iDEAL Advanced

ING Netherlands

Integration manual PHP for iDEAL advanced

Version 3.1 - December 2013

CONTENTS

1. INTRODUCTION

This document is intended for developers who are responsible for integrating iDEAL Advanced into an online shop using the PHP development platform.

1.1 Overview

This document describes the following steps required for the integration process:

- Chapter 1, Introduction and overview
- Chapter 2, Security, gives details of the security aspects of your online shop.
- Chapter 3, Requirements, describes the prerequisites and necessary prior steps (such as generating a key pair for the owner of the online shop).
- Chapter 4, Development, describes the functions of iDEAL Advanced PHP, and how these should be integrated into the online shop.
- Chapter 5, Testing, describes the mandatory tests that have to be carried out before the online shop goes online.
- Chapter 6, Deployment, describes some aspects of the online shop going online.

Please Note For general information about iDEAL Advanced, please refer to the document 'iDEAL General information'. For integration using a different iDEAL Advanced development platform, please refer to the integration manuals for .NET or Java.

1.2 Obligations

You are urgently recommended to read this whole document together with the introductory document 'iDEAL General information' before integrating iDEAL Advanced into an online shop. We request you to pay special attention to the following **responsibilities of the acceptor**:

- **Security**: Every iDEAL acceptor is personally responsible for the secure design of his or her own online shop. The software supplied by ING, including the iDEAL Advanced PHP Connector, is built on the basis of all usual security best practices. Improper integration may, however, nevertheless lead to an insecure online shop. This integration manual contains a separate chapter (Chapter 2) on security aspects. The other chapters contain additional practical instructions relating to the security aspects of your online shop. These passages are generally indicated by the padlock symbol ( ).

- **Obligation of proactivity**: Every iDEAL acceptor must comply with the so-called 'obligation of proactivity'. This obligation means that you yourself are responsible for obtaining the status of a transaction before you make a delivery. For more information about the 'obligation of proactivity', see section 4.3.1.
- **Presentation**: Tools for the presentation of iDEAL on your website can be obtained from the merchant part of http://www.ideal.nl, like iDEAL logos and banners.
- **Testing**: After the integration of iDEAL into your online shop, you are obliged to carry out a number of tests, which are described in Chapter 5 of this document.
- **Example codes**: In a number of chapters in this document, example codes are given for the integration of iDEAL Advanced into your system. These codes are solely for the purpose of illustration. The content of the codes to be actually used by the iDEAL acceptor should be set by the iDEAL acceptor him- or herself. ING Bank N.V. will not be liable for damage of any kind whatsoever that may be suffered by the iDEAL acceptor in connection with or as a consequence of the use of the example codes. The iDEAL acceptor must indemnify ING Bank N.V. against possible claims of third parties for the compensation of damages in connection with or as a consequence of the use of the example codes by the iDEAL acceptor.

1.3 Further questions?

- **Questions**: If you have any questions or comments you can contact the iDEAL service desk. Our service desk can be contacted at ideal@ing.nl or on 020 65 225 80 between 9.00 and 17.00. The iDEAL Dashboard also has a FAQ section.
- **Illustration code**: The download for iDEAL Advanced PHP (available from the iDEAL Dashboard: https://ideal.secure-ing.com) also contains illustration code. This allows all functions of the iDEAL Advanced PHP Connector to be simulated in plain form. This illustration code is explicitly *not* intended as a basis for a complete online shop, but serves for the purpose of illustration only.

1.4 Definitions

The iDEAL system is based on bilateral relationships within the so-called '4-party' model. The 4 parties involved in the model are as follows:

- The acceptor: the owner of the online shop
- The acquirer: the acceptor's bank (ING)
- The consumer: the customer who wants to buy a product from the acceptor's online shop
- The issuer: the consumer's bank

2. SECURITY

This chapter further details the security aspects of your online shop. The remaining chapters of this integration manual contain a number of practical security instructions, indicated by the padlock symbol ( ).

This document in no way claims to give complete instructions for the security of your online shop. That is impossible in view of the scope and complexity of the subject. The aim of this chapter and of the instructions included elsewhere is, in particular, to make you aware of the subject.

There is a wealth of documentation available in books and on the Internet about the threats to which web applications are or may be exposed, and the preventive and detective measures to be taken to protect your own online shop from them. We urgently advise you to examine this documentation. The success of your online shop depends on it to a great extent. The software for iDEAL Advanced PHP supplied by ING is built on the basis of all usual security best practices. Incorrect integration of this software in your online shop, and/or the insecure setup of other parts of your online shop, may nevertheless result in an insecure online shop.

Security as part of the process

The first step in preventing this is to realize that security affects all stages of the process of your online shop's development:

- Analysis & design: Even during the design of your online shop, you should find out about the threats relevant to your online shop and formulate measures to counter them.
- Development: Some code is insecure by definition. A lot of the available security documentation contains explicit examples of insecure code, including the recommended replacement of such code. Take account of this information when building your online shop.
- Testing: Identified threats, and the associated countermeasures, naturally have to be tested. There are many different techniques and tools available for this, which vary per development platform.
- Deployment: Make sure when deploying your online shop that crucial files are not accessible to unauthorized persons. Think, for example, about the configuration of your online shop, your private key and your database. Pay particularly close attention to this if you are not hosting your online shop yourself.

OWASP

The second step towards a (more) secure online shop is to examine the documentation on the security aspects of web applications mentioned above. A good starting point here is the Open Web Application Security Project (OWASP) website, which is to be found at www.owasp.org. The OWASP site offers a wealth of insights into the vulnerabilities of software and the countermeasures to be taken.

According to OWASP, there are more than 300 security issues that (may) affect the security of web applications. OWASP periodically compiles a top 10 of the most frequently occurring vulnerabilities of web applications. The latest version is the top 10 for 2010[1].

OWASP also regularly provides new versions of a number of interesting documents. The most important of these are:

- The OWASP Guide: Describes in detail how web applications and web services can be made more secure. This is illustrated by examples in J2EE, ASP.NET and PHP.[2]
- The OWASP Testing Guide: Gives a detailed description of a test program aimed at security.[3]
- The OWASP AppSec FAQ: A list of frequently asked questions about vulnerabilities and measures that can be taken.[4]

The OWASP site also contains specific pages on various development platforms, including PHP[5]. These pages deal in detail with platform-specific security issues, including insecure code and security measures.

Please Note In the context of OWASP, think about the following points (among others):

- Check ALL input, whatever the source (acquirer, consumer).
- It is better to check on permitted values (white listing) than on forbidden values (black listing).
- Protect your database, by measures including not leaving your database connection open unnecessarily.
- Protect your private data (such as keys) by not storing them at locations that are accessible to unauthorized persons.
- Never send user names and passwords unencrypted.

[1] To be found at https://www.owasp.org/index.php/Top_10_2010.

[2] See http://www.owasp.org/index.php/Category:OWASP_Guide_Project.

[3] See http://www.owasp.org/index.php/Category:OWASP_Testing_Project.

[4] See http://www.owasp.org/index.php/OWASP_AppSec_FAQ.

[5] See http://www.owasp.org/index.php/Category: PHP.

3. REQUIREMENTS

This chapter describes the software, settings and configuration that are required for the successful integration of iDEAL Advanced into the acceptor's system using PHP as the development platform.

3.1 Required software

To integrate iDEAL Advanced PHP successfully into an online shop according to the mode of operation described in this document requires the following software:

For development and configuration purposes:

- A development environment for PHP, for example Netbeans. Other options are Jetbrains PHPStorm, Eclipse for PHP, Zend Studio or Adobe Dreamweaver. The choice depends on the user's/developer's preferences.
- PHP from version 5.3 upwards with cUrl and openssl module installed and enabled.
- To generate your own certificate/key pair:
    - OpenSSL implementation for Windows, to be found at http://www.slproweb.com/products/Win32OpenSSL.html. For OpenSSL in general, see also www.openssl.org. Section 3.5 of this document describes the steps required to make your own certificate/key pair.
    - OpenSSL implementation for Linux, to be found at http://www.devside.net/guides/linux/openssl with information about the installation of OpenSSL under Linux. Section 3.5 of this document describes the steps required to make your own certificate/key pair.
- The iDEAL Advanced PHP Connector, available from the iDEAL Dashboard as part of the iDEAL Advanced Integration PHP package, to be found under the menu option Documentatie (Documentation).

In a Microsoft Windows test and production environment:

- Apache with TLS, for further information about Apache and PHP under Windows see for example http://www.html-site.nl/apache_php_mysql.php
- or IIS with PHP support, see for example http://nl3.php.net/install.windows
- PHP from version 5[6] upwards with cUrl and openssl module installed and enabled.

[6] There are known issues with PHP Version 5.2.x concerning openssl and the functions openssl_sign and openssl_verify

- The acceptor's own PKCS#12 certificate (including his or her own private/public key pair). Section 3.5 of this document describes the necessary steps for generating this.
- The iDEAL Acquiring platform certificate (`ideal_v3.cer`). Available from the iDEAL Dashboard. See section 3.4 for details.
- The iDEAL Advanced PHP Connector. Available through the iDEAL Dashboard, as part of the iDEAL Advanced Integration PHP package, to be found under the menu option Documentatie (Documentation).

In a Linux test and production environment:

- Apache with TLS, for further information about Apache and PHP under Linux see for example http://dan.drydog.com/apache2php.html
- PHP from version 5.3 upwards with cUrl and openssl module installed and enabled.
- The acceptor's own PKCS#12 certificate (including his or her own private/public key pair). Section 3.5 of this document describes the necessary steps for generating this.
- The iDEAL Acquiring platform certificate (`ideal_v3.cer`). Available from the iDEAL Dashboard. See section 3.4 for details.
- The iDEAL Advanced PHP Connector. Available from the iDEAL Dashboard as part of the iDEAL Advanced Integration PHP package, to be found under the menu option Documentatie (Documentation).

3.2 Required settings

- Internet traffic via port 443 (TLS): It must be possible to make a TLS connection to the acquirer's iDEAL server from the web server (where the online shop is located). These connections run via port 443.
- Access to a directory with specific read and write rights during the installation of the certificates on the web server.
- Security measures: PHP must be configured correctly as follows:
    - **REGISTER_GLOBALS**: You are urgently recommended to switch this option OFF. If this option is on, users can influence the application. If REGISTER_GLOBALS is on, variables can be entered directly from the browser's address bar. This makes it easier for malicious persons to misuse the system. See also http://www.php.net/manual/en/security.globals.php.
    - **OPEN_BASEDIR**: With this option, you can indicate the folders to which PHP has access. Use this setting to give minimal protection to folders with your configuration and your keys. See also http://www.php.net/manual/en/features.safe-mode.php#ini.open-basedir
    - **SAFE_MODE**: If this option is on, a check is made as to whether the permissions for the file system correspond to the user who is trying to change/open the file system. If SAFE_MODE is on, this therefore implies that PHP can only

process/open files for which it has actually received permission. You are strongly advised to set this option to ON, even if it means that you have to implement certain functions of your online shop differently. See also http://www.php.net/manual/en/features.safe-mode.php

- **DISPLAY_ERRORS:** This option determines whether error messages should be displayed. If this option is set to OFF, only a white page is displayed when an error occurs. You are strongly advised to set this option to OFF, even if it means that you have to implement certain functions of your online shop differently to ensure that no valuable information is given away. See also http://www.php.net/errorfunc
- **ERROR_REPORTING:** This option determines whether and to what extent warnings are displayed when an error occurs. You are strongly advised to set this option to OFF, even if it means that you have to implement certain functions of your online shop differently to ensure that no valuable information is given away. See also http://www.php.net/errorfunc
- **EXPOSE_PHP**: This option determines whether PHP can indicate that it is installed on a web server. When it is on, for example, the PHP signature is added to the web server's http header. You are urgently advised to set this option to OFF in the production environment. See also http://www.php.net/ini.core

3.3 Using the default configuration file

The following parameters have to be configured in the `config.conf` file:

- `MerchantID`: the ID of the online shop, received by the acceptor during the registration process.
- `SubID`: subID of the online shop, default value = 0 (zero); only to be changed in consultation with the acquirer.
- `MerchantReturnURL`: URL of the page in the online shop to which the consumer is redirected after an iDEAL transaction. This value can be overruled as necessary in the online shop implementation (see section 4.2.2).
- `AcquirerURL`: URL of the acceptor's acquirer; the following prescribed values apply to ING (the URLs are case-sensitive):
  - Test environment: https://idealtest.secure-ing.com/ideal/iDEALv3
  - Production environment: https://ideal.secure-ing.com/ideal/iDEALv3
- `AcquirerTimeout`: number of seconds (default = 10) of waiting time for a response from the iDEAL services. If no response is received during that time, an exception is displayed.
- `Privatecert`: filename of the acceptors private certificate. See section 3.5 for more information about the acceptor's certificate.
- `Certificate0`: the certificate provided by the acquirer (ideal_v3.cer). See section 3.4 for more information.

- `Privatekey`: filename of the acceptors private key (priv.pem). See section 3.5 for more information.
- `PrivatekeyPass`: Password used during the generation of the private key (*`ownPassword`*). See section 3.5 for more information

The settings in the `config.conf` file may, for example, appear as follows:

```
Privatekey=<path-to>/priv.pem
PrivatekeyPass=ownPassword
Privatecert=<path-to>/cert.cer
Certificate0=<path-to>/ideal_v3.cer
AcquirerURL= https://ideal.secure-ing.com/ideal/iDEALv3
AcquirerTimeout=10
MerchantID=005012345
SubID=0
MerchantReturnURL=http://[yourwebpage]/StatReq.php
ExpirationPeriod=PT10M
LogFile=<path-to>/Connector.log
TraceLevel=DEBUG
```

To use the default configuration file you must specify the path of the `config.conf` when instancing the connector. This can be done in two ways:

```
iDEALConnector::getDefaultInstance("<path-to>/config.conf");
```

```
new iDEALConnector(
    new DefaultConfiguration("<path-to>/config.conf"),
    new DefaultLog("<path-to>/connector.log")
);
```

Using the constructor you can use your own custom implementations of `IConnectorConfiguration` and `IConnectorLog`.

Please Note In order to validate your system settings and configuration please open the `configurationCheck.php` page (included in the Connector package).

Please Note Never put the configuration file, encryption keys and other sensitive information in the web server's document root (www or www-root). At that location, undesirable visitors can access your data very easily.

Please Note NEVER activate tracing in the production environment. This can cause both performance and security problems!

3.4 Acquirer's certificate

iDEAL Advanced is supplied with a certificate from the iDEAL Acquiring system: ideal_v3`.cer`. This certificate contains the public key of the iDEAL Acquiring platform, with which it is possible to check that all messages are genuinely being sent by the ING iDEAL Acquiring platform.

The following steps have to be taken to install the certificate from the iDEAL Acquiring platform on the acceptor's system:

1. Download the acquirer certificate (ideal_v3.cer) from the iDEAL dashboard (https://ideal.secure-ing.com/ideal/)
2. Copy the certificate file to a defined folder outside the web root.
3. Give only the web server's user account rights to this certificate; this account only requires read rights.
4. Configure the following parameter in the config.conf file:
   - Certificate0=`<path-to>/`ideal_v3.cer.

Please Note Do NOT place your certificate in the web root. The web root is easy for unauthorized persons to access.

Please Note This certificate serves only as a medium for the public key. It therefore does not matter if the status according to the certificate details is `Ongeldig (Invalid)`.

3.5 Acceptor's certificate and key pair

The following steps have to be carried out in order to generate and activate the acceptor's own certificate and key pair:

1. Generate an RSA private key with the following command (use a self-chosen password for the field *`ownPassword`*):
   `openssl genrsa –aes128 –out priv.pem –passout pass:`*`ownPassword`* `2048`
2. The result of this is the `priv.pem` file. PEM files are intended for the storage of the private key and the public key. The file is saved as the default in the bin directory under the OpenSSL directory.

Please Note Take into account the fact that the password given here is also needed to install the certificate in the ultimate hosting environment. This may not be the acceptor's own environment, but that of an external hosting provider. Therefore, do **not** use a password that the acceptor uses for other business or private purposes.

Please Note Always use a *strong* password, because with this certificate a malicious person can pretend to be you vis-à-vis the acquiring platform.

3. Generate a certificate on the basis of the RSA private key (use the same password for the field *ownPassword* as in step 2):
   `openssl req -x509 -sha256 -new -key priv.pem -passin pass:ownPassword -days 1825 -out cert.cer`

   This command leads to a series of questions about the requestor. Enter the correct values. This relates among other things to the organization name. You need this organization name in some subsequent steps.

   The result is the `cert.cer` file. CER files contain the certificate and the public key. The file is saved as the default in the bin directory under the OpenSSL directory.

4. Copy the private key and the certificate to the same defined folder as the configuration file (see section 3.3).

5. Configure the following parameters in the `config.conf` file; replace the value for `PrivatekeyPass` in *ownPassword* that you also used in step 2:

   ```
   Privatekey=<path-to>/priv.pem
   PrivatekeyPass=ownPassword
   Privatecert=<path-to>/cert.cer
   ```

6. Upload `cert.cer` to the test environment of the iDEAL Acquiring platform (via the iDEAL Dashboard).

7. After positive verification by the iDEAL Service Desk (see Chapter 'Registration process' of the 'iDEAL General information' document): Upload `cert.cer` to the production environment of the iDEAL Acquiring platform (via the iDEAL Dashboard).

Please Note The priv.pem, cert.cer and config.conf files created contain secret information (the private key and configuration data). Make sure that these files are not accessible to others, and that they are therefore not in the web root of the web server (the www-root or www folder).

Recommendation: In order for you to distinguish clearly between the iDEAL production and test environments, it is recommended to create a separate *.cer and *.p12 file for both environments.

4. DEVELOPMENT

This chapter describes in detail how the iDEAL protocols can be integrated into an online shop. The following protocols are used within iDEAL:

- **Directory protocol**: Provides the list of issuers who are members of iDEAL. The consumer (online shop customer) chooses his or her own bank from these. The iDEAL Advanced PHP Connector supports this protocol through the function `getIssuers`.
- **Payment protocol**: Launches and executes a transaction in which money is transferred by the issuer selected by the consumer to the acceptor's acquirer. The iDEAL Advanced PHP Connector supports this protocol through the function `startTransaction`.
- **Query protocol**: Requests the status of a transaction. On this topic, see for example the section on the obligation of proactivity (4.3.1). The iDEAL Advanced PHP Connector supports this protocol through the function `getTransactionStatus`.
- **Error protocol**: Guidelines for processing error situations (on this topic, see also section 2.5 of the iDEAL Reference Guide). The iDEAL Advanced PHP Connector supports this protocol implicitly: every error situation is dealt with through an exception; in the case of iDEAL errors this is an `iDEALException`. Every `iDEALException` contains, among other things, all iDEAL `ErrorRes` information, including the `consumerMessage`.

Please Note If there is a **proxy** server active between the online shop and the acquirer, special measures have to be taken. See section 4.5 on this topic.

4.1 Directory request (getIssuers)

The directory request, implemented in the iDEAL Advanced Connector through `getIssuers`, ensures that the most recent list of member issuers (consumer banks) is provided. The pick list of banks in the online shop should be populated on the basis of this list. The consumer then selects his or her bank from it.

Input

Invoking `getIssuers` does *not* require *any* parameters.

Result

Invoking `getIssuers` can produce two results:

- If no exception is thrown, a `DirectoryResponse` object consisting of the following elements is returned:
  - `DirectoryDate`: the date of the directory listing
  - `AcquirerId`: the Acquirer Id value
  - `Countries`: the list of country objects, each consisting of

- `CountryNames`: the name(s) of the country
- `Issuers`: each country has a list of issuers objects consisting of
  - `Id`: the issuerId
  - `Name`: the issuerName

Section 4.3.1 of the Reference Guide describes in detail how the issuers should be displayed to the consumer.

- If an exception is thrown it should be handled. For further details see section 4.4.

The example code for invoking `getIssuers` may, for example, be as follows:

```
try {
    $iDEALConnector =
    iDEALConnector::getDefaultInstance("<path-to>/config.conf");

    $response = $iDEALConnector->getIssuers();

    foreach ($response->getCountries() as $country) {
          //process response
     }
}
catch (SerializationException $ex)
{
      //handle serialization exception
  }
catch (SecurityException $ex)
{
    //handle security exception
}
catch(ValidationException $ex)
{
    //handle validation exception
}
catch (iDEALException $ex)
{
    $errorCode = $ex->getErrorCode();
    $consumerMessage = $ex->getConsumerMessage();
    $errorMsg = $ex->getMessage();
```

```
        //use error data
    }
    catch(Exception $ex)
    {
      //handle other exceptions (eg: InvalidArgument)
    }
```

Periodic invocation

In practice, the list of issuers rarely changes. It is therefore not necessary to re-invoke the `getIssuers` function for every transaction. Instead, the result can be periodically invoked, and cached or saved between times. It is advised always to check the list for validity **daily** and to refresh it if necessary. The `DirectoryDate` property can be used to check whether the list has been updated.

The example code for this may, for example, look as follows if the issuer list has been logged in between times in the online shop database:

```
DateTime dirDateTime; // datetime stamp of getIssuers
DateTime dbDateTime; // datetime stamp in own database
// Add: invoke getIssuers(), see previous example
$dirDateTime =$response-> getDirectoryDate();
$dbDateTime = // invoke own function to request dbDateTime
if ( $dirDateTime > $dbDateTime )
{
     // Add: function to save new list
     // in own database
}
```

If the issuer list changes at the acquirer, it is advisable to update the list in the online shop accordingly as soon as possible. It is therefore advised to build a function into the online shop with which the acceptor can do this any time he or she wants.

Considering the number of directory requests:

- Maximum of once per 24 hour period;
- Do not perform directory requests prior to each transaction.

Please Note Section 2.2.1 of the Reference Guide provides an overview of possible errors that may occur if the issuer list in the online shop no longer corresponds to the actual list.

4.2 Transaction request (startTransaction)

You initiate an iDEAL transaction with the transaction request, implemented through `startTransaction`. All data required for this are firstly derived from the parameters you have provided and secondly obtained by the connector from the configuration data.

Input

Invoking `startTransaction` requires a minimum of 2 parameters:

- `IssuerId`: the ID of the issuer the consumer has selected from the pick list.
- `Transaction`: the transaction object described as follows:
  - `Amount`: the amount payable in EURO with period used as decimal separator (1 Euro = 1.00).
  - `Description`: the description of the product.
  - `EntranceCode`: a code determined by the online shop with which the purchase can be authenticated upon redirection to the online shop (see section 4.2.2 for details).
  - `ExpirationPeriod`: the expiration period in minutes.
  - `PurchaseId`: the purchase number according to the online shop's system.
  - (optional) `Currency`: the consumer's currency.
  - (optional) `Language`: the consumer's preferred language.
- (optional) `MerchantReturnURL`: if different from the configured value.

Please Note In the iDEAL test environment, the result of transactions is determined by the value of the tendered `amount`. Use, for example, `amount=1.00` to simulate a successful transaction. See section 5.1 for details.

Please Note The appendix "Character Set for Interbank Exchanges" in the Reference Guide contains a table of permitted characters. Other characters, such as diacritical marks, are not permitted. If a character is used in properties of the `Transaction` object (such as `Description`, `EntranceCode` and `PurchaseId`) that is not included in the agreed character set, the characters are converted to an equivalent that does occur in the character set.

Result

Invoking `startTransaction` can essentially produce two results:

- If no exception is thrown, an `AcquirerTransactionResponse` object consisting of the following elements is returned:
  - `AcquirerId`: the ID of the acquirer.

- `TransactionId`: unique identification of the transaction as issued by the acquirer. It is recommended to link this number to your own purchase number (`PurchaseId`) to support your own accounting system.
- `PurchaseId`: the purchase number.
- `TransactionTimestamp`: the transaction timestamp provided by the acquirer.
- `IssuerAuthenticationURL`: the full URL of the issuer (the consumer's bank). The online shop should redirect the consumer automatically to this URL.

- If an exception is thrown it should be handled. For further details see section 4.4.

4.2.1 Redirect to the issuer

Once the transaction request has been successfully initiated, the consumer is passed on via a redirect to his or her own Internet banking environment. The URL at which the issuer expects the consumer is sent in the answer to the transaction request. With the help of the following example code, the consumer is redirected to the issuer:

Please Note The Reference Guide states that the acceptor has to validate the URL before the redirect; one of the checks made in validation is to ensure that the URL does not contain any scripting. The iDEAL Advanced Connector performs this URL check automatically.

```
$connector =
           iDEALConnector::getDefaultInstance("<path-to>/config/conf");
// add parameters
$response = $connector->startTransaction( . . . );


// add error check
$url = $response->getIssuerAuthenticationURL();
$header("Location: $url);
```

Please Note The iDEAL Reference Guide states that the redirect to the issuer has to take place within the browser window in which the consumer has clicked on the Pay button. The acceptor's entire page has to be replaced by the selected issuer's entire page. It is therefore not allowed to use a second browser window (pop-ups) or frames.

4.2.2 Redirect to the online shop

After the Internet banking payment (successful or otherwise), the consumer is automatically redirected to the acceptor's online shop, via the URL that is configured as a value of `merchantReturnURL`[7].

Validation

The return URL automatically contains the `entranceCode` and the `transactionId` of the transaction. The `entranceCode` is initially provided by the acceptor as a parameter of the Payment protocol, and can then (in combination with the transaction ID) be used to 'authenticate' the consumer as the person for whom the transaction was launched.

> **Please Note** It is advised always to perform this validation. Take account here of the required minimum variation of the `entranceCode` as stated in 3.3.1 of the Reference Guide.

The online shop will then request the status of the transaction. To do this, use the `getTransactionStatus` function (see next section).

4.3 Status request (getTransactionStatus)

After carrying out a payment order, the online shop itself should always request the status of the transaction through iDEAL's so-called Query protocol. This protocol is implemented in the iDEAL Advanced connector through the function `getTransactionStatus`.

Input

To invoke `getTransactionStatus`, you only require the `TransactionId` parameter of the transaction that is to be checked.

Result

Invoking `getTransactionStatus` can essentially produce two results:

- If no exception is thrown, an `AcquirerStatusResponse` object consisting of the following elements is returned:
  - `AcquirerId`: the ID of the acquirer.
  - `TransactionId`: the ID of the transaction.

[7] If a different URL is desired in special cases, the URL given in the configuration can be overruled by (temporarily) updating the `merchantReturnURL` property of the Connector.

- `Status`: the status of the transaction.
- `StatusTimestamp`: the timestamp of current status.
- If the transaction has been successful (status=`Success`), the consumer's details will also be provided, i.e. his or her `ConsumerName`, `ConsumerIBAN`, `ConsumerBIC`, `Amount` and `Currency`.

- If an exception is thrown it should be handled. For further details see section 4.4.

The example code for invoking `getTransactionStatus` may, for example, be as follows:

```
try {
$iDEALConnector =
iDEALConnector::getDefaultInstance("<path-to>/config.conf");

$response = $iDEALConnector->getTransactionStatus($trxId);

if ($response->getStatus() === IDEAL_TX_STATUS_SUCCESS){
      $consumerName = $response->getConsumerName();
      $consumerIBAN = $response->getConsumerIBAN();
      $consumerBIC = $response->getConsumerBIC();
      $amount = $response->getAmount();
      $currency = $response->getCurrency();
}
}
catch (SerializationException $ex)
{
  //handle serialization exception
 }
catch (SecurityException $ex)
{
    //handle security exception
}
catch(ValidationException $ex)
{
    //handle validation exception
}
catch (iDEALException $ex)
{
```

```
        $errorCode = $ex->getErrorCode();
        $consumerMessage = $ex->getConsumerMessage();
        $errorMsg = $ex->getMessage();

        //use error data
    }
    catch (Exception $ex)
    {
        //catch other exceptions (eg: InvalidArgument)
    }
```

Transaction successful

Section 2.3.1 of the Reference Guide describes the possible statuses that can be returned by `getTransactionStatus`. *Only* the status `Success` means that the transaction has been successful, and that delivery should be made.

Please Note If the acceptor uses iDEAL reconciliation, then the iDEAL Dashboard will display not only status 003 (Success), but also the subsequent statuses 007 (reconciled) and 009 (paid). These statuses also indicate a successful transaction. You should always receive a confirmation of the status `Success` via `getTransactionStatus`; you cannot distinguish between 003, 007 and 009 from the programming. For more information on reconciliation see the iDEAL Dashboard.

Status unknown

In implementation, take account of the fact that a consumer may decide to close the browser during or immediately after making the payment.

In such cases, he/she is *not* automatically redirected to the online shop, so the status of the transaction is *not* requested automatically. If the online shop is not set up for this situation, a consumer may have made a payment but does not receive a product because according to the information of the acceptor the transaction status is still 'open'.

This is not only undesirable, but also in conflict with the acceptor's 'obligation of proactivity'. This 'obligation of proactivity' and possible solutions to the outlined problems are considered in more detail in the next section.

Considering the number of status requests per transaction:

- Maximum of five times per transaction;
- Maximum of two times during the `expirationPeriod`;
- After the `expirationPeriod` not more often than once per 60 minutes;

- No status request after a final status has been received for a transaction;
- No status request for transactions older than 7 days.

Examples of possible times when a status request can be executed:

- 30 seconds after a transaction request is sent;
- Half-way through an `expirationPeriod`;
- Just after an `expirationPeriod`;
- A certain period after the end of the `expirationPeriod`.

Usually one of the final statuses should be returned shortly after the expiration period. If the "Open" status is still returned after the expiration period, this can indicate a system failure. If this failure is not solved within 24 hours, please refer to the IDEAL Dashboard and if desired contact the iDEAL service desk. Please stop sending StatusRequests.

4.3.1 Obligation of proactivity

The acceptor has a so-called obligation of proactivity with regard to the status of a transaction. This means that an acceptor is personally responsible for obtaining a final status for a transaction, *even* if the consumer is *not* redirected to the online shop after paying for the transaction or otherwise. This latter situation may occur when the customer closes the browser window early.

If the status of a transaction is unknown, or is equal to `Open`, you have the following options for getting the status:

- Perform a status request 'manually' for a certain `transactionID`. This requires implementing a function in the online shop which allows the acceptor to launch a status request for an individual transaction for a given `transactionId`.

> **Please Note** Functions that are solely intended for the online shop owner and not for consumers must be implemented in such a way that unauthorized persons cannot launch them either directly or indirectly. An adequate authorization mechanism must be implemented for this purpose.

- Perform automated periodic status requests for all transactions that have not yet been completed. Here, the same guidelines apply as for requesting a single transaction status 'manually'.
- Perform a status request automatically after the end of the `expirationPeriod`. In this case, see also the last page of section 3.3.1 of the Reference Guide.
- Log into the iDEAL Dashboard, search for the payment and request the status using the 'status request' button.

Example

An online shop that sells flight tickets reserves a ticket for every potential purchase for a period of 15 minutes. A customer has to purchase a flight ticket within this period. If he or she does not, the ticket is released again for sale to other consumers. If the consumer purchases a flight ticket through iDEAL within 5 minutes but closes the browser window at his or her Internet bank after payment, he or she is not redirected to the online shop. The online shop then does not receive the signal that the payment has been completed, and will release the flight ticket after 15 minutes, although the consumer has actually paid!

Measures to prevent this should be taken in the online shop, for example by implementing a query function linked to a timer. The online shop should then (in this specific example) be able to make use of an `expirationPeriod` of, for example, 10 minutes (which means that a consumer can spend a maximum of 10 minutes on an iDEAL transaction; after that the transaction automatically receives the status `expired`). If a consumer has not been redirected to the online shop after the defined expiration period of 10 minutes, the self-constructed query function will then automatically perform a status request, so that it has a final status well before the end of the reservation period of 15 minutes.

4.4 Exception handling

If an error occurs an exception is thrown. There are 4 types of exceptions Validation, Serialization, Security and iDEAL all must be handled by surrounding your API calls with a try-catch in the following manner: in the message traffic between the acceptor, acquirer and/or issuer, the iDEAL Advanced Connector sends back a different message. To see whether an error has occurred, you should check this after getting a response by invoking the following function:

```
try
{
   $iDEALConnector =
         iDEALConnector::getDefaultInstance("<path-to>/config.conf");
    $response = $iDEALConnector->getIssuers(); //can be any API call


   //use response
}
catch (ValidationException $ex)
{
   //handle validation exception
}
catch (SerializationException $ex)
{
   //handle serialization exception
}
catch (SecurityException $ex)
```

```
{
   //handle security exception
}
catch (iDEALException $ex)
{
   $errorCode = $ex ->getErrorCode();
   $errorMsg = $ex ->getMessage();
   $consumerMessage = $ex ->getConsumerMessage();
   // add:  Show $consumerMessage on screen
}
catch (Exception $ex)
{
   // handle other exceptions (eg: InvalidArgument)
}
```

In these variables, the error code, the error message from iDEAL and the error message to the consumer will then be displayed in succession.

Section 2.5.2 of the Reference Guide binds the acceptor to display only the `consumerMessage` from this object to the consumer. To support this, the iDEAL Advanced Connector ensures that the `consumerMessage` field is always populated (either with the value given by the acquirer or with the mandatory texts from table 16 of the Reference Guide).

4.5 Proxy servers

If there is a proxy server active between the online shop and the acquirer's site, it may be necessary to log in to the proxy server as part of the communication. For this purpose, a function has to be implemented in the online shop that entails a user name and password.

> **Please Note** The user name and password for the proxy server have to be saved in a secure manner. If these data are saved in a configuration file, the file must be saved in a secured folder that can only be accessed by PHP.

5. TESTING

This chapter describes all the mandatory tests that have to be carried out before an online shop with iDEAL Advanced integration can go online. In addition, the remaining functions of your online shop should of course also be tested. However, that goes beyond the scope of this document.

5.1 Mandatory tests

The mandatory tests are described functionally in the Chapter '*Registration process*' of the general documentation for iDEAL Advanced. There are seven tests, which should all be performed in the iDEAL test environment (https://idealtest.secure-ing.com).

The following checks should be made prior to testing:

- In the config.conf file, privateCert should be defined as the filename (priv.pem) of the Acceptor's private key.
- The Acceptor's public certificate (cert.cer) must be uploaded to the iDEAL test environment.

After that, the tests are run as follows:

1. The acceptor logs into the iDEAL test environment using the user name and password obtained in the registration process.
2. The acceptor sends 7 test orders to the URL of the iDEAL test environment (https://idealtest.secure-ing.com/ideal/iDEALv3). The test environment returns the following pre-programmed results:

Order	**Expected result if integration is correct**
Transaction with `amount` = 1.00:	`Success`
Transaction with `amount` = 2.00:	`Cancelled`
Transaction with `amount` = 3.00:	`Expired`
Transaction with `amount` = 4.00:	`Open`
Transaction with `amount` = 5.00:	`Failure`
Transaction with `amount` = 7.00:	`SO1000 Failure in system`
Directory request (`getIssuers`)	`Issuer Simulator`

3. The acceptor checks the results obtained.

Please Note Test results are sent automatically to iDEAL for verification several times a day. Positive verification of the test results is needed to activate iDEAL in the production environment. Activation must be carried out by the acceptor, via the iDEAL Dashboard. This can be done from the next day onwards.

6. DEPLOYMENT

This chapter describes some aspects of deploying a website with iDEAL Advanced integration. A full description of all possible deployment scenarios goes beyond the scope of this document.

6.1 Hosting under own management

Check in advance that your own environment meets all the requirements for iDEAL Advanced (as described in Chapter 3).

After deployment, check explicitly that all parts of your site are sufficiently protected from unauthorized persons. For example, think about the configuration, your database, and any pages that are solely intended for you.

6.2 Hosting at an external provider

If hosting is done by an external provider, explicit account must be taken of the privacy aspects of your online shop. In particular, consider your private key (see section 3.5).

Also, check in advance that the hosting provider meets all the requirements for iDEAL Advanced (as described in Chapter 3).

After deployment, check explicitly that all parts of your site are sufficiently protected from unauthorized persons. For example, think about the configuration, your database, and any pages that are solely intended for you.

APPENDIX A: DATA CATALOGUE

Parameter	Format	Description
issuerID	PN..4	ID of the consumer's bank
merchantID	PN..9	Your merchantID is provided in the registration process.
subID	N..max6	Default value = 0 (zero)
merchantReturnURL	AN..max512	URL (not necessarily beginning with http:// or https://) to which the Consumer must be redirected after authorization of the transaction at the Issuer The resource indicated by the URL must be the website of the Merchant or a part thereof.
purchaseID	AN..max35	The online shop's unique order number (determined by the acceptor)
amount	N..max12	The amount payable in euro (WITH period used as a decimal separator). **N.B.** Note that in the test environment the result of a transaction is influenced by the transaction amount. See section 5.1 for details.
currency	AN3	Fixed value = EUR (at present only the euro is supported)
expirationPeriod	RDT	Period within which the iDEAL transaction can take place. Notation PT1H for 1 hour, PT10M for 10 minutes. Maximum value: 1 hour. Minimum value: 1 minute. Suggested value: 10 minutes.
language	CL..2	Code list as per ISO 639-1. (e.g. „en" for English, „nl" for Dutch) should be used. If a non-supported or non-existing language is entered, the standard language of the Issuer is used.
description	AN..max32	Description of the order (determined by the acceptor)
entranceCode	ANS..max40	Code is determined by acceptor. This code makes it possible to relate the consumer to a specific transaction when the consumer is redirected to the online shop after payment. The entranceCode is sent by means of HTTP(S) GET to the merchantReturnURL.
acquirerURL	n/a	URL to which the iDEAL requests have to be sent. Test environment: https://idealtest.secure-ing.com/ideal/iDEALv3 Production environment: https://ideal.secure-ing.com/ideal/iDEALv3 **N.B.** the URLs are case-sensitive.
consumerName	AN … max 70	Name of the Consumer according to the name of the account used for payment.

Parameter	Format	Description
status	AN … max 9	Indicates the status of the transaction **Success**: Positive result; the payment is guaranteed. **Cancelled**: Negative result due to cancellation by Consumer; no payment has been made. **Expired**: Negative result due to expiration of the transaction; no payment has been made. **Failure**: Negative result due to other reasons; no payment has been made. **Open**: Final result not yet known.
consumerBIC	ANS…max 11 (ISO9362)	The BIC of the bank where the Consumer's account is held.
consumerIBAN	IBAN	The IBAN of the account the Consumer used for payment.
createDateTimestamp	datetime	The timestamp of the transaction.
statusDateTimestamp	datetime	The timestamp of the transaction status.

Format	Description
AN	Alphanumerical, free text
ANS	Strictly alphanumerical (letters and numbers only)
N	Numerical
PN	Numerical (padded), content is filled up to maximum length with preceding zeros
..23	Maximum number of characters for alphanumerical and numerical values
CL	Code list
RDT	Relative date time field: PnYnMnDTnHnMnS
DEC	Decimal number only

Parameters related to the certificate:

Parameter	Description
privateCert	filename of the acceptors private certificate. See section 3.5 for details.
CERTIFICATE0	The Certificate provided by the acquirer (ideal_v3.cer). See section 3.4 for details.