Version 0.9.8.2

Guide Swish API

Integration Guide

Datum: 2015/12/10

Table of content

Date	Version	Name	Description
2015-11-13	0.9.8		Version 0.9.8
2015-12-08	0.9.8.1	Claes Scheffer	Changed document name to Guide Swish API. 1.3 Changed some wording. 2.1 Added personal information that needs to be registered about the CPOC. 3 Added information about port. 4 Added guidelines for handling keys and certificate. 8.2 Added refund. 11.1.3 Changed url in code examples. 11.1.4 Changed url in code examples. 11.1.5 Added information about time outs. 11.2.1 Deleted error codes AC05, AC06, AC07, AC15, AM04, AM14, AM21, and DS0K. Added error code RF07. Changed url in code examples. 11.3.1 Clarified mandatory, optional and response parameters. Deleted error codes AC05, AC06, AC07, AC15, AM04, AM14, AM21, and DS0K. Added error code RF07. 11.3.2 Clarified mandatory, optional and response parameters. Deleted error codes AC05, AC06, AC07, AC15, and AM04. Added error code RF07.
2015-12-10	0.9.8.2	Claes Scheffer	7 Corrected Swish API URL for paymentrequests

1. Introduction

1.2 Terms and definitions

Term	Definition
Partner	A partner is a company, working with technical integrations, app development, platform development and/or payment services that may help and facilitate merchant integration and operation for Swish. Banks have agreements with the merchants who in turn may have an agreement with a partner.
Merchant	A merchant is a company, association or organisation which receives payments via Swish. Merchants sign Swish agreements with their respective bank.
Merchant Swish Simulator	The Merchant Swish Simulator is a test tool to test the Swish-API.
Consumer	A consumer is a private Swish customer that can use the Swish app on a mobile device.
Swish handel	Swish handel gives the merchants the possibility to use Swish as a payment method in m- and e-commerce. The service is aimed primarily for m- and e-commerce stores, via apps and browsers. Swish handel consist of two different payment solutions; Swish m-commerce and Swish e-commerce, a security solution and a function for refunds. All of them are reachable for the merchants through Swish API. The service can be offered by the banks under a different product name than Swish handel.
Swish m-commerce	Swish payments from a mobile device made either through an app or via a mobile browser on the same mobile device.
Swish e-commerce	Swish payments initiated by the consumer in a browser in equipment other than the mobile device that hosts the Swish app.
Swish customer	This is any customer to Swish, either a consumer (person) or a merchant.
Payee	This is the Swish customer that receives the payment
Payer	This is the Swish customer that makes the payment
Alias	A unique identifier for a Swish customer. For a consumer it is the mobile number and for a merchant it is the Swish number.
Payment request	A payment request is a transaction sent from a merchant to the Swish system to initiate an e-commerce or m-commerce payment.
Payee Payment Reference	A payee payment reference is the merchant's own identifier of the transaction/order to be paid. It is sent to the Swish system as a parameter to the payment request and is later returned in confirmation messages.

Refund	A refund is a transaction sent from the merchant to the Swish system to return the whole amount or part of a payment. The reference to the original payment must be provided.

1.3 Document purpose

The integration guide is for anyone who wishes to understand and implement Swish handel in their services and systems. The integration guide explains how to connect to the Swish API and includes information about the payment and refund options related to the Swish handel service. More information about the service can be found at https://www.getswish.se/handel.

1.4 Swish overview

By enrolling to the service Swish handel at the merchant's bank and getting access to the Swish API, merchants can handle payments in e-commerce and m-commerce scenarios in a way which is very convenient and familiar to millions of Swedish consumers. The service builds on the ease-of-use of the person-to-person payment service. Enrolled to the service the merchant can receive payments from all private persons using Swish.

It is also possible for merchants to make refunds in real time using Swish using the API. Some banks will also provide the possibility to initiate refunds from the bank's digital channels.

In brief, a payment involves the following steps:

- The merchant creates a payment request using the Swish API that the consumer views and accepts in the Swish app.
- The consumer and the merchant receive payment confirmations instantly when the amount has been transferred from the consumer's to the merchant's account. For security reasons the payment request is only valid during a limited period time for the consumer in the Swish app.

When enrolling to the service, the merchant obtains a Swish alias to one of the merchant's bank accounts. The merchant will also give authorize Certificate Point of Contact persons during enrollment. These persons will use the Swish Certificate Management System to manage digital certificates, which is one component of securing the access to the API.

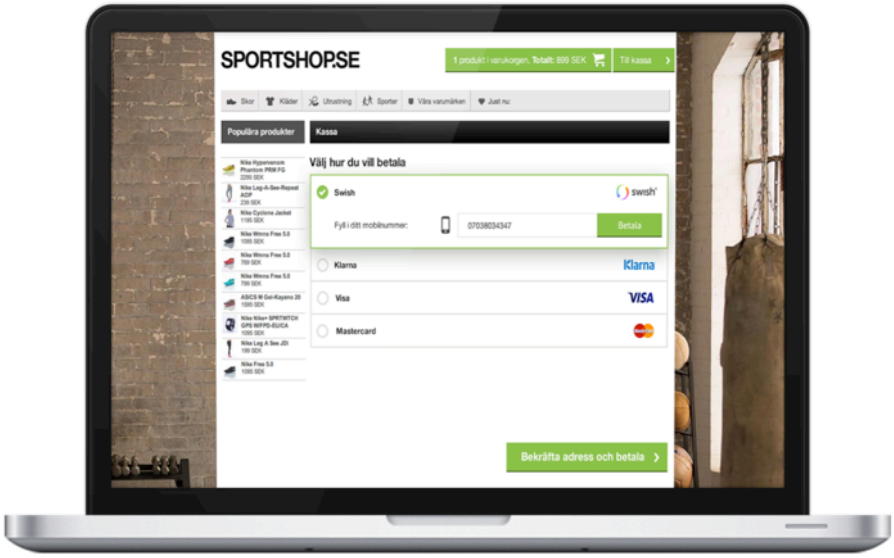
The business transaction when a payment is made using Swish is between the merchant and the consumer and this transaction implies that the consumer makes an advance payment for purchased goods or services.

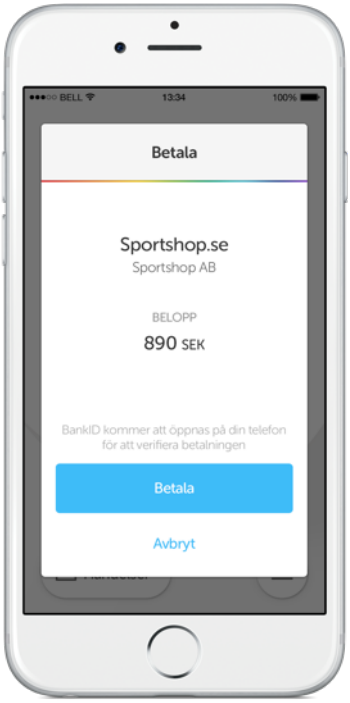
1.5 Payment

It is always the consumer that initiates a payment, and there are two ways to do it; Swish e-commerce or Swish m-commerce.

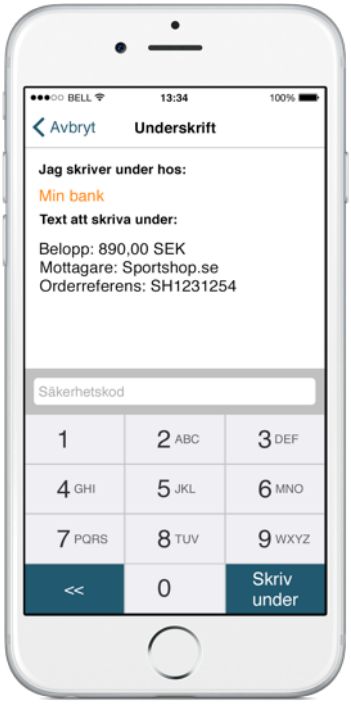
1.5.1 Swish e-commerce

It is always the consumer that initiates a payment, and there are two ways to do it; Swish e-commerce or Swish m-commerce.

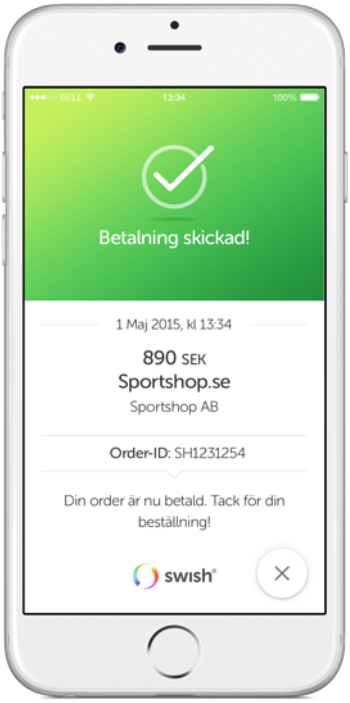




The consumer opens the Swish app, which is preloaded with payment information.



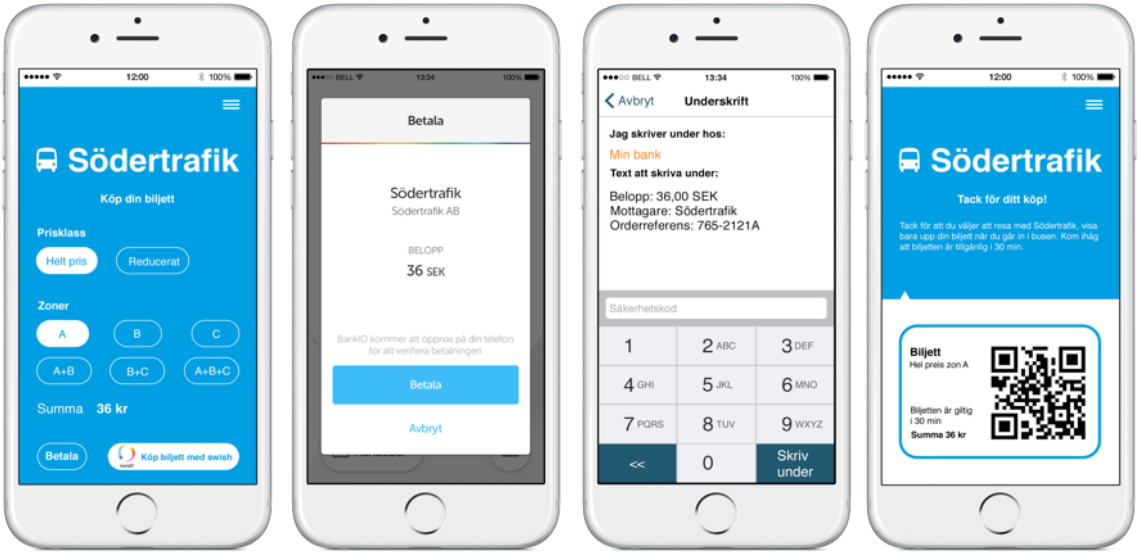
The consumer signs the payment with Mobile BankID



A payment confirmation is shown in the Swish app.

1.5.2 Swish m-commerce

The consumer initiates the payment on the merchant's app using a mobile device. In this case the consumer does need to open the Swish-app.



The consumer chooses to pay with Swish from the merchant's app or website.

The Swish app is activated automatically and preloaded with payment information.

The consumer signs the payment with Mobile BankID.

The consumer gets the payment confirmation in the merchant's app or website.

1.6 Refund

A merchant that has received a Swish payment can refund the whole or part of the original transaction amount to the consumer.

A refund can only be done on an existing payment. The number of refunds on one payment is unlimited, until the total amount reaches the amount of the original payment. A payer Order payment reference ID and message to the consumer can be attached to the refund but these are optional. If the refund is successful, a message will be sent to the payee's app.

A refund can be made on a payment for 12 months.

1.7 Security

In order to protect the Swish API and to ensure the identity of the parties, the security solution encrypts the traffic and authenticates the identities of the merchant and Swish server.

The security solution is implemented as PKI based TLS client/server certificates, where the certificates are issued upon order by the merchant or someone appointed by the merchant. A certificate is valid for 2 years.

A merchant appoints up to 5 persons via their bank, who will be able to logon to an administrative GUI connected to the security solution (identified by BankID/BxID on card or Mobile BankID). An appointed person can administer their certificates using the GUI. An administration of certificates includes view, order new/download and revoke.

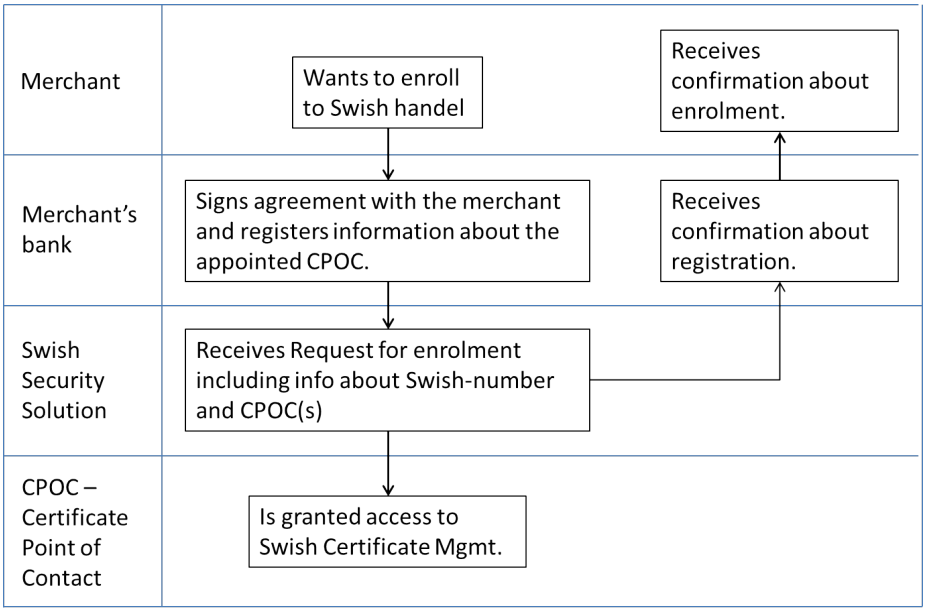
2. Use Cases

The user stories below act as example to increase the understanding of the service Swish handel on a high level.

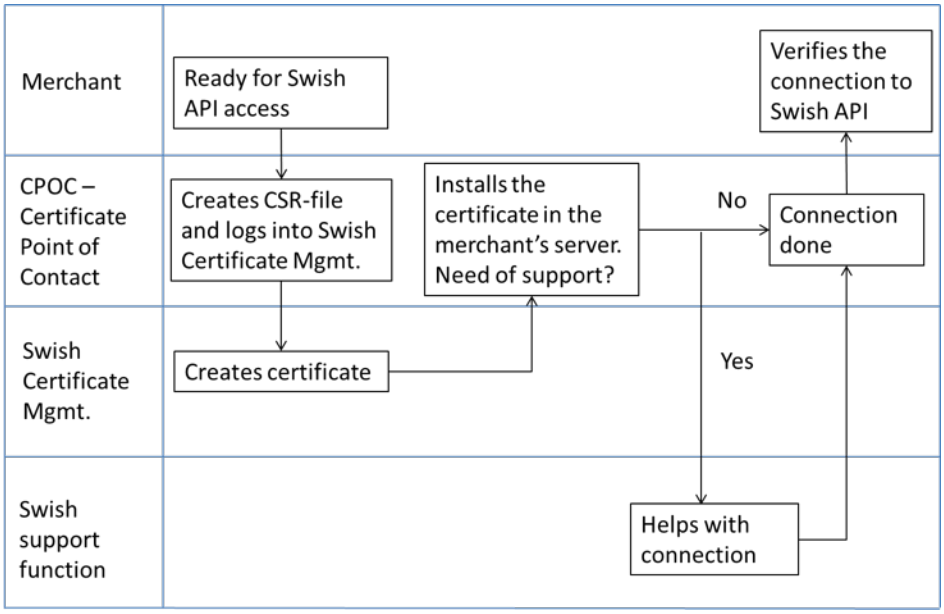
2.1 Swish handel application procedure

This user story describes on a high level how a merchant applies for the service Swish handel.

1. The merchant contacts a Swish connected bank to sign an agreement for the service.
2. The merchant confirms the business terms and signs the agreement with the bank.
    a. The bank obtains and registers the necessary merchant information, including info about the appointed recipients of the API certificate - CPOC (Certificate Point Of Contact). The following personal information is mandatory about the CPOC: Social Security Number, Name, Company Registration Number. Optionally, some banks will also require additional information such as E-mail and phone number.
    b. A Swish number is created for the agreement.
    c. The bank sends an enrollment request to Swish security solution.
3. The security solution receives and registers info about the CPOC connected to the Swish number. The CPOC's are granted access to the certificate management system in the Swish security solution.



4. The merchant is now ready for Swish API access.



5. The merchant or the partner needs to generate a csr-file (Certificate Signing Request). This is normally done by the CPOC.
6. The CPOC logs in to the certificate management system using Mobile BankID, BankID on card or BxID and creates the certificate.
7. The CPOC installs the certificate in the merchant's server and connects it to Swish API. If needed, Swish support function can assist with the connection.
8. The merchant verifies the connection.

2.2 Payment request in the Swish app

2.2.1 Swish m-commerce

The m-commerce flow should be used when the Swish app is on the same device as the merchant's mobile site or app – hence a mobile device.

1. Consumer chooses to pay with Swish for a product or a service in the merchant app.
2. The merchant sends a payment request to the Swish system using the API.
    a. The transaction contains data such as: amount, receiving Swish-number, merchant (payee) payment reference and an optional message to the consumer.
3. The merchant receives a Request Token.
4. Consumer's Swish app is opened automatically by the merchant's site or app showing the payment section that is preloaded with payment information.
    a. The app is opened with the request token as a parameter.
5. Consumer clicks Pay (”Betala”) and the Mobile BankID app opens automatically for signature of the payment transaction.
6. Consumer confirms the payment transaction by signing with the Mobil BankID using his/her password.
7. The amount is transferred in real-time from the consumer's account to the merchant's account.
8. Consumer is linked back to the merchant app for payment transaction confirmation.
    a. Note: the confirmation screen in Swish-app is not displayed in this flow.
9. The merchant receives a confirmation of successful payment.

10. Consumer can view the payment in the events section “Händelser” as a sent payment in the Swish app.

2.2.2 Swish e-commerce

The e-commerce flow should be used when the Swish app is on another device than the merchant’s site.

1. Consumer chooses to pay with Swish for a product or a service at the merchant website and enters his/her mobile phone number which is enrolled to Swish.
2. Information on the merchant website should inform the consumer to open the Swish app to confirm the transaction.
3. The merchant sends a payment request to the Swish system using the API.
    a. The transaction contains data such as: amount, receiving Swish-number, consumer’s mobile phone number, merchant payment reference and an optional message to the consumer.
4. Consumer opens the Swish app showing the payment section, which is preloaded with the information in the payment request.
5. Consumer clicks Pay (”Betala”) and the Mobile BankID app opens automatically for signature of the payment transaction.
6. Consumer confirms the payment transaction by signing with the Mobil BankID using his/her password.
7. The amount is transferred in real-time from the consumer’s account to the merchant’s account.
8. Consumer receives a payment confirmation in the Swish app.
9. The merchant receives a confirmation of successful payment.
10. Consumer receives a payment confirmation at the merchant website.
11. Consumer can view the payment in the event section “Händelser” as a sent payment in the Swish app.

2.2.3 Reject payment

Consumer wants to dismiss the payment request in the Swish app and clicks on “Avbryt”. The rejected payment request is deleted from the Consumer’s Swish app.

2.3 Refund

There are two ways to make a refund: through the bank channel or through the API channel. This section describes the API channel on a high level.

As a merchant, I have received a Swish payment and wish for some reason to refund the whole or part of the original amount to the payer.

The merchant chooses which payment that is to be completely or partially refunded. The merchant specifies the refund amount and sends the refund.

The merchant will also be able to send its own information that will be shown to the payer in the event section in the Swish app, and also on the payee bank account statement. The information will also be used by the company for tallying.

The merchant receives a confirmation that the refund has been completed.

As recipient of a refund, the payee receives a payment notification in the Swish app. The payment notification is marked "Återbetalning" in the event section in the Swish app.

Alternative flow – ”payment notification”:
In cases when the recipient of a refund cannot receive data push notifications at the moment, the refund will be visible in the Swish app next time the recipient logs in. The refund will also appear in the recipient´s bank account statement.

Alternative flow – ”refund receiver not connected to Swish”:
In cases when the recipient of a refund has terminated the Swish agreement since the original payment occurred, the merchant will receive an error message stating that the refund cannot be processed. Refund must in this case be done via another channel.

2.4. Termination Swish handel

1. The merchant terminates the agreement with the bank. The service will stop working.
2. The merchant is responsible for termination/revocation of the API certificates.
3. Refunds will not be possible to do on the terminated Swish number.

3. Technical Requirements

The Swish server requires TLS 1.1 or higher.

The merchant must be able to receive the callback HTTPS POST request from the Swish server on port 443.

4. Merchant Setup Process

4.1 Technical Integration

In order to integrate a merchant commerce solution with Swish API the merchant needs to get a client TLS certificate from Swish Certificate Management and install it on their server. The certificate will be used for client authentication of TLS communication with Swish API. The following steps need to be performed:

1. Generate a pair of 2048 bits RSA keys on your server and create a certificate request (CSR) in a PKCS#10 format.
   This step depends on the type of web server solution that is used and differs between different types of servers. The keys are usually generated to a so-called *keystore* (e.g. Java keystore, Microsoft Windows keystore) or file (e.g. openSSL on Apache/Tomcat). For details please consult your web solution documentation or your supplier.

   Note: The following examples are to be considered regarding secure handling of cryptographic keys and certificates. The Customer’s keys should be installed by the Customer in secure cryptographic units or should be protected in a similar manner. The keys should only be installed on units necessary for production and back-up purposes. The keys should be deleted at all instances when no longer operational. The keys should at all times be stored with strong encryption and protected using passwords or more secure procedures, e.g. smart-cards. Passwords used to protect the keys should be handled two jointly and are to be stored in a secure manner so they cannot be lost or subjected to unauthorized access.

It is highly important to protect the private key from unauthorized access. It is recommended to protect the keys with a password if your server provides this option. Care should be taken to protect the passwords as well.

There are no requirements on the content of the CSR (names or other parameters), except for the keys that need to be 2048-bit RSA.

It is possible to install the same certificate on several servers (depending on technical server setup, but no license limitations), or to issue one key pair and certificate per server.

2. Login to Swish Certificate Management at https://getswishcert.bankgirot.se by using mobile BankID, BankID on card or BxID. Only the person(s) registered by the bank for a specific merchant will be able to perform this step.

3. Provide the organizational number of the merchant and the Swish number for which a certificate is to be generated.

4. Choose tab "New certificate" and paste the content of the generated CSR into the text field. Choose whether the certificate should to be in PKCS#7 or PEM format. Consult your documentation regarding which format suits your solution.

5. A new certificate is generated and provided on the screen. Copy the text string and save it to a file. The response (PKCS#7 or PEM) will contain your client certificate and all CA certificates up to the Swish root.

6. Import the generated certificate and all CA certificates to your server. For details on how to perform this step consult your web solution documentation or your supplier.

7. The Swish server is set up with a TLS server certificate, which needs to be verified when initiating TLS from your web server to Swish. Choose to trust Swish Root CA (o=Getswish AB, ou=Swish Member CA, cn=Swish Root CA v1). The certificate chain for the Swish server TLS certificate, i.e. the Swish Root CA certificate and the Intermediate CA certificate, are available via the Certificate Management GUI via the link “Download Swish server TLS certificate”. For details on how to perform this step consult your web solution documentation or your supplier.

After performing the steps 1 - 7 you should be able to set up TLS with the Swish API.

Note: It is necessary provide the generated certificate together with all CA certificates up to the Swish Root CA in order to correctly set up a TLS session with the Swish API.

Note: No error messages will be returned before a TLS session is successfully established with the Swish API. This means that if the wrong certificate has been used, if the validity time of the certificate has expired, or if the certificate has been revoked, no indication of this is given.

Note: It is recommended to require verification of the Swish API TLS certificate and not to ignore this verification, in case your server allows you to disable server certificate verification.

4.2 Managing certificates

Login to Swish Certificate Management at https://getswishcert.bankgirot.se by using mobile BankID, BankID on card or BxID. Only the person(s) registered by the bank for a specific merchant will be able to perform this step.

Provide the organizational number of the merchant and the Swish number for which a certificate is to be managed.

After logging in a list is provided with all certificates associated with the specific merchant and Swish number, and the status of them. By clicking on “Download” it is possible to see further details and to attain the certificate again.

4.3 Revoking a certificate

If the integrity of the merchant’s private key has been compromised, if a certificate has been replaced by a new one, if the service has been terminated, or if the merchant needs to revoke a certificate for some other reason, this can be done via the Swish Certificate Management.

Login to Swish Certificate Management at https://getswishcert.bankgirot.se by using mobile BankID, BankID on card or BxID. Only the person(s) registered by the bank for a specific merchant will be able to perform this step.

Provide the organizational number of the merchant and the Swish number for which a certificate is to be revoked.

After logging in a list is provided with all certificates associated with the specific merchant and Swish number, and the status of them. By clicking on the trash can it is possible to revoke a specific certificate.

Please be aware that the certificate is irreversibly revoked and that revoking a certificate that is in use may lead to an interruption of the service.

5. Launching

5.1 Detecting if Swish app is installed on the device

5.1.1 Detection by 3rd party app

3rd party apps, excluding web browsers, can detect if the Swish app is installed on the device. Below are the code snippets that show this. Notice that WinPhone application cannot detect the is Swish installed not run it.

iOS (detect if application is installed and call it)

```
static inline bool isSwishAppInstalled(void)
{
   return [[UIApplication sharedApplication
                          canOpenURL:[NSURL URLWithString:@"swish://"]];
}
```

Android (detect if application is installed)

```
//Swish package name is "se.bankgirot.swish"


protected boolean isSwishAppInstalled(Context _context, String
SwishPackageName) {
  boolean isSwishInstalled = false;
  try {
   _context.getPackageManager().getApplicationInfo(SwishPackageName, 0);
    isSwishInstalled = true;
```

WinPhone (detect if application is installed and call it)

```
// The URI to launch
string uriToLaunch = @"swish://paymentrequest<parameters>";
// Create a Uri object from a URI string
var uri = new Uri(uriToLaunch);
// Launch the URI
async void DefaultLaunch(){
   // Launch the URI
   var success = await Windows.System.Launcher.LaunchUriAsync(uri);
   if (success) {
```

5.1.2 Detection with mobile web browsers

The investigation of the abilities to determine if Swish is installed on a device shows that there is an absence of a standard, documented way to do this from the web browsers. The found workaround is based on the time during which the return to the browser was performed. The idea of this approach is that JavaScript code on current page will be frozen right after calling Swish application because the control flow will be given to the Swish if it start successfully and control flow will be returned back to JavaScript when Swish will be finished, thus if the JavaScript code continue executing after short time from the moment of the trying to call Swish this means, that Swish is not installed on the device. The JavaScript code snippet given below:

```
//remember the time of start application
timestart = new Date().getTime();
//try to run application (Swish) in the frame by opening custom URL-SCHEME
createIFrame(url+"&browser="+browserName+"&back="+encodeURIComponent(location.toString())
+"&useragent="+encodeURIComponent(userAgent));
//remember time of returning from application
timeend = new Date().getTime();
//if from the moment of the attempt to run the application to moment when the
//control returns back to this code passed enough much time (more then 3 sec),
//most probably this means that the application was successfully started and
//the user spent the time using the application
if(timeend - timestart > 3000) {
                isSwishInstalled = true;
} else {
                isSwishInstalled = false;
}
```

But, as investigations show, this approach does not work for WinPhone and Android Chrome version 25 and newer. Moreover, because Androids default browser is now based on Chrome core, this is also true for default browser. In the 3 variants, WinPhone browser (Internet Explorer), Android default platform browser and Android Chrome from version 25, they do not immediately return the control to the JavaScript code when the called application (Swish) is not installed on the device, instead they open a dialog and offer the user to go to the platform's market to download the required application (Swish). This means, that in this case the browser will either successfully open Swish or ask the user to go to the market to download and install Swish.

5.1.3 Call Swish app and send parameters

The 3rd party apps, including mobile web browsers, will call the Swish app using a Custom URL Scheme. The 3rd party app has to send the Swish app a Payment request token and some application identifier(s) of itself that the Swish app can use to return back to the initial caller.

The application identifier, in the case of a native iOS application, is application bundle id. For native Android application it is package name. For browser it will be “user agent”. The 3rd party application also have to send a “callback URL”, the string that Swish, or BankID, will use as parameter for callback, the goal of this parameter is to force the application to open a given GUI view and for a browser, to open a given URL.

The code snippets for each platform are given below:

iOS native application calls Swish:

The 3rd party apps will call the Swish app using a custom URL Scheme "swish://paymentrequest". The 3rd party app has to send the Swish app a payment request token and provide a callback URL to return after payment request processing. Both parameters are required. Thus, a correct URL to open Swish app is:

swish://paymentrequest?token=<valid_token>&callbackurl=<valid_callback_URL>

Note that <valid_token> is a placeholder for valid payment request token value, which should be received from server before calling Swish app.

Note that <valid_callback_URL> is a placeholder for valid callback URL value. To provide callback URL a merchant app must register a unique URL scheme. In Xcode select the project and in the Info tab expand

URL Types and add the URL scheme m_com_app. Callback URL should be passed in URL-encoded format.

Note: m_com_app is an example. The merchant app should use it's own unique URL scheme.

Assume we have registered URL scheme "merchant://" (for example) and token received from server. To open Swish app from merchant app the following code could be used:

```
// character set is used to encode callback URL properly
NSCharacterSet *notAllowedCharactersSet =
[NSCharacterSet
characterSetWithCharactersInString:@"!*'();:@&=+$,/?%#[]"];

NSCharacterSet *allowedCharactersSet =
[notAllowedCharactersSet invertedSet];

NSString *callbackURLStr = @"merchant://";

NSString *encodedCallbackURLStr =
[callbackURLStr
stringByAddingPercentEncodingWithAllowedCharacters:allowedCharactersS
et];

NSString *swishURLStr = [NSString
stringWithFormat:@"swish://paymentrequest?token=%@&callbackurl=%@",
token, encodedCallbackURLStr];

NSURL *swishURL = [[NSURL alloc] initWithString: swishURLStr];

if ([[UIApplication sharedApplication] canOpenURL:swishURL]) {
  if ([[UIApplication sharedApplication] openURL:swishURL]) {
    // Success
  }
  else {
    // Error handling
  }
}
else {
  // Swish app is not installed
  // error handling
}
```

The merchant app must implement the following function that will be called when the merchant app is re-launched.

-(BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)annotation;

Android native application calls Swish:

```
public static boolean startSwish(Activity activity, String token,
                                 String callBackUrl, int requestCode) {
  If ( token == null || token.length() == 0 || callBackUrl == null ||
        callBackUrl.length() == 0 || activity == null) {
    return false;
  }
  Uri scheme =
Uri.parse("swish://paymentrequest?token="+token+"&callbackurl="+callB
ackUrl);
  Intent intent = new Intent(Intent.ACTION_VIEW, scheme);
  intent.setPackage("se.bankgirot.swish");
```

Handling result
App will start callBackUrl when payment is complete (accepted or rejected).
If no CallBackUrl will be specified app can check with

```
@Override
protected void onActivityResult(int requestCode, int
resultCode, Intent data)
```

resultCode will be `RESULT_OK` or `RESULT_CANCELED`

WinPhone native application call Swish:

```
// Create the URI string
var uriToLaunch =
string.Format("swish://paymentrequest?token={0}&callbackurl={1}",
<INSERT TOKEN HERE>, Uri.EscapeDataString("fp-app-x://"));
// Create the URI to launch from a string.
```

If the Swish app is not present on the device the operating system presents a dialogue asking to open Windows Phone store. Merchant app must inform the user.

The Merchant app must register a unique URL scheme to make it possible for the Swish app to re-launch Merchant app. In Visual Studio:

1. Open Package.appxmanifest
2. Open the tab Declarations.
3. Add a "Protocol". Under name enter ”rp_app_x”.
4. Enter a logo and a "Display name".

Note: rp_app_x is an example, Merchant should use its own unique URL scheme.

Merchant must also implement the following to be successfully re-launched by Swish App. In Visual Studio add the class AssociationUriMapper:

```
/// <summary>

/// The association uri mapper.

/// </summary>

internal class AssociationUriMapper : UriMapperBase {

  /// <summary>

  /// When overridden in a derived class, converts a requested
uniform resource identifier (URI) to a new URI.

  /// </summary>

  /// <returns>

  /// A URI to use for the request instead of the value in the
<paramref name="uri"/> parameter.

  /// </returns>

  /// <param name="uri">The original URI value to be mapped to a new
URI.</param>

  public override Uri MapUri(Uri uri) {
```

In App.xaml.cs, add AssociationUriMapper as UriMapper by adding the following line to the method InitializePhoneApplication:

```
// Assign the URI-mapper class to the application frame.

RootFrame.UriMapper = new AssociationUriMapper();
```

Possible Swish URI string:

1. Use Token and callback address (start Swish app and return to Merchant app after)
   `swish://paymentrequest?token=<INSERT TOKEN HERE>&callbackurl=fp-app-x://`

2. Use Token (start Swish app without return to Merchant app)
   `swish://paymentrequest?token=<INSERT TOKEN HERE>`

3. Use callback address (start Swish app only)
   `swish://callbackurl=fp-app-x://`

4. Without parameters (start Swish app only)
   `swish://`

5. Use Token and callback url (start Swish app and return to Merchant app with parameters)
   `swish://paymentrequest?token=<INSERT TOKEN HERE>&callbackurl=fp-app-x://<PARAMETERS>`

JavaScript calls Swish:

```
swish://paymentrequest?token=value&callbackurl=back_scheme
```

For Chrome version >= 24:

```
obj.href="intent://view/#Intent;<scheme_name>;<package_name>;<payment_reques
t_token>;<browser_name>;<call_back_url>;<user_agent>;end"
```

Directly returning back to the calling app from the BankID app

The Swish app will call the BankID app when the user accepts the payment request. However, when the user has completed the signing process, in the M-Commerce case, the BankID app should return directly to the app that originally called the Swish app (e.g. merchant app or browser), without going back to the Swish app first.

The investigation of the abilities to satisfy this requirement shows that the implementation will be platform dependent.

For iOS it will be implemented by passing a callback URL as a parameter to the BankID app, in the same way as it is now done to return from the BankID app to the Swish app. In other words, the Swish app will prepare a correct callback URL-SCHEME based on the information received from the caller and then pass this to the BankID app.

iOS code examples

Current Swish code to call BankID and return back to Swish looks like:

```
NSString* bankIDUrl = [NSString stringWithFormat:@"bankid://redirect=swish://%@",
rndStr];
...
openURL:[NSURL URLWithString: bankIDUrl]];
```

To force the BankID app to return to the app that originally called it, it needs to pass the BankID app a correct “redirect” URL. For example, to call the Google Chrome browser, the code will look like this:

```
NSString* bankIDUrl = [NSString stringWithFormat:@"bankid://redirect=googlechrome://%@",
CallbackURL];
...
openURL:[NSURL URLWithString: bankIDUrl]];
```

For Android, as well as for WinPhone, there are two ways to implement the required behaviour. The first option is that the BankID app will return to the Swish app, in the same way as it works now, and then the Swish app will handle this return as an immediate return to the original caller, without showing any view. The second option is that the Swish app can remove its view from the navigation stack, by destroying that activity in Android or closing the app in WinPhone, and then don’t send any callback URL to the BankID app. This means that the original caller will be in the foreground right after the user finishes using the BankID app. So for Android and WinPhone there is no need for any specific way to call BankID and the return to the original caller will be done by code in the Swish app.

6. Test Environment

A Merchant Swish Simulator is available for merchants to test their integration with Swish API. The Merchant Swish Simulator will validate requests and return simulated but correctly formatted responses. The Merchant Swish Simulator will return a simulated result of the request in the callback URL. It is also possible to retrieve the payment request status, and to simulate different error situations.

A user guide for the Merchant Swish Simulator, can be be found at https://www.getswish.se/handel.

7. Production Environment

The Swish server IP address for IP filtering:

194.242.111.220:443

Swish API URL:

https://swicpc.bankgirot.se/swish-cpcapi/api/v1/paymentrequests

https://swicpc.bankgirot.se/swish-cpcapi/api/v1/refunds

Swish server TLS certificate is issued under the following root CA that should to be configured as trusted:

cn=Swish Root CA v1

ou=Swish Member Banks CA

o=Getswish AB

The complete certificate chain of the Swish server TLS certificate is available through Swish Certificate Management.

8. Guidelines for using the Swish API

When integrating with Swish API the following guidelines will support stable performance of the system and a smooth consumer experience.

8.1 Consumer in control of payment requests

Each payment request transaction sent to the API must be initiated by a physical paying consumer. The merchant must make sure that the consumer does not receive what he/she perceives as “spam” or unwanted payment requests.

8.2 Use the call-back for payment requests and refunds

When sending the payment request or refund a call-back is provided to the merchant of the status of the payment. This is the normal usage scenario, which should be used in most cases.

As a backup there is also a “Retrieve” for Payment Requests and Refunds for reconciliation in the case that the normal call-back fails for some reason. Note that this is a backup – and not the standard flow for receiving payment status.

8.3 Refund transactions – avoid large batches

The “create refund” message is intended for real-time one-by-one calls. It is not intended for batching up a large quantity and then sending the whole batch in a short period of time.

There should be at least 1 second between each refund transaction and if more than 100 transactions are to be sent in a sequence they should be sent during night time.

8.4 Renewal of Client TLS Certificate

The validity of the client TLS certificate is two years. It is the merchant's responsibility to generate new keys and certificate in due time, prior to the expiry of the old certificate, in order to ensure uninterrupted functionality of the commerce site. The merchant could authorise another company (a partner to the merchant) to manage the certificate renewal process.

8.5 Displaying the Swish alias to consumers

When enrolling to Swish the merchant will receive a Swish alias (123 XXX YYYY) which uniquely identifies the enrolment and which is used as an alias to the payee’s bank account.

We recommend e-commerce and m-commerce merchants not to expose this to consumers since it

1. Can be used for unprompted payments by entering the Swish alias in the Swish app.
2. Some banks may block unprompted payments to Swish aliases enrolled to “Swish Handel”

The Swish alias for transactions generated by payment requests or refunds will not be displayed by the Swish app or the bank’s consumer interfaces.

9. Versioning the Web Service API

9.1 Versions

Changes may be made to the API to correct errors or to introduce new functionality. When changed, a new version of the API will be made available via a new URL. Merchant should always use the latest version of the API.

The general rule is that old versions of the API will be discontinued two years after the release of the successor. But if deemed necessary, for example for security reasons, a version of the API may be discontinued prematurely. As new functionality is introduced to the system the behaviour of an existing version of the API may change, e.g. existing faults may also be used in new situations.

10. Support

10.1 Deployment support

Please see the manuals and FAQ available at https://www.getswish.se/handel

If you can't find the technical information you need, you can contact the deployment support organisation. The contact details are also published at https://www.getswish.se/handel.

For all commercial questions, please contact your bank.

10.2 Operating information

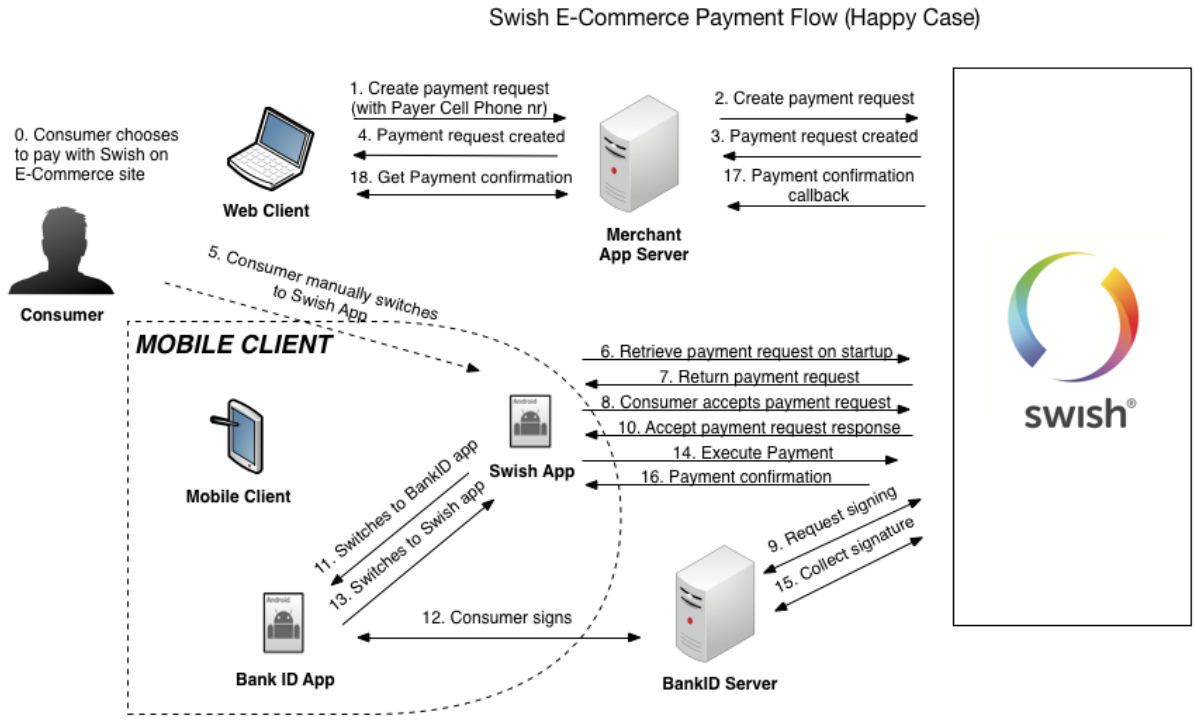
Operating information is available at https://getswish.se/driftsinformation

11. Swish API Description

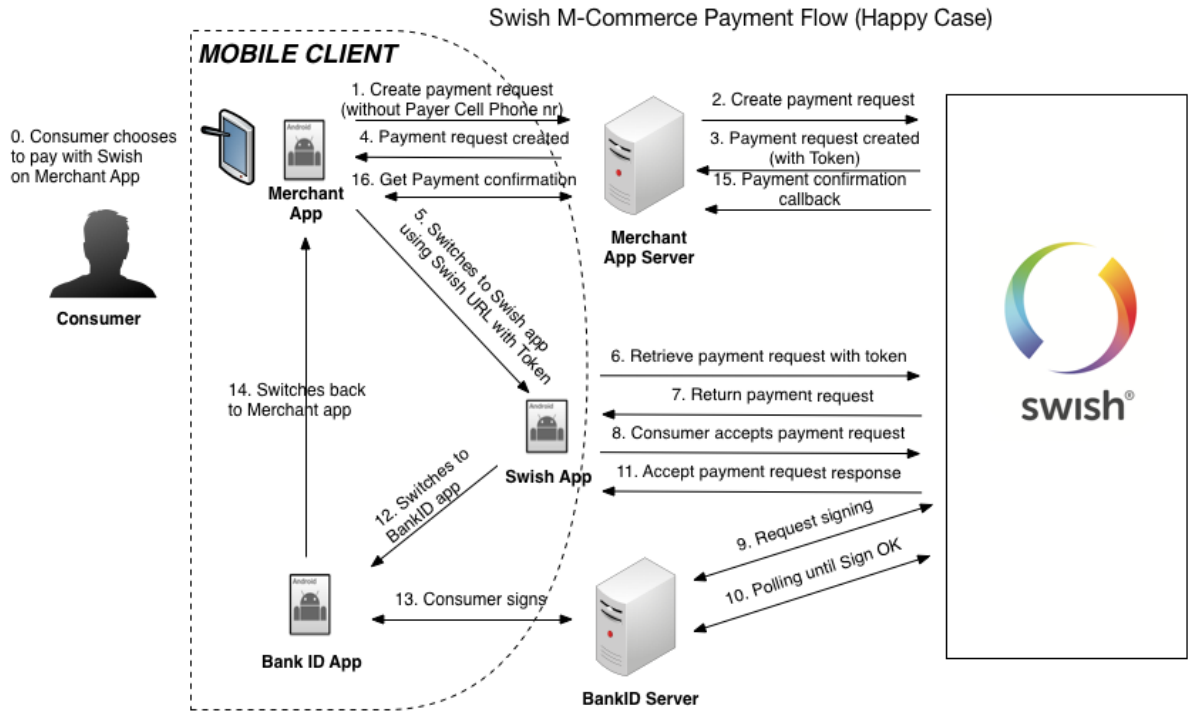
11.1 Payment Request

Payment requests are created/retrieved with the operations listed below. There are two main flows to this use case, one for Swish m-commerce and one for Swish e-commerce. The main difference is that in the Swish e-commerce case the consumer is prompted for his/her mobile phone number, and then the consumer has to manually open the Swish app. But in the Swish m-commerce case the consumer's mobile phone number is initially not known to the merchant. So instead, in this case, the API returns a Payment request token. This token is used to build a so called Swish URL, which the merchant can use to call the Swish app from their app. The Payment request token is then a parameter to the Swish URL. The result of the payment request is then later returned asynchronously in a Callback URL to the merchant. Alternatively, the merchant can manually retrieve it. The recommended approach is to receive it with the Callback URL. These flows are illustrated in the pictures below.

11.1.1 Swish e-commerce



11.1.2 Swish m-commerce



11.1.3 Create Payment Request

`POST` `/api/v1/paymentrequests`

The Http request body have to contain a Payment Request Object.

Potential Http status codes returned:

- 201 Created: Returned when Payment request was successfully created. Will return a Location header and if it is Swish m-commerce case, it will also return PaymentRequestToken header.
- 400 Bad Request: Returned when the Create Payment Request operation was malformed.
- 401 Unauthorized: Returned when there are authentication problems with the certificate. Or the Swish number in the certificate is not enrolled. Will return nothing else.
- 403 Forbidden: Returned when the payeeAlias in the payment request object is not the same as merchant's Swish number.
- 415 Unsupported Media Type: Returned when Content-Type header is not "application/json". Will return nothing else.
- 422 Unprocessable Entity: Returned when there are validation errors. Will return an Array of Error Objects.
- 500 Internal Server Error: Returned if there was some unknown/unforeseen error that occurred on the server, this should normally not happen. Will return nothing else.

Potential Error codes returned on Error Objects when validation fails (HTTP status code 422 is returned):

- FF08 - PaymentReference is invalid
- RP03 - Callback URL is missing or does not use Https
- BE18 - Payer alias is invalid
- RP01 - Missing Merchant Swish Number
- PA02 - Amount value is missing or not a valid number
- AM06 - Specified transaction amount is less than agreed minimum
- AM02 - Amount value is too large
- AM03 - Invalid or missing Currency
- RP02 - Wrong formated message
- RP06 - A payment request already exist for that payer. Only applicable for Swish e-commerce.
- ACMT03 - Payer not Enrolled
- ACMT01 - Counterpart is not activated
- ACMT07 - Payee not Enrolled

Create Payment request example (Swish e-commerce):

```
curl -v --request POST https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/paymentrequests \
    --header "Content-Type: application/json" \
    --data @- <<!
{
    "payeePaymentReference": "0123456789",
    "callbackUrl": "https://example.com/api/swishcb/paymentrequests",
    "payerAlias": "4671234768",
    "payeeAlias": "1234760039",
    "amount": "100",
    "currency": "SEK",
    "message": "Kingston USB Flash Drive 8 GB"
}
!

< HTTP/1.1 201 Created
< Location: https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/paymentrequests/AB23D7406ECE4542A80152D909EF9F6B
```

Create Payment request example (Swish m-commerce):

```
curl -v --request POST https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/paymentrequests \
    --header "Content-Type: application/json" \
    --data @- <<!
{
    "payeePaymentReference": "0123456789",
    "callbackUrl": "https://example.com/api/swishcb/paymentrequests",
    "payeeAlias": "1234760039",
    "amount": "100",
    "currency": "SEK",
    "message": "Kingston USB Flash Drive 8 GB"
}
!

< HTTP/1.1 201 Created
< Location: https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/paymentrequests/AB23D7406ECE4542A80152D909EF9F6B
< PaymentRequestToken: f34DS34lfd0d03fdDselkfd3ffk21
```

The PaymentRequestToken is then used to open the swish app, using the Custom URL Scheme e.g.:

```
swish://paymentrequest?token=f34DS34lfd0d03fdDselkfd3ffk21
```

11.1.4 Retrieve Payment Request

`GET` `/api/v1/paymentrequests/{id}`

Potential HTTP status codes returned:

- 200 OK: Returned when Payment request was found. Will return Payment Request Object.
- 401 Unauthorized: Returned when there are authentication problems with the certificate. Or the Swish number in the certificate is not enrolled. Will return nothing else.
- 404 Not found: Returned when the Payment request was not found or it was not created by the merchant. Will return nothing else.
- 500 Internal Server Error: Returned if there was some unknown/unforeseen error that occurred on the server, this should normally not happen. Will return nothing else.

Retrieve Payment request example (Swish m-commerce):

```
curl -v --request GET https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/paymentrequests/AB23D7406ECE4542A80152D909EF9F6B

< HTTP/1.1 200 OK
{
    "id": "AB23D7406ECE4542A80152D909EF9F6B",
    "payeePaymentReference": "0123456789",
    "paymentReference", "6D6CD7406ECE4542A80152D909EF9F6B",
    "callbackUrl": "https://example.com/api/swishcb/paymentrequests",
    "payerAlias": "07211234567",
    "payeeAlias": "1231234567890",
    "amount": "100",
    "currency", "SEK",
    "message": "Kingston USB Flash Drive 8 GB",
    "status": "PAID",
    "dateCreated": "2015-02-19T22:01:53+01:00",
    "datePaid": "2015-02-19T22:03:53+01:00"
}
```

11.1.5 Callback

Swish will make a callback HTTPS POST request, with the Payment Request Object, to the Callback URL supplied in the Create Payment Request operation when either of the following events (status) happens:

- PAID - The payment was successful
- DECLINED - The payer declined to make the payment
- ERROR - Some error occurred, like payment was blocked, payment request timed out etc. See list of error codes for all potential error conditions.

A payment request has to be accepted or declined by the consumer within eight minutes for e-commerce and five minutes for m-commerce. When the time has elapsed an ERROR status is returned to the Callback URL. If the consumer accepts the payment request a status is returned to the Callback URL within 12 seconds.

The callback endpoint has to use HTTPS and we highly recommend IP filtering as well. It is however up to the merchant to make sure the endpoint is available. Swish will only make the callback request once, if the merchant has not received a callback response after the timeout, the merchant can choose to call the Retrieve Payment Request. Swish will always try to make a callback request before the timeout period, but if it times out, then a timeout callback is sent with status ERROR and the error code will have value TM01.

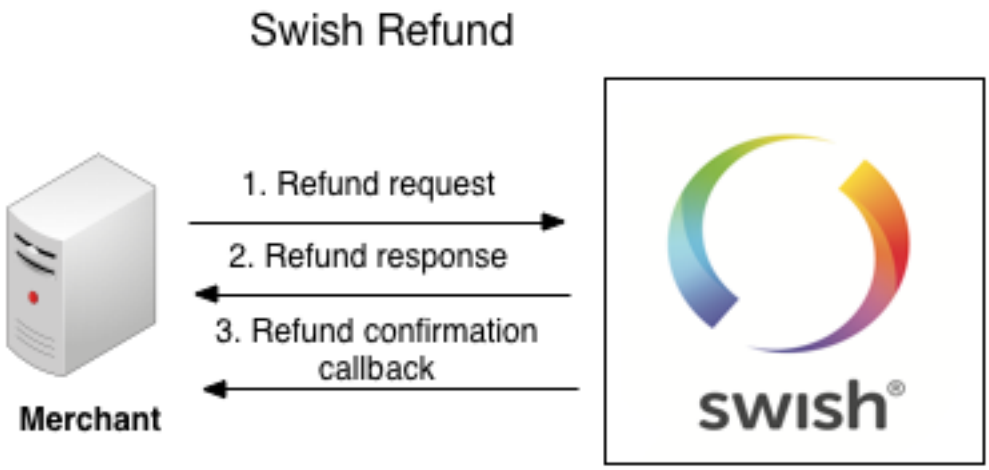
11.2 Payment Refund

Refunds are initiated based on a Payment reference from an earlier payment. To make a refund create first a Refund, similar to how you create a payment request, and then you will receive a reference to the refund, but the result of the refund is returned in a Callback, similar to how payment request works. A refund normally completes much faster than a payment request, but a callback is used because the actual payment might take a long time, normally it does not, but since it might, the result is returned asynchronously in the callback. The callback, in the happy case, will return an intermediate response with status DEBITED. This response is guaranteed to have returned in under 10 s or you will get an ERROR response. The DEBITED response means that the money has been taken from the merchants (payers) account, but has not been put into the payees account yet. Normally this should happen very soon afterwards, but this "might" take a long time. Moreover, it is not guaranteed to succeed, in other words the receiving bank might refuse to put money into the account. In that case the Commerce customer will receive an ERROR response and the money is put back into the Commerce customers account. So these are the potential callback scenarios:

1. Happy case: DEBITED, PAID
2. Early error: ERROR
3. Late error: DEBITED, ERROR

So in other words there is a tradeoff here, between speed and accuracy that the merchant needs to make:

1. Use the early fast guaranteed response of DEBITED to give a quick response that might turn out to be inaccurate later on.
2. Ignore the DEBITED response and wait for the PAID response that is always accurate but not always fast.



11.2.1 Create Refund

`POST` `/api/v1/refunds`

The Http request body have to contain a Refund Object.

Potential Http status codes returned:

- 201 Created: Returned when Refund was successfully created. Will return a Location header.
- 400 Bad Request: Returned when Create refund POST operation was malformed.
- 401 Unauthorized: Returned when there are authentication problems with the certificate. Or the Swish number in the certificate is not enrolled. Will return nothing else.
- 403 Forbidden: Returned when the payerAlias in the refund object is not the same as merchant's Swish number.
- 415 Unsupported Media Type: Returned when Content-Type header is not "application/json". Will return nothing else.
- 422 Unprocessable Entity: Returned when there are validation errors. Will return an Array of Error Objects.
- 500 Internal Server Error: Returned if there was some unknown/unforeseen error that occurred on the server, this should normally not happen. Will return nothing else.
- 504 Gateway Timeout: Returned when the Bank validation answers take too long and Swish times out. This rarely happens.

Potential Error codes returned on Error Objects when validation fails (Http status code 422 is returned):

- FF08 – Payment Reference is invalid
- RP03 - Callback URL is missing or does not use Https
- PA02 - Amount value is missing or not a valid number
- AM06 - Specified transaction amount is less than agreed minimum
- RF08 - Amount value is too large or amount exceeds the amount of the original payment minus any previous refunds. Note: the remaining available amount is put into the additional information field.
  Note: the remaining available amount is put into the additional information field.
- AM03 - Invalid or missing Currency
- RP01 - Missing merchant Swish Number
- RP02 - Wrong formated message
- ACMT07 - Payee not Enrolled
- ACMT01 - Counterpart is not activated
- RF02 - Original Payment not found or original payment is more than than 13 months old
- RF03 - Payer alias in the refund does not match the payee alias in the original payment
- RF04 - Payer organization number do not match original payment payee organization number.
- RF06 - The Payer SSN in the original payment is not the same as the SSN for the current Payee. Note: Typically this means that the Mobile number has been transferred to another person.
- RF07 - Transaction declined
- FF10 - Bank system processing error
- BE18 - Payer alias is invalid

Create Refund example:

```
curl -v --request POST https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/refunds \
    --header "Content-Type: application/json" \
    --data @- <<!
{
    "payerPaymentReference": "0123456789",
    "originalPaymentReference": "6D6CD7406ECE4542A80152D909EF9F6B",
    "callbackUrl": "https://example.com/api/swishcb/refunds",
    "payerAlias": "1231234567890",
    "amount": "100",
    "currency": "SEK",
    "message": "Refund for Kingston USB Flash Drive 8 GB"
}
!

< HTTP/1.1 201 Created
< Location: https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/refunds/ABC2D7406ECE4542A80152D909EF9F6B
```

11.2.2 Retrieve Refund

`GET` `/api/v1/refunds/{id}`

Potential HTTP status codes returned:

- 200 OK: Returned when refund was found. Will return Refund Object.
- 401 Unauthorized: Returned when there are authentication problems with the certificate. Or the Swish number in the certificate is not enrolled. Will return nothing else.
- 404 Not found: Returned when no refund was found or it was not created by the merchant. Will return nothing else.
- 500 Internal Server Error: Returned if there was some unknown/unforeseen error that occurred on the server, this should normally not happen. Will return nothing else.

Retrieve Refund example:

```
curl -v --request GET https://swicpc.bankgirot.se/swish-
cpcapi/api/v1/refunds/ABC2D7406ECE4542A80152D909EF9F6B

< HTTP/1.1 200 OK
{
    "id": "ABC2D7406ECE4542A80152D909EF9F6B",
    "payerPaymentReference": "0123456789",
    "originalPaymentReference": "6D6CD7406ECE4542A80152D909EF9F6B",
    "callbackUrl": "https://example.com/api/swishcb/refunds",
    "payerAlias": "1231234567890",
    "payeeAlias": "07211234567",
    "amount": "100",
    "currency": "SEK",
    "message": "Refund for Kingston USB Flash Drive 8 GB",
    "status": "PAID",
    "dateCreated": "2015-02-19T22:01:53+01:00",
    "datePaid": "2015-02-19T22:03:53+01:00"
}
```

11.2.3 Callback

Swish will make a callback HTTPS POST request, with the Refund Object, to the Callback URL supplied in the Create Refund operation when either of the following events (status) happens:

- PAID - The payment was successful
- ERROR - Some error occurred. See list of error codes for all potential error conditions.

11.3 Objects

The date fields use the ISO 8601 date format. Since the Swish server creates these date fields, and the servers are located in Sweden, the timezone used is CET, which is UTC+01:00 or UTC+02:00, depending on whether it is Central European Summer Time (CEST) or not. See the code examples for samples.

11.3.1 Payment Request Object

The Payment Request Object is used in all 3 Payment Request operations (Create, Retrieve and Callback). The fields that are mandatory are for the Create operation, but of course those fields will also be available on the other operations.

Legend:

- M - Mandatory input parameter (for Create operation)
- O - Optional input parameter (for Create operation)
- R - Response parameter (should not be supplied in Create operation)

Property	Type		Description
`id`	*string*	R	Payment request ID
`payeePaymentReference`	*string*	O	Payment reference of the payee, which is the merchant that receives the payment. This reference could be order id or similar.
`paymentReference`	*string*	R	Payment reference, from the bank, of the payment that occurred based on the Payment request. Only available if status is PAID.
`callbackUrl`	*string*	M	URL that Swish will use to notify caller about the outcome of the Payment request. The URL has to use HTTPS.
`payerAlias`	*string*	O	The registered Cell phone number of the person that makes the payment. It can only contain numbers and has to be at least 8 and at most 15 numbers. It also needs to match the following format in order to be found in Swish: countrycode + cell phone number (without leading zero). E.g.: 46712345678
`payeeAlias`	*string*	M	The Swish number of the payee
`amount`	*string*	M	The amount of money to pay. The amount cannot be less than 1 SEK and not more than 999999999999.99 SEK. Valid value have to be all numbers or with 2 digit decimal seperated with a period.
`currency`	*string*	M	The currency to use. Only supported value currently is SEK.
`message`	*string*	O	Merchant supplied message about the payment/order. Max 50 chars. Allowed characters are the letters a-ö, A-Ö, the numbers 0-9 and the special characters :;.,?!()”.
`status`	*string*	R	The status of the transaction. Possible values: CREATED, PAID, DECLINED, ERROR.
`dateCreated`	*string*	R	The time and date that the payment request was created.

Property	Type		Description
`datePaid`	*string*	R	The time and date that the payment request was paid. Only applicable if status was PAID.
`errorCode`	*string*	R	A code indicating what type of error occurred. Only applicable if status is ERROR.
`errorMessage`	*string*	R	A descriptive error message (in English) indicating what type of error occurred. Only applicable if status is ERROR
`additionalInformation`	*string*	R	Additional information about the error. Only applicable if status is ERROR.

Potential Error codes values:

Code	Description
ACMT03	Payer not Enrolled
ACMT01	Counterpart is not activated
ACMT07	Payee not Enrolled
RF07	Transaction declined
BANKIDCL	Payer cancelled bankid signing
FF10	Bank system processing error
TM01	Swish timed out before the payment was started
DS24	Swish timed out waiting for an answer from the banks after payment was started. Note: If this happens Swish has no knowledge of whether the payment was successful or not. The merchant should inform its consumer about this and recommend them to check with their bank about the status of this payment.

11.3.2 Refund Object

The Refund Object is used in all 3 Refund operations (Create, Retrieve and Callback). The fields that are mandatory are for the Create operation, but of course those fields will also be available on the other operations.

Legend:

- M - Mandatory input parameter (for Create operation)
- O - Optional input parameter (for Create operation)
- R - Response parameter (should not be supplied in Create operation)

Property	Type		Description
`id`	*string*	R	Refund ID
`payerPaymentReference`	*string*	O	Payment reference supplied by the merchant. This could be order id or similar.
`originalPaymentReference`	*string*	M	Reference of the original payment that this refund is for.
`paymentReference`	*string*	R	Reference of the refund payment that occurred based on the created refund. Only available if status is PAID.
`callbackUrl`	*string*	M	URL that Swish will use to notify caller about the outcome of the Refund. The URL has to use HTTPS.
`payerAlias`	*string*	M	The Swish number of the merchant that makes the refund payment.
`payeeAlias`	*string*	R	The Cell phone number of the person that receives the refund payment.
`amount`	*string*	M	The amount of money to refund. The amount cannot be less than 1 SEK and not more than 999999999999.99 SEK. Moreover, the amount can not exceed the remaining amount of the original payment that the refund is for.
`currency`	*string*	M	The currency to use. Only supported value currently is SEK.
`message`	*string*	O	Merchant supplied message about the refund. Max 50 chars. Allowed characters are the letters a-ö, A-Ö, the numbers 0-9 and the special characters :;.,?!()”.
`status`	*string*	R	The status of the refund transaction. Possible values: CREATED, DEBITED, PAID, ERROR.
`dateCreated`	*string*	R	The time and date that the payment refund was created.
`datePaid`	*string*	R	The time and date that the payment refund was paid.

Property	Type		Description
`errorCode`	*string*	R	A code indicating what type of error occurred. Only applicable if status is ERROR.
`errorMessage`	*string*	R	A descriptive error message (in English) indicating what type of error occurred. Only applicable if status is ERROR
`additionalInformation`	*string*	R	Additional information about the error. Only applicable if status is ERROR.

Potential Error codes values:

Code	Description
ACMT07	Payee not Enrolled
ACMT01	Counterpart is not activated
RF07	Transaction declined
FF10	Bank system processing error
DS24	Swish timed out waiting for an answer from the banks after payment was started. Note: If this happens Swish has no knowledge of whether the payment was successful or not. The merchant should inform its consumer about this and recommend them to check with their bank about the status of this payment.

11.3.3 Error Object

Property	Type	Description
`errorCode`	*string*	A code indicating what type of error occurred.
`errorMessage`	*string*	A descriptive error message (in English) indicating what type of error occurred.
`additionalInformation`	*string*	Additional information about the error.

Example: Array of Error objects:

```
[{
    "errorCode": "PA02",
    "errorMessage": "Amount value is missing or not a valid number",
    "additionalInformation": ""
},{
    "errorCode": "AM03",
    "errorMessage": "Invalid or missing Currency",
    "additionalInformation": ""
},{
    "errorCode": "RF08",
    "errorMessage": "Amount value is too large or amount exceeds the amount
of the original payment minus any previous refunds",
    "additionalInformation": "100.00"
}]
```