First Data

Internet Payment Gateway Integration Guide First Data API

Version 1.5 (EMEA)

First Data Internet Payment Gateway

INTEGRATION GUIDE
FIRST DATA API
VERSION 1.5 (EMEA)

Contents

Getting Support

There are different manuals available for the First Data Internet Payment Gateway. This Integration Guide First Data API will be the most helpful for integration issues.
For information about settings, customisation, reports and how to process transactions manually (by keying in the information) please refer to the First Data Virtual Terminal User Guide.

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

1 Introduction

The First Data API is an Application Programming Interface which allows you to connect your application with the First Data Internet Payment Gateway. In this way, your application is able to submit credit card 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, the First Data 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 First Data API Web Service 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 First Data API Web Service.
- **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 First Data API is SSL-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 API holds a server certificate which your application may check for making sure that it speaks to the API Web Service. Finally, your application has to do a basic authorization (user name / password) before being allowed to communicate with the Web Service. In this way, the users who are authorized to communicate with First Data API are identified. These two security mechanisms guarantee that the transaction data sent to First Data API 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 First Data API's features, the focus of this guide lies on integrating the First Data API functionality into your application. A detailed description, explaining how this is done step by step, is presented in this guide. The first chapter describes all the supported transaction types.

2 Artefacts You Need

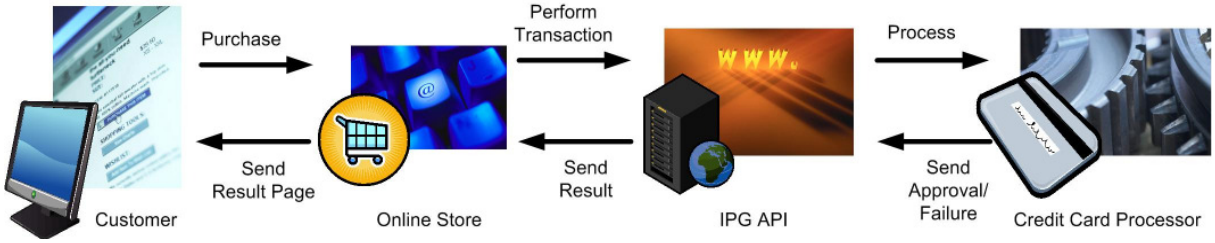
Supporting a high degree of security requires several artefacts you need for communicating securely with First Data API. Since these artefacts are referenced throughout the remainder of this guide, the following checklist shall provide an overview enabling you to make sure that you have received the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

- *Store ID:* Your store ID (e.g. `101`) which is required for the basic authorization.
- *User ID:* The user ID denoting the user who is allowed to access First Data API, e.g. `007`. Again, this is required for the basic authorization.
- *Password:* The password required for the basic authorization.
- *Client Certificate p12 File:* The client certificate 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 Internet Payment Gateway. For connecting with Java you need a ks-File, e.g.: WS101._.007.ks.
- *Client Certificate Installation Password:* The password which is required for installing the p12 client certificate 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 `WS101._.007.key`. Some tools which support you in setting up your application for using First Data API require this password when doing the client authentication at the Internet Payment Gateway.
- *Client Certificate Private Key Password:* This password protects the private key of the client certificate. Some tools which support you in setting up your application for using First Data API require this password when doing the client authentication at the Internet Payment Gateway. It follows the naming scheme `ckp_`*creationTimestamp*. For instance, this might be `ckp_1193927132`.
- *Client Certificate PEM File:* The client certificate 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 `WS101._.007.pem`. Some tools which support you in setting up your application for using the Internet Payment Gateway require this file instead of the p12 file described above.
- *Server Certificate PEM File:* The server certificate stored in the PEM file `geotrust.pem` which is required for authenticating the server running First Data API. For connecting with Java you need the truststore.ks-file.

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 First Data API. Having received the transaction, the First Data Internet Payment 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 API interface 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). In case of First Data API, a WSDL file defining the Web Service can be found at:

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

Note that you will have to supply your client certificate, your credentials, and the server certificate 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 the server 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, you have to install the server certificate. The most straightforward way to do this is to open the folder in which you have saved your server certificate PEM file and rename the file to `geotrust.crt`.
8. Then, double-click the renamed server certificate file.
9. Click *Install Certificate*. This starts the same wizard as above.
10. Click *Next*. Select *Place all certificates in the following store* and browse for the *Trusted Root Certification Authorities* folder. Click *Next*.
11. Check the displayed settings and click *Finish* (you might have to confirm the installation). The server certificate is now installed in the local computer's trusted certificates store. Here, Microsoft Internet Explorer can lookup the server certificate for verifying the First Data API server certificate received when calling the WSDL URL above.
12. Now, open a Microsoft Internet Explorer window and provide the above URL in the address field.
13. 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, First Data API sends its server certificate (identifying it uniquely) to you. This certificate is verified against the trusted one you have installed above. 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 First Data API Web Service is SSL-encrypted.

14. 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*. 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 SSL connection established in the steps above. Then, click *OK*.
15. The First Data 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 API Web Service 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 First Data API. 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 First Data 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 SSL-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 Internet Payment Gateway. It describes what happens if a customer pays with her credit card in an online store using First Data 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 SSL connection by providing the client and server certificate.
- Then, the online store sends the HTTPS request to the First Data API Web Service and waits for an HTTP response.
- First Data 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 First Data API, the SOAP message contained in the HTTP request body is parsed out. This triggers the Web Service operation handling the transaction processing to run.
- First Data API 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 SSL connection between your application and First Data API
- Send the HTTPS POST request to the First Data Internet Payment 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 First Data API Web Service 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 First Data API for encoding transactions.

5.1 Credit Card transactions

Regardless of the transaction type, the basic XML document structure of a credit 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.

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>4111111111111111</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.

5.1.2 PreAuth

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

```
<ipgapi:IPGApiOrderRequest
            xmlns:v1=":/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>4111111111111111</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 PostAuth

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>59.45</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>
```

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>4111111111111111</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:CreditCardData>
            <v1:Payment>
                  <v1:ChargeTotal>59.45</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>19.95</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>
```

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

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>4111111111111111</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:OrderId>
                        62e3b5df-2911-4e89-8356-1e49302b1807
                  </v1:OrderId>
                  <v1:TDate>1190244932</v1:TDate>
            </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.

5.2 UK Debit

Regardless of the transaction type, the basic XML document structure of a UK Debit 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:UK_DebitCardTxType>...</v1:UK_DebitCardTxType>
            <v1:UK_DebitCardData>...</v1:UK_DebitCardData>
            <v1:Payment>...</v1:Payment>
            <v1:TransactionDetails>...</v1:TransactionDetails>
            <v1:Billing>...</v1:Billing>
            <v1:Shipping>...</v1:Shipping>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

The element UK_DebitCardTXType is mandatory for all debit card transactions. The other elements depend on the transaction type. The transaction content is type-specific.

5.2.1 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:UK_DebitCardTxType>
                  <v1:Type>sale</v1:Type>
            </v1:UK_DebitCardTxType>
            <v1:UK_DebitCardData>
                  <v1:CardNumber>6799660000000000013</v1:CardNumber>
                  <v1:ExpMonth>12</v1:ExpMonth>
                  <v1:ExpYear>07</v1:ExpYear>
            </v1:UK_DebitCardData>
            <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.

5.2.2 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:UK_DebitCardTxType>
                  <v1:Type>return</v1:Type>
            </v1:UK_DebitCardTxType>
            <v1:Payment>
                  <v1:ChargeTotal>19.95</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>
```

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

5.2.3 Credit

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:UK_DebitCardTxType>
                <v1:Type>credit</v1:Type>
            </v1:UK_DebitCardTxType>
            <v1:UK_DebitCardData>
                <v1:CardNumber>6799660000000000013</v1:CardNumber>
                <v1:ExpMonth>12</v1:ExpMonth>
                <v1:ExpYear>07</v1:ExpYear>
            </v1:UK_DebitCardData>
            <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.

5.2.4 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:UK_DebitCardTxType>
                <v1:Type>void</v1:Type>
            </v1:UK_DebitCardTxType>
            <v1:TransactionDetails>
                <v1:OrderId>
                    62e3b5df-2911-4e89-8356-1e49302b1807
                </v1:OrderId>
                <v1:TDate>1190244932</v1:TDate>
            </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.

5.3 German Direct Debit

Regardless of the transaction type, the basic XML document structure of a German Direct Debit 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:DE_DirectDebitTxType>...</v1:DE_DirectDebitTxType>
            <v1:DE_DirectDebitData>...</v1:DE_DirectDebitData>
            <v1:Payment>...</v1:Payment>
            <v1:TransactionDetails>...</v1:TransactionDetails>
            <v1:Billing>...</v1:Billing>
            <v1:Shipping>...</v1:Shipping>
      </v1:Transaction>
</ipgapi:IPGApiOrderRequest>
```

The element DE_DirectDebitTXType is mandatory for all debit transactions. The other elements depend on the transaction type. The transaction content is type-specific.

5.3.1 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:DE_DirectDebitTxType>
                  <v1:Type>sale</v1:Type>
            </v1:DE_DirectDebitTxType>
            <v1:DE_DirectDebitData>
                  <v1:BankCode>50010060</v1:BankCode>
                  <v1:AccountNumber>32121604</v1:AccountNumber>
            </v1:DE_DirectDebitData>
            <v1:Payment>
                  <v1:ChargeTotal>19.00</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.

5.3.2 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:DE_DirectDebitTxType>
                  <v1:Type>void</v1:Type>
            </v1:DE_DirectDebitTxType>
            <v1:TransactionDetails>
                  <v1:OrderId>
                        62e3b5df-2911-4e89-8356-1e49302b1807
                  </v1:OrderId>
                  <v1:TDate>1190244932</v1:TDate>
            </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.

6 Additional Web Service actions

6.1 Initiate Clearing

Clearing for German Direct Debit 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>
```

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:

```
<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 />
<ns2:TransactionValues>
      <ns3:CreditCardTxType>
            <ns3:Type>sale</ns3:Type>
      </ns3:CreditCardTxType>
      <ns3:CreditCardData>
            <ns3:CardNumber>5426...4979</ns3:CardNumber>
            <ns3:ExpMonth>12</ns3:ExpMonth>
            <ns3:ExpYear>2008</ns3:ExpYear>
      </ns3:CreditCardData>
      <ns3:Payment>
            <ns3:ChargeTotal>1</ns3:ChargeTotal>
            <ns3:Currency>978</ns3:Currency>
      </ns3:Payment>
      <ns3:TransactionDetails>
            <ns3:OrderId>
                  b5b7fb49-3310-4212-9103-5da8bd026600
            </ns3:OrderId>
            <ns3:TDate>2008-11-10 08:52:45.0</ns3:TDate>
            <ns3:TransactionOrigin>ECI</ns3:TransactionOrigin>
      </ns3:TransactionDetails>
      <ns4:IPGApiOrderResponse>
            <ns4:OrderId>
                  b5b7fb49-3310-4212-9103-5da8bd026600
            </ns4:OrderId>
            <ns4:ApprovalCode></ns4:ApprovalCode>
            <ns4:TDate>2008-11-10 08:52:45.0</ns4:TDate>
            <ns4:TerminalID>99000002</ns4:TerminalID>
      </ns4:IPGApiOrderResponse>
      <ns2:ReceiptNumber>1865</ns2:ReceiptNumber>
      <ns2:ResponseCode>
            Y::0000036686:PPXM:0018841865
      </ns2:ResponseCode>
      <ns2:TraceNumber>001884</ns2:TraceNumber>
      <ns2:TransactionState>voided</ns2:TransactionState>
</ns2:TransactionValues>
<ns2:TransactionValues>
      <ns3:CreditCardTxType>
            <ns3:Type>credit</ns3:Type>
      </ns3:CreditCardTxType>
      <ns3:CreditCardData>
            <ns3:CardNumber>5426...4979</ns3:CardNumber>
            <ns3:ExpMonth>12</ns3:ExpMonth>
            <ns3:ExpYear>2008</ns3:ExpYear>
      </ns3:CreditCardData>
      <ns3:Payment>
            <ns3:ChargeTotal>1</ns3:ChargeTotal>
            <ns3:Currency>978</ns3:Currency>
      </ns3:Payment>
      <ns3:TransactionDetails>
            <ns3:OrderId>
                  b5b7fb49-3310-4212-9103-5da8bd026600
            </ns3:OrderId>
            <ns3:TDate>2008-11-10 08:52:49.0</ns3:TDate>
            <ns3:TransactionOrigin>ECI</ns3:TransactionOrigin>
      </ns3:TransactionDetails>
      <ns4:IPGApiOrderResponse>
```

```
                  <ns4:OrderId>
                        b5b7fb49-3310-4212-9103-5da8bd026600
                  </ns4:OrderId>
                  <ns4:ApprovalCode></ns4:ApprovalCode>
                  <ns4:TDate>2008-11-10 08:52:49.0</ns4:TDate>
                  <ns4:TerminalID>99000002</ns4:TerminalID>
            </ns4:IPGApiOrderResponse>
            <ns2:ReceiptNumber>1866</ns2:ReceiptNumber>
            <ns2:ResponseCode>
                  Y::0000036687:PPXM:0018851866
            </ns2:ResponseCode>
            <ns2:TraceNumber>001885</ns2:TraceNumber>
            <ns2:TransactionState>captured</ns2:TransactionState>
      </ns2:TransactionValues>
      <ns2:TransactionValues>
            <ns3:CreditCardTxType>
                  <ns3:Type>void</ns3:Type>
            </ns3:CreditCardTxType>
            <ns3:CreditCardData>
                  <ns3:CardNumber>5426...4979</ns3:CardNumber>
                  <ns3:ExpMonth>12</ns3:ExpMonth>
                  <ns3:ExpYear>2008</ns3:ExpYear>
            </ns3:CreditCardData>
            <ns3:Payment>
                  <ns3:ChargeTotal>1</ns3:ChargeTotal>
                  <ns3:Currency>978</ns3:Currency>
            </ns3:Payment>
            <ns3:TransactionDetails>
                  <ns3:OrderId>
                        b5b7fb49-3310-4212-9103-5da8bd026600
                  </ns3:OrderId>
                  <ns3:TDate>2008-11-10 08:52:52.0</ns3:TDate>
                  <ns3:TransactionOrigin>ECI</ns3:TransactionOrigin>
            </ns3:TransactionDetails>
            <ns4:IPGApiOrderResponse>
                  <ns4:OrderId>
                        b5b7fb49-3310-4212-9103-5da8bd026600
                  </ns4:OrderId>
                  <ns4:ApprovalCode></ns4:ApprovalCode>
                  <ns4:TDate>2008-11-10 08:52:52.0</ns4:TDate>
                  <ns4:TerminalID>99000002</ns4:TerminalID>
            </ns4:IPGApiOrderResponse>
            <ns2:ReceiptNumber>1867</ns2:ReceiptNumber>
            <ns2:ResponseCode>
                  Y::0000036686:PPX :0018861867
            </ns2:ResponseCode>
            <ns2:TraceNumber>001886</ns2:TraceNumber>
      </ns2:TransactionValues>
</ns4:IPGApiActionResponse>
```

6.3 Recurring Payments

The action *RecurringPayment* allows you to install, modify or cancel periodic credit card payments. Also, it allows you to schedule a single payment in the future.

The following example shows how to install a “periodic” payment with only one execution (*InstallmentCount*) on 31 December 2011:

```
<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>
                              20111231
                        </ns2:RecurringStartDate>
                        <ns2:InstallmentCount>1</ns2:InstallmentCount>
                        <ns2:InstallmentFrequency>
                              1
                        </ns2:InstallmentFrequency>
                        <ns2:InstallmentPeriod>
                              month
                        </ns2:InstallmentPeriod>
                  </ns2:RecurringPaymentInformation>
                  <ns2:CreditCardData>
                        <ns3:CardNumber>4035875676474977</ns3:CardNumber>
                        <ns3:ExpMonth>12</ns3:ExpMonth>
                        <ns3:ExpYear>12</ns3:ExpYear>
                        <ns3:CardCodeValue>977</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. A start date in the past is not allowed.

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. If you want to change the amount, you also need to include the currency.

To delete 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>
```

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>
```

7 Data Vault

With the Data Vault product 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, the API solution offers you the following functions:

- **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>4111111111111111</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.

- **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>
```

- **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 as well as one with account number and bank code. Please note that also in this case, existing records will be updated if the *HostedDataID* is the same.

```
<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>
                                    4035875676474977
                              </ns3:CardNumber>
                              <ns3:ExpMonth>12</ns3:ExpMonth>
                              <ns3:ExpYear>08</ns3:ExpYear>
                        </ns2:CreditCardData>
```

```
                        <ns2:HostedDataID>
                              d763bba7-1cfa-4d3d-94af-9fbe29ec0e26
                        </ns2:HostedDataID>
                  </ns2:DataStorageItem>
                  <ns2:DataStorageItem>
                        <ns2:DE_DirectDebitData>
                              <ns3:BankCode>50014560</ns3:BankCode>
                              <ns3:AccountNumber>
                                    32121503
                              </ns3:AccountNumber>
                        </ns2:DE_DirectDebitData>
                        <ns2:HostedDataID>
                              691c7cb3-a752-4d6d-abde-83cad63de258
                        </ns2:HostedDataID>
                  </ns2:DataStorageItem>
            </ns2:StoreHostedData>
      </ns2:Action>
</ns4:IPGApiActionRequest>
```

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>false</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>
```

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

8 XML-Tag overview

8.1 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.

Abbreviations:

m: mandatory
o: optional
d: optional with default value
a und **b**: maximum one of the two values
1: if **a** 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

Path/Name	**Credit Card**							**Direct Debit**	
all paths relative to `ipgapi:IPGApiOrderRequest/ v1:Transaction`	Sale	ForceTicket	PreAuth	PostAuth	Return	Credit	Void	Sale	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:CreditCard3DSecure/ v1:VerificationResponse	3	3	3			3			
v1:CreditCard3DSecure/ v1:PayerAuthenticationResponse	s	s	s			s			
v1:CreditCard3DSecure/ v1:AuthenticationValue	s	s	s			s			
v1:CreditCard3DSecure/ v1:XID	s	s	s			s			
v1:DE_DirectDebitTxType/ v1:Type								**m**	**m**
v1:DE_DirectDebitData/ v1:BankCode								**a**	
v1:DE_DirectDebitData/ v1:AccountNumber								**a**	

v1:DE_DirectDebitData/ v1:TrackData								**b**	
v1:Payment/ v1:HostedDataID	***1***	***1***	***1***			***1***		***1***	
v1:Payment/ v1:ChargeTotal	**m**	**m**	**m**	**m**	**m**	**m**		**m**	
v1:Payment/ v1:Currency	**m**	**m**	**m**	**m**	**m**	**m**		**m**	
v1:TransactionDetails/ v1:OrderId	*o*	*o*	*o*	**m**	**m**	*o*	**m**	*o*	**m**
v1:TransactionDetails/ v1:Ip	*o*		*o*			*o*		*o*	
v1:TransactionDetails/ v1:ReferenceNumber		**m**							
v1:TransactionDetails/ v1:TDate							**m**		**m**
v1:TransactionDetails/ v1:TransactionOrigin	*d*		*d*			*d*			
v1:TransactionDetails/ v1:InvoiceNumber	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:CustomerID	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Name	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Company	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Address1	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Address2	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:City	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:State	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Zip	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Country	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Phone	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Fax	*o*	*o*	*o*			*o*		*o*	
v1:Billing/ v1:Email	*o*	*o*	*o*			*o*		*o*	
v1:Shipping/ v1:Type	*o*	*o*	*o*			*o*		*o*	
v1:Shipping/ v1: Name	*o*	*o*	*o*			*o*		*o*	
v1:Shipping/ v1: Address1	*o*	*o*	*o*			*o*		*o*	
v1:Shipping/ v1: Address2	*o*	*o*	*o*			*o*		*o*	

v1:Shipping/ v1: City	*o*	*o*	*o*			*o*		*o*	
v1:Shipping/ v1: State	*o*	*o*	*o*			*o*		*o*	
v1:Shipping/ v1: Zip	*o*	*o*	*o*			*o*		*o*	
v1:Shipping/ v1: Country	*o*	*o*	*o*			*o*		*o*	

8.2 Description of the XML-Tags

8.2.1 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`.

8.2.2 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 First Data API Web Service.
`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: `%…?;…?`

8.2.3 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: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.

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.

8.2.4 DE_DirectDebitTxType

Path/Name	XML Schema type	Description
`v1:DE_DirectDebitTxType/ v1:Type`	`xs:string`	Stores the transaction type. Possible values are `sale` or `void`.

8.2.5 DE_DirectDebitData

Path/Name	XML Schema type	Description
`v1:DE_DirectDebitData/ v1:BankCode`	`xs:string`	Stores the bank code of the customer. Please make sure that the value only contains numbers and no spaces.
`v1:DE_DirectDebitData/ v1:AccountNumber`	`xs:string`	Stores the account number of the customer. Please make sure that the value only contains numbers and no spaces.
`v1:DE_DirectDebitData/ 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 `BankCode` and `AccountNumber`). The field needs to contain the concatenated track 2 and 3 data. The track data must include the track and field separators as they are stored on the card.

8.2.6 Payment

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: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)

8.2.7 TransactionDetails

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 Internet Payment Gateway will generate one automatically.
v1:TransactionDetails/ v1:Ip	xs:string	Stores the customer's IP address which can be used by the First Data API Web Service 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 First Data API Web Service needs this number for uniquely mapping a *ForceTicket* transaction to a previously performed external authorization.
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: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) and RETAIL (face to face).
v1:TransactionDetails/ v1:InvoiceNumber	xs:string	Stores the invoice number.

8.2.8 Billing

Path/Name	XML Schema	Description

	type	
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.

8.2.9 Shipping

Path/Name	XML Schema type	Description
v1:Shipping/ v1:Type	xs:string	Stores the way of delivery.
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.

9 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>
                                    4111111111111111
                              </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, First Data API automatically maps the `ipgapi:IPGApiOrderRequest` element to the corresponding Web Service operation.

10 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.

10.1 SOAP Response Message

A SOAP response message is received as the result to the credit card processor (started by the First Data Internet Payment 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>
```

10.2 SOAP Fault Message

In general, a SOAP fault message returned by the First Data API Web Service 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.

10.2.1 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`

10.2.2 SOAP-ENV:Client

10.2.2.1 MerchantException

This fault type occurs if First Data API 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.

10.2.2.2 ProcessingException

A fault of this type is raised whenever First Data API 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>
                              DECLINED
                        </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.

11 Analysing the Transaction Result

11.1 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 Internet

Payment 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 Internet Payment 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 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.
`ipgapi:TerminalID`	`xs:string`	The Terminal ID used for this transaction.

11.2 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 Internet Payment Gateway before returning the transaction failure.
`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 Internet Payment 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: • `DECLINE` • `FRAUD` • `FAILED` `DECLINE` 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 Internet Payment Gateway. If an internal gateway error should occur, the returned value is `FAILED`.
`ipgapi:TerminalID`	`xs:string`	The Terminal ID used for this transaction.

12 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 First Data API Web Service – 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.
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 Internet Payment Gateway by checking these credentials. In order to use First Data 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

12.1 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.

12.1.1 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 SSL communication will be discussed in the next chapter extending the above script.

12.1.2 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 SSL 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."\"".
...
?>
```

12.2 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 SSL channel.

13 Establishing an SSL 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 First Data API Web Service) can be sure of communicating with each other and no one else.

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

1. The client starts to establish the secure connection by sending its client certificate to the server.
2. The server receives the client certificate and verifies it against the client certificate it has stored for this client.
3. If valid, the server responds by sending its server certificate.
4. The client receives the server certificate and verifies it against the trusted server certificate.
5. If valid, both parties establish the SSL channel, as they can be sure that they are communicating with each other and no one else. All data exchanged between both parties is encrypted.

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 an SSL connection.

13.1 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 certificate 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.

13.1.1 Using the PHP cURL Extension

Building on the script started in the previous chapter, the parameters which are necessary for establishing an SSL 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\geotrust.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.

13.1.2 Using the cURL Command Line Tool

Building on the script started in the previous chapter, the statements which initialize the SSL 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\geotrust.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.

13.2 ASP

For making the above SSL 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 SSL, 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 First Data 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 server certificate PEM 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. 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 First Data API server certificate received during the SSL setup process.

After installing both certificates one could assume that the environment allowing ASP to communicate via SSL 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 SSL connection to the Internet Payment 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 SSL 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 Internet Payment 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.

14 Sending the HTTPS POST Request and Receiving the Response

The actual communication with the First Data API Web Service 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 SSL 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 First Data API Web Service to return a standard HTTP error code in the HTTP header of the response or sending an SSL failure. Their meanings can be found in any HTTP/SSL guide.

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.

14.1 PHP

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

14.1.1 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);
?>
```

14.1.2 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;
?>
```

14.2 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>
```

15 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 Internet Payment 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.

15.1 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);
            }
      }
}
```

15.2 How to construct a transaction and handle the response

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

- Credit Card
- German Direct Debit
- 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
}
```

15.3 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.

15.4 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 domain, final String password)
```

The parameters user, domain and password 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 of UsernamePasswordCredentials for Unix systems:

http://hc.apache.org/httpclient-3.x/apidocs/org/apache/commons/httpclient/UsernamePasswordCredentials.html#UsernamePasswordCredentials(java.lang.String,%20java.lang.String)

or NTCredentials for MS Windows:

http://hc.apache.org/httpclient-3.x/apidocs/org/apache/commons/httpclient/NTCredentials.html#NTCredentials(java.lang.String,%20java.lang.String,%20java.lang.String,%20java.lang.String)

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

Appendix

ipgapi.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            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"
            elementFormDefault="qualified"
            targetNamespace="http://ipg-online.com/ipgapi/schemas/ipgapi">

      <xs:import
            namespace="http://ipg-online.com/ipgapi/schemas/v1"
            schemaLocation="v1.xsd" />
      <xs:import
            namespace="http://ipg-online.com/ipgapi/schemas/a1"
            schemaLocation="a1.xsd" />
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Request - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

      <xs:element name="IPGApiOrderRequest">
            <xs:complexType>
                  <xs:choice>
                        <xs:element ref="v1:Transaction" />
                  </xs:choice>
            </xs:complexType>
      </xs:element>

      <xs:element name="IPGApiActionRequest">
            <xs:complexType>
                  <xs:choice>
                        <xs:element ref="a1:Action" />
                  </xs:choice>
            </xs:complexType>
      </xs:element>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Response  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

      <xs:element name="IPGApiOrderResponse">
            <xs:complexType>
                  <xs:all>
                        <xs:element
                              name="ApprovalCode"
                              type="xs:string" />
                        <xs:element
                              name="AVSResponse"
                              type="xs:string" />
                        <xs:element
                              name="CommercialServiceProvider"
                              type="xs:string" />
                        <xs:element
                              name="ErrorMessage"
                              type="xs:string" />
                        <xs:element
```

```
                        name="OrderId"
                        type="xs:string" />
                  <xs:element
                        name="ProcessorApprovalCode"
                        type="xs:string" />
                  <xs:element
                        name="ProcessorReceiptNumber"
                        type="xs:string" />
                  <xs:element
                        name="ProcessorReferenceNumber"
                        type="xs:string" />
                  <xs:element
                        name="ProcessorResponseCode"
                        type="xs:string" />
                  <xs:element
                        name="ProcessorResponseMessage"
                        type="xs:string" />
                  <xs:element
                        name="ProcessorTraceNumber"
                        type="xs:string" />
                  <xs:element
                        name="TDate"
                        type="xs:string" />
                  <xs:element
                        name="TerminalID"
                        type="xs:string" />
                  <xs:element
                        name="TransactionResult"
                        type="xs:string" />
                  <xs:element
                        name="TransactionTime"
                        type="xs:string" />
            </xs:all>
      </xs:complexType>
</xs:element>

<xs:element name="IPGApiActionResponse">
      <xs:complexType>
            <xs:sequence>
                  <xs:element
                        name="successfully"
                        type="xs:boolean" />
                  <xs:element
                        name="OrderId"
                        type="xs:string"
                        minOccurs="0" />
                  <xs:element
                        ref="a1:Error"
                        minOccurs="0"
                        maxOccurs="unbounded" />
                  <xs:element
                        ref="a1:TransactionValues"
                        minOccurs="0"
                        maxOccurs="unbounded" />
            </xs:sequence>
      </xs:complexType>
</xs:element>

</xs:schema>
```

v1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            targetNamespace="http://ipg-online.com/ipgapi/schemas/v1"
            elementFormDefault="qualified">

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- main  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

      <xs:element name="Transaction" type="v1:Transaction" />

      <xs:complexType name="Transaction">
            <xs:sequence>
                  <xs:choice>
                        <xs:sequence>
                              <xs:element
                                    name="CreditCardTxType"
                                    type="v1:CreditCardTxType" />
                              <xs:element
                                    name="CreditCardData"
                                    type="v1:CreditCardData"
                                    minOccurs="0" />
                              <xs:element
                                    ref="v1:CreditCard3DSecure"
                                    minOccurs="0" />
                        </xs:sequence>
                        <xs:sequence>
                              <xs:element
                                    name="CustomerCardTxType"
                                    type="v1:CustomerCardTxType" />
                              <xs:element
                                    name="CustomerCardData"
                                    type="v1:CustomerCardData"
                                    minOccurs="0" />
                        </xs:sequence>
                        <xs:sequence>
                              <xs:element
                                    name="DE_DirectDebitTxType"
                                    type="v1:DE_DirectDebitTxType" />
                              <xs:element
                                    name="DE_DirectDebitData"
                                    type="v1:DE_DirectDebitData"
                                    minOccurs="0" />
                        </xs:sequence>
                        <xs:sequence>
                              <xs:element
                                    name="UK_DebitCardTxType"
                                    type="v1:UK_DebitCardTxType" />
                              <xs:element
                                    name="UK_DebitCardData"
                                    type="v1:UK_DebitCardData"
                                    minOccurs="0" />
                        </xs:sequence>
                  </xs:choice>
                  <xs:element ref="v1:Payment" minOccurs="0" />
                  <xs:element ref="v1:TransactionDetails" minOccurs="0" />
```

```xml
				<xs:element ref="v1:Billing" minOccurs="0" />
				<xs:element ref="v1:Shipping" minOccurs="0" />
			</xs:sequence>
		</xs:complexType>

	<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
	<!-- Options - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
	<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

		<xs:complexType name="CreditCardTxType">
			<xs:complexContent>
				<xs:extension base="v1:Options">
					<xs:sequence>
						<xs:element name="Type">
							<xs:simpleType>
								<xs:restriction base="xs:string">
									<xs:enumeration
										value="credit" />
									<xs:enumeration
										value="forceTicket" />
									<xs:enumeration
										value="postAuth" /
									<xs:enumeration
										value="preAuth" />
									<xs:enumeration
										value="return" />
									<xs:enumeration
										value="sale" />
									<xs:enumeration
										value="void" />
								</xs:restriction>
							</xs:simpleType>
						</xs:element>
					</xs:sequence>
				</xs:extension>
			</xs:complexContent>
		</xs:complexType>

		<xs:complexType name="CustomerCardTxType">
			<xs:complexContent>
				<xs:extension base="v1:Options">
					<xs:sequence>
						<xs:element name="Type">
							<xs:simpleType>
								<xs:restriction base="xs:string">
									<xs:enumeration
										value="forceTicket" />
									<xs:enumeration
										value="BWListCheck" />
									<xs:enumeration
										value="sale" />
								</xs:restriction>
							</xs:simpleType>
						</xs:element>
					</xs:sequence>
				</xs:extension>
			</xs:complexContent>
		</xs:complexType>

		<xs:complexType name="DE_DirectDebitTxType">
			<xs:complexContent>
```

```xml
				<xs:extension base="v1:Options">
					<xs:sequence>
						<xs:element name="Type">
							<xs:simpleType>
								<xs:restriction base="xs:string">
									<xs:enumeration
										value="sale" />
									<xs:enumeration
										value="void" />
								</xs:restriction>
							</xs:simpleType>
						</xs:element>
					</xs:sequence>
				</xs:extension>
			</xs:complexContent>
		</xs:complexType>

		<xs:complexType name="UK_DebitCardTxType">
			<xs:complexContent>
				<xs:extension base="v1:Options">
					<xs:sequence>
						<xs:element name="Type">
							<xs:simpleType>
								<xs:restriction base="xs:string">
									<xs:enumeration
										value="credit" />
									<xs:enumeration
										value="return" />
									<xs:enumeration
										value="sale" />
									<xs:enumeration
										value="void" />
								</xs:restriction>
							</xs:simpleType>
						</xs:element>
					</xs:sequence>
				</xs:extension>
			</xs:complexContent>
		</xs:complexType>

		<xs:complexType name="Options">
			<xs:sequence>
				<xs:element
					name="StoreId"
					type="v1:String20max"
					minOccurs="0" />
			</xs:sequence>
		</xs:complexType>

	<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
	<!-- Card  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
	<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

		<xs:complexType name="CreditCardData">
			<xs:sequence>
				<xs:choice>
					<xs:sequence>
						<xs:group ref="v1:Card" minOccurs="0" />
						<xs:element
							name="CardCodeValue"
							type="v1:CardCodeValue"
```

```
                              minOccurs="0" />
                  </xs:sequence>
                  <xs:element name="TrackData" type="v1:TrackData" />
            </xs:choice>
      </xs:sequence>
</xs:complexType>

<xs:complexType name="CustomerCardData">
      <xs:sequence>
            <xs:choice>
                  <xs:group ref="v1:Card" />
                  <xs:element name="TrackData" type="v1:TrackData" />
            </xs:choice>
      </xs:sequence>
</xs:complexType>

<xs:complexType name="DE_DirectDebitData">
      <xs:sequence>
            <xs:choice>
                  <xs:sequence>
                        <xs:element name="BankCode">
                              <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                          <xs:pattern
                                                value="[0-9]{8}" />
                                    </xs:restriction>
                              </xs:simpleType>
                        </xs:element>
                        <xs:element name="AccountNumber">
                              <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                          <xs:pattern
                                                value="[0-9]{1,10}" />
                                    </xs:restriction>
                              </xs:simpleType>
                        </xs:element>
                  </xs:sequence>
                  <xs:element name="TrackData" type="v1:TrackData" />
            </xs:choice>
      </xs:sequence>
</xs:complexType>

<xs:complexType name="UK_DebitCardData">
      <xs:sequence>
            <xs:group ref="v1:Card" />
            <xs:element
                  name="CardCodeValue"
                  type="v1:CardCodeValue"
                  minOccurs="0" />
            <xs:element name="IssueNo" minOccurs="0">
                  <xs:simpleType>
                        <xs:restriction base="xs:string">
                              <xs:pattern value="[0-9]{0,4}" />
                        </xs:restriction>
                  </xs:simpleType>
            </xs:element>
      </xs:sequence>
</xs:complexType>

<xs:group name="Card">
      <xs:sequence>
```

```xml
                <xs:element name="CardNumber">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{13,24}" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="ExpMonth">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="(0[1-9])|(1[0-2])" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="ExpYear">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{2}" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
        </xs:group>

        <xs:simpleType name="CardCodeValue">
            <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{3,4}|" />
            </xs:restriction>
        </xs:simpleType>

        <xs:simpleType name="TrackData">
            <xs:restriction base="xs:string">
                <!--    track 1 length
                    + track 2 length
                    + track 3 length
                    = 90 + 64 + 107
                    = 261 -->
                <xs:pattern value="[a-zA-Z0-9;%=^?/ ]{1,261}" />
            </xs:restriction>
        </xs:simpleType>

    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- common  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

        <xs:element name="CreditCard3DSecure">
            <xs:complexType>
                <xs:all>
                    <xs:element name="VerificationResponse">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="N" />
                                <xs:enumeration value="U" />
                                <xs:enumeration value="Y" />
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:element>
                    <xs:element
                            name="PayerAuthenticationResponse"
                            minOccurs="0">
                        <xs:simpleType>
```

```
                              <xs:restriction base="xs:string">
                                    <xs:enumeration value="A" />
                                    <!-- xs:enumeration value="E" /-->
                                    <xs:enumeration value="N" />
                                    <xs:enumeration value="U" />
                                    <xs:enumeration value="Y" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element
                        name="AuthenticationValue"
                        type="v1:String20"
                        minOccurs="0" />
                  <xs:element
                        name="XID"
                        type="v1:String20"
                        minOccurs="0" />
            </xs:all>
      </xs:complexType>
</xs:element>

<xs:element name="Payment">
      <xs:complexType>
            <xs:sequence>
                  <xs:element
                        name="HostedDataID"
                        type="v1:String128max"
                        minOccurs="0" />
                  <xs:group ref="v1:Amount" minOccurs="0" />
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:group name="Amount">
      <xs:sequence>
            <xs:element name="ChargeTotal">
                  <xs:simpleType>
                        <xs:restriction base="xs:decimal">
                              <xs:pattern
            value="([1-9]([0-9]{0,12}))?[0-9](\.[0-9]{1,2})?" />
                        </xs:restriction>
                  </xs:simpleType>
            </xs:element>
            <xs:element name="Currency">
                  <xs:simpleType>
                        <xs:restriction base="xs:string">
                        <!-- http://de.wikipedia.org/wiki/ISO_4217 -->
                        <!-- http://en.wikipedia.org/wiki/ISO_4217 -->
                              <xs:pattern value="[0-9]{3}" />
                        </xs:restriction>
                  </xs:simpleType>
            </xs:element>
      </xs:sequence>
</xs:group>

<xs:element name="TransactionDetails">
      <xs:complexType>
            <xs:all>
                  <xs:element
                        name="InvoiceNumber"
                        type="v1:String48max"
```

```
                        minOccurs="0" />
                  <xs:element
                        name="OrderId"
                        type="v1:String100max"
                        minOccurs="0" />
                  <xs:element name="Ip" minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:pattern
                                          value="(25[0-5]|(2[0-4]|1[0-
9]|[1-9])?[0-9])\.(25[0-5]|(2[0-4]|1[0-9]|[1-9])?[0-9])\.(25[0-5]|(2[0-
4]|1[0-9]|[1-9])?[0-9])\.(25[0-5]|(2[0-4]|1[0-9]|[1-9])?[0-9])|" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element name="ReferenceNumber" minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:pattern
                                    value="(NEW)?[0-9a-zA-Z]{1,6}" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element name="TDate" minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:pattern value="[0-9]{10}|" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element
                              name="TransactionOrigin"
                              default="ECI"
                              minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:enumeration value="ECI" />
                                    <xs:enumeration value="MAIL" />
                                    <xs:enumeration
                                          value="TELEPHONE" />
                                    <xs:enumeration value="MOTO" />
                                    <xs:enumeration value="RETAIL" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
            </xs:all>
      </xs:complexType>
</xs:element>

<xs:element name="Billing">
      <xs:complexType>
            <xs:all>
                  <xs:element
                        name="CustomerID"
                        type="v1:String32max"
                        minOccurs="0" />
                  <xs:element
                        name="Name"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
```

```
                        name="Company"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="Address1"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="Address2"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="City"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="State"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="Zip"
                        type="v1:String24max"
                        minOccurs="0" />
                  <xs:element
                        name="Country"
                        type="v1:String32max"
                        minOccurs="0" />
                  <xs:element
                        name="Phone"
                        type="v1:String32max"
                        minOccurs="0" />
                  <xs:element
                        name="Fax"
                        type="v1:String32max"
                        minOccurs="0" />
                  <xs:element
                        name="Email"
                        type="v1:String64max"
                        minOccurs="0" />

                  <!-- DEPRECATED,
                        this field is not evaluated and will be
                        removed in v2.xsd -->
                  <xs:element
                        name="Addrnum"
                        type="v1:String96max"
                        minOccurs="0" />
            </xs:all>
      </xs:complexType>
</xs:element>

<xs:element name="Shipping">
      <xs:complexType>
            <xs:all>
                  <xs:element
                        name="Type"
                        type="v1:String20max"
                        minOccurs="0" />
                  <xs:element
                        name="Name"
                        type="v1:String96max"
```

```
                        minOccurs="0" />
                  <xs:element
                        name="Address1"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="Address2"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="City"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="State"
                        type="v1:String96max"
                        minOccurs="0" />
                  <xs:element
                        name="Zip"
                        type="v1:String24max"
                        minOccurs="0" />
                  <xs:element
                        name="Country"
                        type="v1:String32max"
                        minOccurs="0" />
            </xs:all>
      </xs:complexType>
</xs:element>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- simple types  - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

      <xs:simpleType name="String128max">
            <xs:restriction base="xs:string">
                  <xs:maxLength value="128" />
            </xs:restriction>
      </xs:simpleType>

      <xs:simpleType name="String100max">
            <xs:restriction base="xs:string">
                  <xs:maxLength value="100" />
            </xs:restriction>
      </xs:simpleType>

      <xs:simpleType name="String96max">
            <xs:restriction base="xs:string">
                  <xs:maxLength value="96" />
            </xs:restriction>
      </xs:simpleType>

      <xs:simpleType name="String64max">
            <xs:restriction base="xs:string">
                  <xs:maxLength value="64" />
            </xs:restriction>
      </xs:simpleType>

      <xs:simpleType name="String32max">
            <xs:restriction base="xs:string">
                  <xs:maxLength value="32" />
            </xs:restriction>
```

```
	</xs:simpleType>

	<xs:simpleType name="String24max">
		<xs:restriction base="xs:string">
			<xs:maxLength value="24" />
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="String20max">
		<xs:restriction base="xs:string">
			<xs:maxLength value="20" />
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="String20">
		<xs:restriction base="xs:string">
			<xs:length value="20" />
		</xs:restriction>
	</xs:simpleType>

</xs:schema>
```

a1.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:a1="http://ipg-online.com/ipgapi/schemas/a1"
            xmlns:v1="http://ipg-online.com/ipgapi/schemas/v1"
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi"
            targetNamespace="http://ipg-online.com/ipgapi/schemas/a1"
            elementFormDefault="qualified">

      <xs:import
            namespace="http://ipg-online.com/ipgapi/schemas/ipgapi"
            schemaLocation="ipgapi.xsd" />
      <xs:import
            namespace="http://ipg-online.com/ipgapi/schemas/v1"
            schemaLocation="v1.xsd" />

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Request - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

      <xs:element name="Action">
            <xs:complexType>
                  <xs:sequence>
                        <xs:choice>
                              <xs:element ref="a1:InitiateClearing" />
                              <xs:element ref="a1:InquiryOrder" />
                              <xs:element ref="a1:StoreHostedData" />
                              <xs:element ref="a1:RecurringPayment" />
                        </xs:choice>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>

      <xs:element name="InitiateClearing">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
                              name="StoreId"
                              type="v1:String20max"
                              minOccurs="0" />
                  </xs:sequence>
            </xs:complexType>
      </xs:element>

      <xs:element name="InquiryOrder">
            <xs:complexType>
                  <xs:all>
                        <xs:element
                              name="StoreId"
                              type="v1:String20max"
                              minOccurs="0" />
                        <xs:element name="OrderId" minOccurs="0">
                              <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                          <xs:maxLength value="100" />
                                    </xs:restriction>
                              </xs:simpleType>
```

```
                  </xs:element>
            </xs:all>
      </xs:complexType>
</xs:element>

<xs:element name="StoreHostedData">
      <xs:complexType>
            <xs:sequence>
                  <xs:element
                        name="StoreId"
                        type="v1:String20max"
                        minOccurs="0" />
                  <xs:element
                              name="DataStorageItem"
                              maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:choice>
                                          <xs:element
                                          name="CreditCardData"
                                          type="v1:CreditCardData" />
                                          <xs:element
                                          name="CustomerCardData"
                                          type="v1:CustomerCardData"
                                          />
                                          <xs:element
                                          name="DE_DirectDebitData"
                                          type="v1:DE_DirectDebitData"
                                          />
                                          <xs:element
                                          name="UK_DebitCardData"
                                          type="v1:UK_DebitCardData"
                                          />
                                    </xs:choice>
                                    <xs:element
                                          name="HostedDataID"
                                          type="v1:String128max" />
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="RecurringPayment">
      <xs:complexType>
            <xs:all>
                  <xs:element name="Function">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:enumeration value="cancel" />
                                    <xs:enumeration value="install" />
                                    <xs:enumeration value="modify" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>
                  <xs:element
                        name="OrderId"
                        type="v1:String100max"
                        minOccurs="0" />
                  <xs:element
```

```
                        name="StoreId"
                        type="v1:String20max"
                        minOccurs="0" />
                  <xs:element
                        ref="a1:RecurringPaymentInformation"
                        minOccurs="0" />

                  <!-- no trackdata -->
                  <xs:element
                        name="CreditCardData"
                        type="v1:CreditCardData"
                        minOccurs="0" />

                  <!-- no hostedDataID -->
                  <xs:element ref="v1:Payment" minOccurs="0" />

            </xs:all>
      </xs:complexType>
</xs:element>

<xs:element name="RecurringPaymentInformation">
      <xs:complexType>
            <xs:all>
                  <xs:element
                              name="RecurringStartDate"
                              minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <!-- YYYYMMDD -->
                                    <xs:pattern value="[0-9]{8}" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>

                  <!-- 999 = unlimited -->
                  <xs:element name="InstallmentCount" minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:integer">
                                    <xs:maxInclusive value="999" />
                                    <xs:minInclusive value="1" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>

                  <!-- for future use -->
                  <xs:element name="MaximumFailures" minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:integer">
                                    <xs:maxInclusive value="5" />
                                    <xs:minInclusive value="1" />
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:element>

                  <xs:element
                        name="InstallmentFrequency"
                        type="xs:positiveInteger"
                        minOccurs="0" />
                  <xs:element name="InstallmentPeriod" minOccurs="0">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
```

```xml
                        <xs:enumeration value="day" />
                        <xs:enumeration value="week" />
                        <xs:enumeration value="month" />
                        <xs:enumeration value="year" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:all>
    </xs:complexType>
</xs:element>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- Response  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

    <xs:element name="Error">
        <xs:complexType>
            <xs:all>
                <xs:element name="ErrorMessage" type="xs:string" />
            </xs:all>
            <xs:attribute name="Code" type="xs:string" />
        </xs:complexType>
    </xs:element>

    <xs:element name="TransactionValues">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="v1:Transaction">
                    <xs:sequence>
                        <xs:element
                        ref="ipgapi:IPGApiOrderResponse" />
                        <xs:element
                            name="ReceiptNumber"
                            type="xs:string" />
                        <xs:element
                            name="ResponseCode"
                            type="xs:string" />
                        <xs:element
                            name="TraceNumber"
                            type="xs:string" />
                        <xs:element
                            name="TransactionState"
                            type="xs:string" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

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 "&lt;/SOAP-ENV:Body&gt;".
</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, "&gt;" or "/&gt;".
</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:
      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.

Troubleshooting - Processing Exceptions

```
<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) Y null 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 PayerAuthenticationCode for 3DSecure is wrong. Allowed combinations are

VerificationResponse	PayerAuthenticationResponse	PayerAuthenticationCode
null	null	null
N	null	null
N	N	null
U	null	null
Y	A	null
Y	A	x
Y	U	null
Y	Y	null
Y	Y	x

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>DECLINED</ipgapi:TransactionResult>
            <ipgapi:TerminalID />
            <ipgapi:ProcessorResponseCode />
            <ipgapi:ProcessorApprovalCode />
            <ipgapi:ProcessorReceiptNumber />
            <ipgapi:ProcessorTraceNumber />
      </ipgapi:IPGApiOrderResponse>
</detail>
```

Explanation:
This is an example with a German Direct Debit transaction, which is not supported for the merchant. If you should receive this result for a transaction type which is included in your agreement, please contact our technical support team.

```
<detail>
      <ipgapi:IPGApiOrderResponse
            xmlns:ipgapi="http://ipg-online.com/ipgapi/schemas/ipgapi">
            <ipgapi:CommercialServiceProvider />
            <ipgapi:TransactionTime>1233656752933</ipgapi:TransactionTime>
            <ipgapi:ProcessorReferenceNumber />
            <ipgapi:ProcessorResponseMessage />
            <ipgapi:ErrorMessage>
                SGS-005005: Duplicate transaction.
            </ipgapi:ErrorMessage>
            <ipgapi:OrderId>
                IPGAPI-REQUEST-29351d8e-2634-4725-9d93-91b83704e00d
            </ipgapi:OrderId>
            <ipgapi:ApprovalCode />
            <ipgapi:AVSResponse />
            <ipgapi:TDate />
            <ipgapi:TransactionResult>FRAUD</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>DECLINED</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>1235390834046</ipgapi:TransactionTime>
            <ipgapi:ProcessorReferenceNumber />
            <ipgapi:ProcessorResponseMessage />
            <ipgapi:ErrorMessage>
                  SGS-005999: There was an unknown error in the database.
            </ipgapi:ErrorMessage>
            <ipgapi:OrderId>
                  IPGAPI-REQUEST-d26ea5c1-d0de-41d2-8c41-3d6755fc204c
            </ipgapi:OrderId>
            <ipgapi:ApprovalCode />
            <ipgapi:AVSResponse />
            <ipgapi:TDate>1235390830</ipgapi:TDate>
            <ipgapi:TransactionResult>DECLINED</ipgapi:TransactionResult>
            <ipgapi:TerminalID />
            <ipgapi:ProcessorResponseCode />
            <ipgapi:ProcessorApprovalCode />
            <ipgapi:ProcessorReceiptNumber />
            <ipgapi:ProcessorTraceNumber />
      </ipgapi:IPGApiOrderResponse>
</detail>
```

Possible Explanation:
One possible explanation is that it has been tried to void a credit card transaction as a different payment/transaction type.

```
<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.

Troubleshooting - Login error messages when using cURL

*** 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: append .cer to the file name geotrust.pem and open the certificate with a double click. You should see the issuer Equifax. Please change the name geotrust.pem.cer after the test back to geotrust.pem.

```
<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

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.