.setClientCertificate("LOCAL_MACHINE\\My\\WS101._.007");
...
%>
</body></html>
```

Note that if you use VB Script, the code looks almost the same – however, do not forget to replace the doubled backslashes in the path with single ones (i.e. the path to the certificate would be "LOCAL_MACHINE\My\WS101._.007" instead).

Note that this script is extended in the next chapter by the statements doing the actual HTTP request.

19 Sending the HTTPS POST Request and Receiving the Response

The actual communication with the Web Service API takes place when sending the HTTPS request and waiting for a response. Again, how this is done depends on the technology you are using. Most HTTP libraries fully cover the underlying communication details and reduce this process to a single operation call returning the HTTP response as result object.
In any case, the parameters which are required for successfully performing an HTTP POST request over TLS and receiving the response (carrying a 200 HTTP status code) have been described in the previous two chapters. Setting invalid or incorrect parameters results in the web server running the Web Service API to return a standard HTTP error code in the HTTP header of the response or sending an TLS failure. Their meanings can be found in any HTTP/TLS guide.
Please note, that only TLS secured communication over standard HTTPS TCP port 443 is accepted.

However, there is one important exception: In case the HTTP parameters you have provided are correct, but the Web Service has failed to process your transaction due to an incorrect value contained in the SOAP request message (e.g. an invalid credit card number), a SOAP exception is thrown and transferred in the body of an HTTP response carrying the error code 500. Details about the exception cause are provided in the SOAP fault message which is described in the context of the next chapter.

In order to complete the PHP and ASP scripts, built gradually in the previous chapters, the following two chapters will provide the statements necessary for doing an HTTP call using these technologies.

19.1 PHP

Again, the distinction between the PHP cURL extension and the cURL command line tool is made in the following:

19.1.1 Using the PHP cURL Extension

The PHP script using the cURL extension is finally completed by doing the call with the statements shown below. Note that the HTTP call returns a SOAP response or fault message in the HTTP response body.

```
<?php
...
// telling cURL to return the HTTP response body as operation result
// value when calling curl_exec:
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
// calling cURL and saving the SOAP response message in a variable which
// contains a string like "<SOAP-ENV:Envelope ...>...</SOAP-ENV:Envelope>":
$result = curl_exec($ch);
// closing cURL:
curl_close($ch);
?>
```

19.1.2 Using the cURL Command Line Tool

Doing the HTTP call with the cURL command line tool simply requires completing the command line statement and executing the external tool. However, reading the HTTP response is more complicated

as the PHP exec command saves each line returned by an external program as one element of an array. Concatenating all elements of that array results in the SOAP response or fault message which has been returned in the HTTP response body. The following statements handle the HTTP call and complete the script:

```
<?php
...
// saving the whole command in one variable:
$curl = $path.
        $data.
        $contentType.
        $user.
        $serverCert.
        $clientCert.
        $clientKey.
        $keyPW.
        $apiUrl;
// preparing the array containing the lines returned by the cURL
// command line tool:
$returnArray = array();
// performing the HTTP call by executing the cURL command line tool:
exec($curl, $returnArray);
// preparing a variable taking the complete result:
$result = "";
// concatenating the different lines returned by the cURL command
// line tool – this result in the variable $result carrying the entire
// SOAP response message as string:
foreach($returnArray as $item)
        $result = $result.$item;
?>
```

19.2 ASP

Doing the actual HTTP call with WinHTTP in ASP is limited to one simple operation call taking the SOAP request XML as a parameter. After successfully performing the request a SOAP response or fault message is returned which can be retrieved as a string by accessing the request object's responseText property. How such a SOAP response message looks like is described in the next chapter. The following statements complete the ASP script:

```
 <%@ language="javascript"%>
<html>...<body>
<%
...
// doing the HTTP call with the SOAP request message as input:
request.send(body);
// saving the SOAP response message in a string variable:
var response = request.responseText;
%>
</body></html>
```

20 Using a Java Client to connect to the web service

For quick and simple integration, First Data provides a Java Client to connect to the Gateway web service. An instance of the IPGApiClient class manages the connection to the web service, builds XML and the SOAP messages and evaluates the responses. To construct a transaction or to handle a response, the developer works with simple Java bean classes.

The IPGApiClient uses the apache http client. Some settings of the http client impact every http client for the same class loader environment.

20.1 Instance an IPGApiClient

There are several constructors available to instantiate the IPGApiClient. The example below illustrates how to use the easiest one of the constructors. The getBytes method is also included for the completion and simplification of the example.

```
String url = "https://test.ipg-online.com/ipgapi/services";
String storeId = "your store id";
String password = "your password";
byte[] key = getBytes("/path/to/your/keyStore.ks");
String keyPW = "your key store password";

IPGApiClient client = new IPGApiClient(url, storeId, password, key, keyPW);

/**
 * getBytes
 * reads a resource and returns a byte array
 * @param resource the resource to read
 * @return the resource as byte array
 */
public static byte[] getBytes(final String resource) throws IOException {
        final InputStream input = IO.class.getResourceAsStream(resource);
        if (input == null) {
                throw new IOException(resource);
        }
        try {
                final byte[] bytes = new byte[input.available()];
                input.read(bytes);
                return bytes;
        } finally {
                try {
                        input.close();
                } catch (IOException e) {
                        log.warn(resource);
                }
        }
}
```

20.2 How to construct a transaction and handle the response

There are different classes for transactions with the following card types:

- Credit Card
- UK Debit Cards.

The following factory class can be used to generate the class you need:

de.firstdata.ipgapi.client.transaction.IPGApiTransactionFactory

The following example shows a Credit Card Sale transaction for an amount of 7 Euros:

```
Amount amount = new Amount("7", "978"); // ISO 4217: EUR = 978
CreditCard cC = new CreditCard("1111222233334444", "07", "17", null);
CCSaleTransaction transaction =
        IPGApiTransactionFactory.createSaleTransactionCredit(amount, cC);
// some transactions may include further information e.g. the customer
transaction.setName("a name");
try {
        IPGApiResult result = client.commitTransaction(transaction);
        // now you can read the conclusion
        System.out.println(result.getOrderId());
        System.out.println(result.getTransactionTime());
        // ...
} catch (ProcessingException e) {
        // ERROR: transaction not passed
}
```

20.3 How to construct an action

The following Factory Class can be used to generate the class you need:

de.firstdata.ipgapi.client.transaction.IPGApiActionFactory

To commit an action you need to use the *commitAction* method of the IPGApiClient. The further process is similar to payment transactions.

20.4 How to connect behind a proxy

Before you use the IPGApiClient behind a proxy you must set the proxy configuration of the client with the IPGApiClient method:

```
IPGApiClient.setProxy(
        final String host, final Integer port,
        final String user, final String password,
        final String workstation, final String domain)
```

The parameters user, password, workstation and domain should be null if no identification needed. If you need to identify on a MS Windows proxy you must set the parameter domain. To identify on systems like Unix the parameter domain must be null. For more information see the apache javadoc.

After setting the proxy parameters you must call the IPGApiClient.init() method.

21 Appendix

XML

The Web Service API uses the XML standard for communication as described on

http://www.w3.org/standards/xml/core

including the specification of namespaces described on

http://www.w3.org/TR/2009/REC-xml-names-20091208/

To make the names of the XML tags unique (e.g. in IPG: IPGApiActionRequest, Action, RecurringPayment, etc.), namespaces are used.

Example:

http://ipg-online.com/ipgapi/schemas/ipgapi, http://ipg-online.com/ipgapi/schemas/a1, …

These namespaces are defined in the xsd files like
xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi".

The same namespaces must be declared in the XML files (no parsing with hardcoded namespace references), starting with keyword xmlns.

To aviod errors with the namespaces we recommend to use libaries to manage the XML messages.

In the course of future product develoment, it may be necessary that we extend the IPGApiRequest or IPGApiResponse with further members. While extending the request will have no impact on your implemented code, extending the response might cause errors if you check the response against ipgapi.xsd. We therefore recommend to deactivate the check.

XML Schemas

The definitions for the XML document building blocks can be found here:

ipgapi.xsd	https://www.ipg-online.com/ipgapi/schemas/ipgapi.xsd
v1.xsd	https://www.ipg-online.com/ipgapi/schemas/v1.xsd
a1.xsd	https://www.ipg-online.com/ipgapi/schemas/a1.xsd

Troubleshooting - Merchant Exceptions

<detail>
XML is not wellformed: Premature end of message.
</detail>

Possible Explanation:
You have sent an absolutely empty message. The message contains neither a soap message nor an IPG API message or any other characters in the http body.

<detail>
XML is not wellformed: Content is not allowed in prolog.
</detail>

Possible Explanation:
The message can't be interpreted as an XML message.

<detail>
XML is not wellformed:
XML document structures must start and end within the same entity.
</detail>

Possible Explanation:
The message starts like an XML message but the end tag of the first open tag is missing.

<detail>
XML is not wellformed:
The element type "SOAP-ENV:Body" must be terminated
by the matching end-tag "</SOAP-ENV:Body>".

</detail>

Possible Explanation:
To an open internal tag (not the top level tag) the end tag is missing. In this example the end tag </SOAP-ENV:Body> is missing.

<detail>
XML is not wellformed:
Element type "irgend" must be followed by either attribute specifications, ">" or "/>".
</detail>

Possible Explanation:
The message isn't an XML message or a correct XML message. A ">" character is missing for the tag irgend.

<detail>
XML is not wellformed:
Open quote is expected for attribute "xmlns:ns3"
associated with an element type "ns3:IPGApiOrderRequest".
</detail>

Possible Explanation:
The value of one attribute isn't enclosed in quotation marks. In IPG API attributes are only used for the name spaces.

<detail>
XML is not wellformed:
The prefix "ipgapi" for element "ipgapi:IPGApiOrderRequest"
is not bound.
</detail>

Possible Explanation:
The name space "ipgapi" isn't declared. To declare a name space use the xmlns prefix. In this case you should take
xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi" as attribute in the top level tag of the IPG API message (IPGApiOrderRequest or IPGApiActionRequest).

<detail>
XML is not wellformed:
The prefix "xmln" for attribute "xmln:ns2" associated
with an element type "ns3:IPGApiOrderRequest" is not bound.
</detail>

Possible Explanation:
To declare an own name space, only the predefined name space xmlns allowed. In this case the prefix is written as xmln and not as xmlns.

<detail>
XML is not wellformed:
Unable to create envelope from given source
because the namespace was not recognized
</detail>

Possible Explanation:

The message could be interpreted as an XML message and the enclosing soap message is correct, but the including IPG API message in the soap body has no name spaces or the name spaces are not declared correctly. The correct name spaces are described in the xsd.

<detail>
XML is not wellformed:
The processing instruction target matching "[xX][mM][lL]"
is not allowed.
</detail>

Possible Explanation:
The whole message must be a correct XML message so that the including IPG API message must not contains the xml declaration <?xml … ?>.

<detail>
Unexpected characters before XML declaration
</detail>

Possible Explanation:
The XML must start with “<?xml”. Please check, if you send an empty line or another white space character in front of the xml and remove them.

<detail>
XML is not a SOAP message:
Unable to create envelope from given source
because the root element is not named "Envelope"
</detail>

Possible Explanation:
The message seems to be a correct XML message but only soap messages are accepted. This message must be enclosed by a soap message.

<detail>
XML is not a valid SOAP message:
Error with the determination of the type.
Probably the envelope part is not correct.
</detail>

Possible Explanation:
The soap body tag is missing.

<detail>
Source object passed to "{0}" has no contents.
</detail>

Possible Explanation:
The soap body is empty. The including IPG API message is missing.

<detail>
Included XML is not a valid IPG API message:
</detail>

unsupported top level {namespace}tag "irgendwas" in the soap body. Only one of [{http://ipg-online.com/ipgapi/schemas/ipgapi}IPGApiActionRequest, {http://ipg-online.com/ipgapi/schemas/ipgapi}IPGApiOrderRequest] allowed.

</detail>

Possible Explanation:
The first tag in the including IPG API message must be one of IPGApiActionRequest or IPGApiOrderRequest tag and not the tag irgendwas. In this case this tag has no namespace.

<detail>

Included XML is not a valid IPG API message: unsupported top level {namespace}tag "{http://firstdata.de/ipgapi/schemas/ipgapi}IPGApiOrderRequest" in the soap body. Only one of [{http://ipg-online.com/ipgapi/schemas/ipgapi}IPGApiActionRequest, {http://ipg-online.com/ipgapi/schemas/ipgapi}IPGApiOrderRequest] allowed.

</detail>

Possible Explanation:
The top level tag of the included IPG API message no allowed tag. In this case the name space is wrong.

<detail>

cvc-pattern-valid:
Value '1.234' is not facet-valid with respect to pattern '([1-9]([0-9]{0,12}))?[0-9](\.[0-9]{1,2})?' for type '#AnonType_ChargeTotalAmount'
cvc-type.3.1.3:
The value '1.234' of element 'ns3:ChargeTotal' is not valid.

</detail>

Possible Explanation:
The value of a tag does not correspond with the declaration in the xsd. The value has three decimal places but the xsd only allows two.

<detail>

cvc-complex-type.2.4.a: Invalid content was found starting with element 'ns2:ExpYear'. One of '{"http://ipg-online.com/ipgapi/schemas/v1":ExpMonth}' is expected.

</detail>

Possible Explanation:
The occurrences of the tags must be corresponding to the xsd. We recommend to use the tags in the same sequence as they are declared in the xsd. In this case the tag ExpMonth is expected and not ExpYear.

```
<detail>
      <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
            <ipgapi:CommercialServiceProvider />
            <ipgapi:TransactionTime>1233656751183</ipgapi:TransactionTime>
            <ipgapi:ProcessorReferenceNumber />
            <ipgapi:ProcessorResponseMessage />
            <ipgapi:ErrorMessage>
                  SGS-C: 000003:
                  illegal combination of values for the 3DSecure:
                  (VerificationResponse, PayerAuthenticationResponse,
                  PayerAuthenticationCode) N Y null
            </ipgapi:ErrorMessage>
            <ipgapi:OrderId />
            <ipgapi:ApprovalCode />
            <ipgapi:AVSResponse />
            <ipgapi:TDate />
            <ipgapi:TransactionResult>FAILED</ipgapi:TransactionResult>
            <ipgapi:TerminalID />
            <ipgapi:ProcessorResponseCode />
            <ipgapi:ProcessorApprovalCode />
            <ipgapi:ProcessorReceiptNumber />
            <ipgapi:ProcessorTraceNumber />
      </ipgapi:IPGApiOrderResponse>
</detail>
```

Explanation:
The combination of the three values VerificationResponse, PayerAuthenticationResponse and AuthenticationValue for 3DSecure is wrong. Allowed combinations are

Verification-Response	Payer-Authentication-Response	Authentication Value	IPG 3dsecure response code	Comments
null	null	null	n/a	Transaction will be passed to auth system without any 3dsecure information No MC ECI, Visa ECI = 7
N	null	null	7	Cardholder not enrolled No MC ECI, Visa ECI = 7
N	N	null	7	Cardholder not enrolled No MC ECI, Visa ECI = 7
U	null	null	5	Unable to authenticate (DS not accessible) No MC ECI, Visa ECI = 7
Y	A	null	4	Attempt (ACS cannot tell result of authentication) MC ECI = 1, Visa ECI = 6
Y	A	x	4	Attempt (ACS cannot tell result of authentication) MC ECI = 1, Visa ECI = 6
Y	U	null	6	Unable to authenticate (ACS not accessible) No MC ECI, Visa ECI = 7
Y	Y	null	2	Auth Success (no CAAV / UCAF) MC ECI = 2, Visa ECI = 5
Y	Y	x	1	Auth Success MC ECI = 2, Visa ECI = 5

Y	N	null	3	Auth Failure (Signature verification incorrect) - IPG declines the transaction ("N:-5101:3D Secure authentication failed") No MC or Visa ECI

Other combinations not listed above will be declined by IPG with a IPG 3dsecure response code of 8 and "N:-5100:Invalid 3D Secure values".

XID (created by MPI before sending Verification request) needs to be set for VISA transactions.

The payer authentication code x means, that the value is not null.

```
<detail>
      <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
            <ipgapi:CommercialServiceProvider />
            <ipgapi:TransactionTime>1233659493267</ipgapi:TransactionTime>
            <ipgapi:ProcessorReferenceNumber />
            <ipgapi:ProcessorResponseMessage />
            <ipgapi:ErrorMessage>
                  SGS-005002:
                  The merchant is not setup to support the requested
                  service.
            </ipgapi:ErrorMessage>
            <ipgapi:OrderId>
                  IPGAPI-REQUEST-9c555d62-3850-4726-8589-5a2444c98c5d
            </ipgapi:OrderId>
            <ipgapi:ApprovalCode />
            <ipgapi:AVSResponse />
            <ipgapi:TDate />
            <ipgapi:TransactionResult>FAILED</ipgapi:TransactionResult>
            <ipgapi:TerminalID />
            <ipgapi:ProcessorResponseCode />
            <ipgapi:ProcessorApprovalCode />
            <ipgapi:ProcessorReceiptNumber />
            <ipgapi:ProcessorTraceNumber />
      </ipgapi:IPGApiOrderResponse>
</detail>
```

Explanation:
After a transaction further transactions with the same data blocked are for a configurable time span.
See User Guide Virtual Terminal for details about the fraud settings.

```
<detail>
      <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
            <ipgapi:CommercialServiceProvider />
            <ipgapi:TransactionTime>1233656752308</ipgapi:TransactionTime>
            <ipgapi:ProcessorReferenceNumber />
            <ipgapi:ProcessorResponseMessage />
            <ipgapi:ErrorMessage>
                  SGS-005009:
                  The currency is not allowed for this terminal.
            </ipgapi:ErrorMessage>
            <ipgapi:OrderId>
```

```
                  IPGAPI-REQUEST-a58f6631-eb71-49c8-bbca-23fff53252fc
            </ipgapi:OrderId>
            <ipgapi:ApprovalCode />
            <ipgapi:AVSResponse />
            <ipgapi:TDate />
            <ipgapi:TransactionResult>FAILED</ipgapi:TransactionResult>
            <ipgapi:TerminalID />
            <ipgapi:ProcessorResponseCode />
            <ipgapi:ProcessorApprovalCode />
            <ipgapi:ProcessorReceiptNumber />
            <ipgapi:ProcessorTraceNumber />
      </ipgapi:IPGApiOrderResponse>
</detail>
```

Explanation:
This is an example with US Dollar, which is no allowed currency for this store.

```
<detail>
      <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
            <ipgapi:CommercialServiceProvider />
            <ipgapi:TransactionTime>1234346305732</ipgapi:TransactionTime>
            <ipgapi:ProcessorReferenceNumber />
            <ipgapi:ProcessorResponseMessage />
            <ipgapi:ErrorMessage>
                  SGS-032000: Unknown processor error occured.
            </ipgapi:ErrorMessage>
            <ipgapi:OrderId>
                  IPGAPI-REQUEST-b3223ee5-156b-4d22-bc3f-910709d59202
            </ipgapi:OrderId>
            <ipgapi:ApprovalCode />
            <ipgapi:AVSResponse />
            <ipgapi:TDate>1234346284</ipgapi:TDate>
            <ipgapi:TransactionResult>DECLINED</ipgapi:TransactionResult>
            <ipgapi:TerminalID />
            <ipgapi:ProcessorResponseCode />
            <ipgapi:ProcessorApprovalCode />
            <ipgapi:ProcessorReceiptNumber />
            <ipgapi:ProcessorTraceNumber />
      </ipgapi:IPGApiOrderResponse>
</detail>
```

Explanation:
If your transactions are normally executed, one possible explanation is that the number of Terminal IDs assigned to your store are not sufficient for your transaction volume. Please contact our Sales team to order further Terminal IDs for load balancing.

*** About to connect() to test.ipg-online.com port 443 (#0)**
*** Trying 217.73.32.55... connected**
*** Connected to test.ipg-online.com (217.73.32.55) port 443 (#0)**
*** unable to set private key file: 'C:\API\config\WS120666668._.1.key' type PEM**
*** Closing connection #0**
curl: (58) unable to set private key file: 'C:\API\config\WS120666668._.1.key' type PEM

Explanation:
Keystore and password do not fit. Check if you used the right keystore and password. Please check if you used the **WS**<storeId>._.1.pem file. If you append .cer to the file name you can open the certificate with a double click. The certificate must be exposed for your store. Please remove the extension .cer after the check.

*** SSL certificate problem, verify that the CA cert is OK. Details:**
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
*** Closing connection #0**
curl: (60) SSL certificate problem, verify that the CA cert is OK. Details:
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
More details here: http://curl.haxx.se/docs/sslcerts.html

curl performs SSL certificate verification by default, using a "bundle" of Certificate Authority (CA) public keys (CA certs). The default bundle is named curl-ca-bundle.crt; you can specify an alternate file using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in the bundle, the certificate verification probably failed due to a problem with the certificate (it might be expired, or the name might not match the domain name in the URL).
If you'd like to turn off curl's verification of the certificate, use the -k (or --insecure) option

Explanation:
The truststore certificate is wrong. Please verify the trustore: Open the file tlstrust.pem and check that one of them matched the root of the server certificate of the Gateway.

```
<html>
	<head>
		<title>Apache Tomcat/5.5.20 - Error report</title>
		<style>
			<!--
H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;}
H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;}
H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;}
BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;}
B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;}
P {font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;}
A {color : black;}
A.name {color : black;}
HR {color : #525D76;}
			-->
		</style>
	</head>
	<body>
		<h1>HTTP Status 401 - </h1>
```

```
                <HR size="1" noshade="noshade">
                <p><b>type</b> Status report</p><p><b>message</b>
                        <u></u></p><p><b>description</b>
                        <u>This request requires HTTP authentication ().</u></p>
                <HR size="1" noshade="noshade">
                <h3>Apache Tomcat/5.5.20</h3>
        </body>
</html>
```

Explanation:
Your certificates are OK and accepted but your password or your user is wrong.#

Troubleshooting - Login error messages when using the Java Client

java.io.IOException: Keystore was tampered with, or password was incorrect

Explanation:
Your keystore password doesn't fit to the keystore or the truststore password to the truststore. You can check the password with the keytool which is a component of the JDK. You can find it in the bin directory of the JDK. For testing the password call
c:\Programme\Java\jdk1.6.0_07\bin\keytool.exe -list -v -keystore <your keystore or truststore> -storepass <your keystore or truststore password>

javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: No trusted certificate found

Explanation:
Your truststore is wrong. You can inspect your truststore with keytool, a component of the JDK. Call
c:\Programme\Java\jdk1.6.0_07\bin\keytool.exe -list -v -keystore <your truststore> -storepass <your truststore password>
and you must find the issuer Equifax
OU=Equifax Secure Certificate Authority, O=Equifax, C=US in the output. Check the MD5 and SHA1 values too.

```
<html>
        <head>
                <title>Apache Tomcat/5.5.20 - Error report</title>
                <style><!--H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-
color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-
color:#525D76;font-size:16px;} H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-
color:#525D76;font-size:14px;} BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-
color:white;} B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {font-
family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;}A {color : black;}A.name
{color : black;}HR {color : #525D76;}--></style>
        </head>
        <body>
                <h1>HTTP Status 401 -</h1>
                <HR size="1" noshade="noshade">
                <p>
                        <b>type</b>
                        Status report
                </p>
                <p>
                        <b>message</b>
                        <u></u>
```

```
                </p>
                <p>
                        <b>description</b>
                        <u>This request requires HTTP authentication ().</u>
                </p>
                <HR size="1" noshade="noshade">
                <h3>Apache Tomcat/5.5.20</h3>
        </body>
</html>
```

Explanation: Your user id or password is wrong.

Troubleshooting - .NET integration issues

SSL Handshake problems in .NET - Logging

System.Net has valuable tracing capabilities. Particularly with a complex process like an SSL handshake, these capabilities become critical to debugging.

Just add a block like on example below to the end of your app.config, run your app again and take a look at the lines being added to your bin\Debug\trace.log file.

```
<system.diagnostics>
  <trace autoflush="true"/>
  <sources>
    <source name="System.Net" maxdatasize="1024">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>
    <source name="System.Net.Sockets" maxdatasize="1024">
      <listeners>
        <add name="TraceFile"/>
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add name="TraceFile" type="System.Diagnostics.TextWriterTraceListener"
      initializeData="trace.log"/>
  </sharedListeners>
  <switches>
    <add name="System.Net" value="Verbose" />
    <add name="System.Net.Sockets" value="Verbose" />
  </switches>
</system.diagnostics>
```

SSL Handshake problems in .NET- Specific Setting

Setting `Expect100Continue` to true can help in SSL handshake error situations.
When this property is set to true, 100-Continue behavior is used. Requests that use the PUT and POST methods will add an Expect header to the request if the ‘Expect100Continue’ property is true and ContentLength property is greater than zero or the SendChunked property is true. This mechanism allows you to avoid sending large amounts of data over the network when the server, based on the request headers, intends to reject the request.
If this property is set to true, the request headers are sent to the server. If the server has not rejected the request, it sends a 100-Continue response signaling that the data can be transmitted.

HTTP connectivity – Basic authentication

Requesting Headers without User and Password leads to error 401:Unathorized.
The .NET ServicePointManager does not allow setting RequestHeaders with username and password, so that the first request ends with 401 response, then .NET submits the second part. This leads to worse performance than sending user/pw right away

HTTP Connectivity - Connection Pool usage

Do not send in parallel more connection pools than *connectionLimit* allows. The .NET ServicePoint sets 2 as the default value for the size of the connection pool (attribute *connectionLimit*). Mind this needs to be increased if more than two transactions shall be sent to IPG at the same time. The number of concurrent requests must be lower than the pool size = connection limit.