|  |
| --- |
| Payer API |
| Payer API and programming guidelines |
| This document describes Payer Financial Services payment gateway both technically (API:s) and the concept of secure payments with help of some pictures and descriptive schemes of the flow. |

[Välj datum]

Payer Financial Services AB – manual v 0.95 2015-04-07

# Table of Contents

[Table of Contents 1](#_Toc416203293)

[Payer payment gateway 3](#_Toc416203294)

[Selecting implementation type: Module or General 3](#_Toc416203295)

[API, concept and implementation guide 3](#_Toc416203296)

[Payers Adminweb / Payer Administration 3](#_Toc416203297)

[The PostAPI 3](#_Toc416203298)

[A payment flowchart using the PostAPI 3](#_Toc416203299)

[A payment made in steps 4](#_Toc416203300)

[Callbacks – the concept behind 6](#_Toc416203301)

[Parameters 6](#_Toc416203302)

[Merchant\_reference\_id 6](#_Toc416203303)

[Security: 6](#_Toc416203304)

[Purchase security: 6](#_Toc416203305)

[Callback security: 6](#_Toc416203306)

[Handling special cases 7](#_Toc416203307)

[Authorize only 7](#_Toc416203308)

[Recurring debit 7](#_Toc416203309)

[The form post toward payment gateway 8](#_Toc416203310)

[The form itself 8](#_Toc416203311)

[XML files and structures 9](#_Toc416203312)

[XML file as an example 9](#_Toc416203313)

[XSD Shema controlling the PostAPI XML structure 10](#_Toc416203314)

[The Web service API 13](#_Toc416203315)

[Payer Recurring Debit 14](#_Toc416203316)

[Introduction 14](#_Toc416203317)

[Two steps 14](#_Toc416203318)

[Storing 14](#_Toc416203319)

[Debit 14](#_Toc416203320)

[Web Service (SOAP) 15](#_Toc416203321)

[Create account for Web Service 15](#_Toc416203322)

[Function debitByReference 15](#_Toc416203323)

[Function commitTransaction 16](#_Toc416203324)

[Flowcharts 17](#_Toc416203325)

[Payment steps 17](#_Toc416203326)

[System schema 17](#_Toc416203327)

# Payer payment gateway

## Selecting implementation type: Module or General

Payer recommends merchants to select a readymade Module from the directory structure startpaket/Modules. If you plan to integrate towards Payer without a module – this documentation is describing the general implementation.

## API, concept and implementation guide

Payer has one major way of performing payment processing and it’s called the PostAPI. The PostAPI concept is built upon basic HTTP and WEB centric technologies and protocols such as http, https, form post, base64 and md5.

There is also some major functions and backing/supporting functions delivered as Web Services (SOAP + JSON API:s) The first part of this document is about the PostAPI.

## Payers Adminweb / Payer Administration

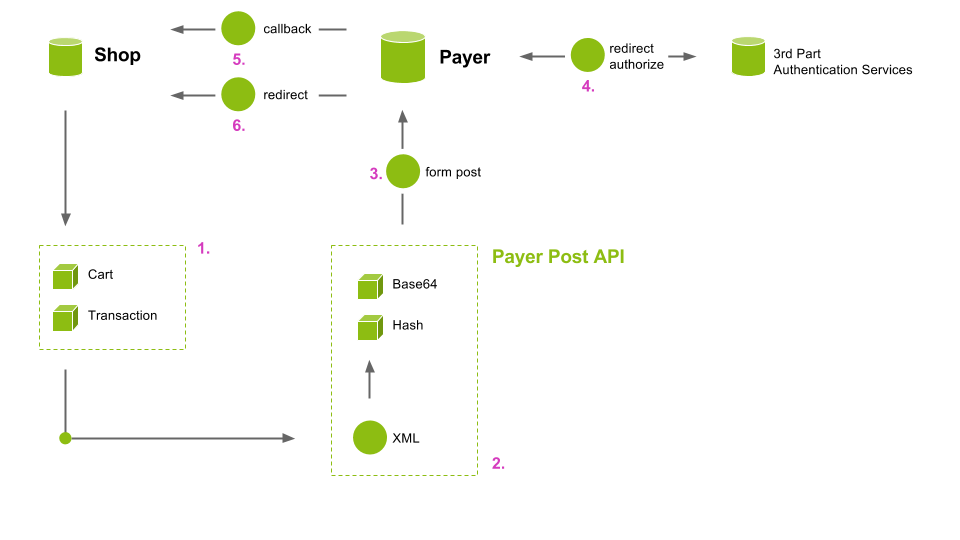
Merchants have access to an administrator web where merchants can configure the system, read reports and handle transactions and refunds. It is called “ Payer adminweb” and is located at this website URL:

<https://secure.payer.se/adminweb>

# The PostAPI

Overview about the PostAPI and its function, API and flow.

## A payment flowchart using the PostAPI



Figur 1

## A payment made in steps

1. A customer (buyer) is finalizing the buying process and wants to pay.
2. The information about the customer and the customers purchase order is serialized into a XML structure. This structure is defined in an XSD document. Added is also information ([URL:s](file:///C:\Users\bihla\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\DSK4ZFDE\s)) for where to go if purchase is cancelled and if successful. Also provided is information of the callbacks. The callback URL:s are responsible for transmitting a signal when a successful purchase has been made.

The XML structure is converted to a BASE64 data string.

A MD5 checksum is calculated with the help of two pre-shared keys (Key1+Key2)

1. The BASE64 and the MD5 –checksum is put into a standard HTML form with the “action” element pointing to Payers payment gateway.

To facilitate the serialization of XML data, encoding and signing and posting (step 2), Payer have created a API called PostAPI. The API is provided for all popular programming languages but is not needed. One can build XML, encode and sign “manually”. The base for the payment is a payment description in pure XML.

A button (or auto) will launch the “form post” towards the Payer payment gateway and payer will display the information again to verify and select payment options. The shop may invoke payment as a single payment option at a time – or as a menu where buyer might select any of the given options.

1. Depending on purchase type [card, invoice, bank, sms, installment or wywallet] the payment information is handed over to partners (or internally) to validate the buyer and purchase details (this differs depending on payment type). Some payment types need buyer to be redirected for external validation (internet bank or 3D- secure validation) and other will utilize an iFrame port for this purpose. In case of authentication/payment detour – the user will be returned back to Payers payment gateway to finalize paying process.
2. If payment is successful – a callback is performed. When handling card payments a “authorize” callback is performed before the settlement callback. All payments generate a “settle” callback when purchase is finalized. The callback is responsible for updating the “pending” order in web shop to “paid” or similar. It is possible to leave card transactions in an “authorized” but “unsettled” state – and to settle transactions at a later stage (at delivery time or similar).

See also: callback security, authorize only

1. The buyer is redirected back to the shops “thank you” page or similar. The thank you page should look for the order status and if order is still pending, buyer should be encouraged to try the payment one more time. Normally the order is already marked “paid” by the systems callback mechanism.

## Callbacks – the concept behind

The callbacks in Payer payment gateway are a fundamental part of the design. The callbacks are responsible for the delivery of the status of the purchase back to the merchants shop. All payment methods have a settle callback – some have a authorize callback as well.

If a callback is directed to a URL simple url like this:

<http://shop.shopper.com/payment/settle>?order\_id=334455

The actual callback could look like this:

<http://shop.shopper.com/payment/settle?order_id=334455&payer_testmode=false&payer_payment_type=invoice&payer_callback_type=settle&payer_added_fee=40&payer_payment_id=IFO@XXX60o5c72fii616hloi83n&payread_payment_id=IFO@XXX60o5c72fii616hloi83n&md5sum=28E69FA28EE10A2C40E8E30468FE3E68>

### Parameters

As you can see – payer are adding a lot of metadata information:

order\_id=334455 Original parameter added by merchants shop.

payer\_testmode=true [true|false] – indicates test or live mode

payer\_payment\_type=invoice [invoice|card|sms|wywallet|bank|enter] – payment type

payer\_callback\_type=settle [authorize|settle|store] – callback type

payer\_added\_fee=40 [when payer adds the fee for a specific payment type] - fee

payer\_payment\_id=xxyy [xxx@yyyyy – reference: max 64 characters long] - id

md5sum=28…68 [checksum signing the callback] - checksum

the md5sum is calculated by formula MD5(key1+callbackURL+key2) – the plus signs are not included in the calculation. The key1 and key2 are also referred to as KeyA and KeyB and can be found in the starting package or in Payers web for merchants.

The target URL at the merchants shop should check both the IP address and the md5sum to ensure a valid payment is behind this call. To fail to check this – the security is jeopardized. By checking both IP and checksum gives a strong protection against forged callbacks. See more under Security / callback.

### Merchant\_reference\_id

In the example above the shop is adding their own referece as **order\_id=334455.** There is also more options here. You can use the set\_reference\_id() function. If set\_reference\_id() function is used you will get one more parameter added to the list above: **payer\_merchant\_reference\_id**

## Security:

### Purchase security:

The purchase can only be initiated if the purchase is signed with the two pre-shared keys – key1 and key2

### Callback security:

The callbacks are responsible for updating the order status. This is a security sensitive part of the process. If this request could be forged – a fake payment could be created. The callbacks are protected by two effective security features.

* A application level firewall – accepting callbacks only from trusted IP-addresses : 79.136.103.5,94.140.57.180,94.140.57.181,94.140.57.184
* A checksum validation feature ensures the callback is properly signed buy a trusted partner. The signing is used by pre-shared keys – key1 and key2

The callback [URL:s](file:///C:\Users\bihla\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\DSK4ZFDE\s) are just standard [URL:s](file:///C:\Users\bihla\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\DSK4ZFDE\s) responsible for signaling back to the merchants e-shop solution.

The very simplest form would look something like this (settle callback):

<http://theshop.com/callback/settle?order_id=12345>

In the example given here above the idea is to trigger a settle script on the merchants shop. This script might send a “thank you” letter to buyer and also to mark the order 12345 as paid. You might also trigger events for reducing “in stock counter” or similar.

Note: The callback is made from the web server (server-to-server) and is not a part of the clients session. You better rely on a pure database update then to trust and load a session even if it is possible to do so – but no cookies are sent and session handling is not performed automatically. On some webservers it is possible however to load a session scope if session-ID is known and distributed in callback.

# Handling special cases

## Authorize only

When charging credit cards or bank cards – it’s standard in some businesses to not charge the card when purchased – but to charge when delivered. Payer can handle this flow by activating the auth\_only mode. You can do it per purchase through the PostAPI – or by simply turn that mode on in the merchants interface.

To activate ‘auth\_only’ mode you set the auth\_only=true in the Options field.

You can set it in several ways (PostAPI version .v29 or later):

use the API: set\_auth\_only(true);

Prior to version .v29 you can do:

use the API: set\_options(‘auth\_only=true’);

Raw XML:

<options>auth\_only=true</options>

If system is in *authorize-only* mode – the purchase ends right after authorize callback stage and user is redirected back to shop as usual – BUT the money is just reserved – not settled. This means that the settle callback is not executed – only authorize callback is (and parameter payer\_settle\_id is added) – and the system is left in this way until merchant clicks button in Payer’s adminweb administration or merchants uses the webservice API to call function **commitTransaction**.

To make a call to commitTransaction – you need to initialize the web service framework as described in chapter Web Service.

## Recurring debit

Recurring debit is the name of a service making it possible to store a credit card once for all and make debit against that card at any given time for any given amount.

To be able to perform recurring debits you need to have a special agreement with the bank so they can allow you to perform periodic billing. The bank will also provide you with a legal contract that will instruct you as a merchant want you can and cannot do. Technically – there is not much restrictions but legally (by contract with bank) there is some limitations you must learn.

When using recurring debit – the credit card information is stored and tokenized (changed) from the PAN (card number) to a more neutral token. This token has the limitation that it can only charge towards the registering merchants site and cannot be used for paying somewhere else. This is a good security mechanism. See chapter “recurring debit” for more information.

# The form post toward payment gateway

## The form itself

Refer: Figur 1 – step 3

<form action="https://secure.payer.se/PostAPI\_V1/InitPayFlow" method="post">

<input type="hidden" name="**payer\_agentid**" value="XYZZY">

<input type="hidden" name="**payer\_xml\_writer**" value="payer\_php\_0\_2\_v27">

<input type="hidden" name="**payer\_data**" value="PD94b ... BASE64 ... FwaV8wXzI+Cg==">

<input type="hidden" name="**payer\_charset**" value="UTF-8">

<input type="hidden" name="**payer\_checksum**" value="28f12c9218ecaa017e064987d442d1e2">

<input type="submit" value="Click to proceed to payment">

</form>

The form fields are as you can see above: payer\_agentid, payer\_xml\_writer, payer\_data, payer\_charset and payer\_checksum

The **payer\_agentid** should contain the by payer given merchantid (agent id).

The **payer\_xml\_writer** should contain the verison of the Payer PostAPI used by client.

The **payer\_data** consists of the Base64 encoded XML data.

The **payer\_charset** should contain the charset used on the client side. This is usually one of ISO-8851-9 or UTF-8. If unsure you can see this by visiting your site and enter your shop (e-commerce webshop) and check in the browser encoding section to see how your browser has inerpreted the page. You can also refer to your webserver configuration files or webshop readme.txt

The **payer\_checksum** is calculated as MD5(key1 + payer\_data + key2). The key1 and key2 are pre-shared secret keys that should be kept in a way that only webserver user (or root) can read them.

The Key1 and Key2 and agentID:s can be found in the starting package.

Check file startpaket/API/PHP/PayReadConf.php

# XML files and structures

## XML file as an example

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

<payread\_post\_api\_0\_2 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="payread\_post\_api\_0\_2.xsd">

<seller\_details>

<agent\_id>XYZZY</agent\_id>

</seller\_details>

<buyer\_details> <first\_name></first\_name>

<last\_name></last\_name>

<address\_line\_1></address\_line\_1>

<address\_line\_2></address\_line\_2>

<postal\_code></postal\_code>

<city></city>

<country\_code></country\_code>

<phone\_home></phone\_home>

<phone\_work></phone\_work>

<phone\_mobile></phone\_mobile>

<email>anders@and.se</email>

<organisation>Svenska klubben för sakernas ting</organisation>

<orgnr>999999-9991</orgnr>

</buyer\_details>

<purchase>

<currency>SEK</currency>

<description>En fantastisk order #102313</description>

<purchase\_list>

<freeform\_purchase>

<line\_number>1</line\_number>

<description>Förbetalda frimärken</description>

<price\_including\_vat>12.50</price\_including\_vat>

<vat\_percentage>25</vat\_percentage>

<quantity>20</quantity>

</freeform\_purchase>

</purchase\_list>

</purchase>

<processing\_control>

<success\_redirect\_url>https://app.com/payment/payersuccess</success\_redirect\_url>

<authorize\_notification\_url>https://app.com/payment/payerauthorize?referenceId=b788-49d2-b98f-77d83a5517a3</authorize\_notification\_url>

<settle\_notification\_url>https://app.com/payment/payersettle?referenceId=b788-49d2-b98f-77d83a5517a3</settle\_notification\_url>

<redirect\_back\_to\_shop\_url>https://app.com/a/account/buy</redirect\_back\_to\_shop\_url>

</processing\_control>

<database\_overrides>

<accepted\_payment\_methods>

<payment\_method>card</payment\_method>

</accepted\_payment\_methods>

<debug\_mode>false</debug\_mode>

<test\_mode>false</test\_mode>

<language>sv</language>

</database\_overrides>

</payread\_post\_api\_0\_2>

## XSD Shema controlling the PostAPI XML structure

<?xml version=*"1.0"* encoding=*"ISO-8859-1"*?>

<xs:schema xmlns:xs=*"http://www.w3.org/2001/XMLSchema"*>

<xs:element name=*"payread\_post\_api\_0\_2"*>

<xs:complexType>

<xs:sequence>

<xs:element name=*"seller\_details"*>

<xs:complexType>

<xs:all>

<xs:element name=*"agent\_id"* type=*"xs:string"*/>

<!-- here we can send client information / webshop used etc -->

<xs:element name=*"client\_version"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

</xs:all>

</xs:complexType>

</xs:element>

<xs:element name=*"buyer\_details"*>

<xs:complexType>

<xs:all>

<xs:element name=*"first\_name"* type=*"xs:string"*/>

<xs:element name=*"last\_name"* type=*"xs:string"*/>

<xs:element name=*"address\_line\_1"* type=*"xs:string"*/>

<xs:element name=*"address\_line\_2"* type=*"xs:string"*/>

<xs:element name=*"postal\_code"* type=*"xs:string"*/>

<xs:element name=*"city"* type=*"xs:string"*/>

<xs:element name=*"country\_code"* type=*"countryCode"*/>

<xs:element name=*"phone\_home"* type=*"xs:string"*/>

<xs:element name=*"phone\_work"* type=*"xs:string"*/>

<xs:element name=*"phone\_mobile"* type=*"xs:string"*/>

<xs:element name=*"email"* type=*"xs:string"*/>

<!-- this is the company name -->

<xs:element name=*"organisation"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<!-- use when an organisation/company is the buyer -->

<xs:element name=*"your\_reference"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<!-- this is the personnummer/organisationsnummer -->

<xs:element name=*"orgnr"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<!-- Any optional parameters with comma seperated -->

<xs:element name=*"options"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<!-- customer\_id is the webshops own identity on the user -->

<xs:element name=*"customer\_id"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

</xs:all>

</xs:complexType>

</xs:element>

<xs:element name=*"purchase"*>

<xs:complexType>

<xs:all>

<xs:element name=*"reference\_id"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"description"* type=*"descriptionLong"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"message"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"hide\_details"* type=*"xs:boolean"* default=*"false"*/>

<xs:element name=*"currency"* type=*"xs:string"*/>

<xs:element name=*"purchase\_list"*>

<xs:complexType>

<xs:sequence>

<xs:element name=*"catalog\_purchase"* minOccurs=*"0"* maxOccurs=*"unbounded"*>

<xs:complexType>

<xs:all>

<xs:element name=*"line\_number"* type=*"xs:integer"*/>

<xs:element name=*"id"* type=*"xs:string"*/>

<xs:element name=*"quantity"* type=*"quantity"*/>

</xs:all>

</xs:complexType>

</xs:element>

<xs:element name=*"freeform\_purchase"* minOccurs=*"0"* maxOccurs=*"unbounded"*>

<xs:complexType>

<xs:all>

<xs:element name=*"line\_number"* type=*"xs:integer"*/>

<xs:element name=*"description"* type=*"xs:string"*/>

<xs:element name=*"price\_including\_vat"* type=*"price"*/>

<xs:element name=*"vat\_percentage"* type=*"percentage"*/>

<xs:element name=*"quantity"* type=*"quantity"*/>

<xs:element name=*"item\_number"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"unit"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"account"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"agent\_id"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

</xs:all>

</xs:complexType>

</xs:element>

<xs:element name=*"subscription\_purchase"* minOccurs=*"0"* maxOccurs=*"unbounded"*>

<xs:complexType>

<xs:all>

<xs:element name=*"line\_number"* type=*"xs:integer"*/>

<xs:element name=*"description"* type=*"xs:string"*/>

<xs:element name=*"price\_including\_vat"* type=*"price"*/>

<xs:element name=*"vat\_percentage"* type=*"percentage"*/>

<xs:element name=*"quantity"* type=*"quantity"*/>

<xs:element name=*"item\_number"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"unit"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"account"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"*/>

<xs:element name=*"recurring\_price\_including\_vat"* type=*"price"*/>

<xs:element name=*"start\_date"* type=*"xs:date"*/>

<xs:element name=*"stop\_date"* type=*"xs:date"*/>

<xs:element name=*"count"* type=*"xs:integer"*/>

<xs:element name=*"periodicity"* type=*"xs:string"*/>

<xs:element name=*"cancel\_days"* type=*"xs:integer"*/>

</xs:all>

</xs:complexType>

</xs:element>

<xs:element name=*"info\_line"* minOccurs=*"0"* maxOccurs=*"unbounded"*>

<xs:complexType>

<xs:all>

<xs:element name=*"line\_number"* type=*"xs:integer"*/>

<xs:element name=*"text"* type=*"xs:string"*/>

</xs:all>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:all>

</xs:complexType>

</xs:element>

<xs:element name=*"processing\_control"*>

<xs:complexType>

<xs:all>

<xs:element name=*"success\_redirect\_url"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"* />

<xs:element name=*"error\_redirect\_url"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"1"* />

<xs:element name=*"authorize\_notification\_url"* type=*"xs:string"*/>

<xs:element name=*"settle\_notification\_url"* type=*"xs:string"*/>

<xs:element name=*"redirect\_back\_to\_shop\_url"* type=*"xs:string"*/>

</xs:all>

</xs:complexType>

</xs:element>

<xs:element name=*"database\_overrides"*>

<xs:complexType>

<xs:sequence>

<xs:element name=*"accepted\_payment\_methods"*>

<xs:complexType>

<xs:sequence>

<xs:element name=*"payment\_method"* type=*"xs:string"* minOccurs=*"1"* maxOccurs=*"unbounded"*/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name=*"debug\_mode"* type=*"debugMode"* default=*"silent"*/>

<xs:element name=*"test\_mode"* type=*"xs:boolean"* default=*"false"*/>

<xs:element name=*"language"* type=*"countryCode"* default=*"se"*/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:simpleType name=*"price"*>

<xs:restriction base=*"xs:decimal"*>

<xs:fractionDigits value=*"100"*/>

<!-- <xs:minInclusive value="0"/>-->

</xs:restriction>

</xs:simpleType>

<xs:simpleType name=*"quantity"*>

<xs:restriction base=*"xs:integer"*>

<xs:minInclusive value=*"-10000"*/>

<xs:maxInclusive value=*"10000"*/>

</xs:restriction>

</xs:simpleType>

<xs:simpleType name=*"percentage"*>

<xs:restriction base=*"xs:decimal"*>

<xs:fractionDigits value=*"2"*/>

<xs:minInclusive value=*"0"*/>

<xs:maxInclusive value=*"100"*/>

</xs:restriction>

</xs:simpleType>

<xs:simpleType name=*"countryCode"*>

<xs:restriction base=*"xs:string"*>

<xs:length value=*"2"*/>

</xs:restriction>

</xs:simpleType>

<xs:simpleType name=*"descriptionLong"*>

<xs:restriction base=*"xs:string"*>

<xs:length value=*"254"*/>

</xs:restriction>

</xs:simpleType>

<xs:simpleType name=*"debugMode"*>

<xs:restriction base=*"xs:string"*>

<xs:enumeration value=*"silent"*/>

<xs:enumeration value=*"brief"*/>

<xs:enumeration value=*"verbose"*/>

</xs:restriction>

</xs:simpleType>

</xs:schema>

# The Web service API

At the core of payment processing we have API:s written as SOAP Web Services to deliver industry standard, high security and server-to-server processing of payments. Earlier Payer relied heavily on this interface. Around 2006 the move towards PCI-DSS and 3D-Secure made the SOAP/Web Service interface more of a choice for merchants not using credit cards. We now use web services mostly for checking on transaction status or refunding transactions. We also use web service to trigger “recurring debits” – a very popular service.

The web service is split up in two different services:

* The new – simplified and supported WSDL:

<https://secure.payer.se/PostAPI_V1/services/PublicPayerCore?wsdl>

Note: As you see, the PostAPI\_V1 is a part of the Web Service interface URL and is a part of the PostAPI package. However – most of the Web Service calls in the PublicPayerCore package are separate from the PostAPI and can be used independently.

* The original - legacy (deprecated but still operational) WSDL:

<https://secure.payer.se/soap/PaymentGateway_V2.wsdl>

Note: This web service is Payers original and old style interface. Use is discouraged. A manual for this is provided in: startpaket/WebService/documentation/WebServiceAPI/WS\_2.0.xx.html

# Payer Recurring Debit

## Introduction

Recurring debit by Payer makes it possible to store a credit card number (primary account number = PAN) once for all. The process of storing is handled mainly at the Payer server to avoid the need for the Merchant to be fully PCI-compliant. In the storing process, Payer is storing the PAN and returns a “neutral” Unique key (token) that can be used instead of the PAN. Now the Merchant can “bill” the Buyer at any given time – with any given amount (this is regulated in the agreement between the Merchant and the Bank giving the Card Agreement as Recurring Debit)

## Two steps

* The first step is to store the card number (PAN) and retrieve the Unique ID (Storing).
* The second step is performing a “debit”, to actually debit the user’s card by using the Unique ID as reference (Debit).

## Storing

Storing the card number (PAN) is performed by using the standard PostAPI interface that Payer is providing for normal purchases. The only difference is that we set the “options” field in BuyerEntryData with the keyword “store =true”. Now we will perform a store and the Unique ID referring to the stored card will be returned through the “auth-callback” in an variable called “payer\_unique\_id”. Settle-callback is not used when performing a store, but must be filled with a URL. You can put a dummy one in.

## Debit

To debit the card you need to use Payers *Web Service* interface. You do that by using the standard procedure for your language/framework. In PHP 5 or later you can use the built-in SOAP support.

In JAVA you would probably use axis/axis2 toolkit to build your code stubs. Examples will be provided for the most popular languages/frameworks out there. C# users can do this straight from the Visual Studio IDE by clicking “add Web Service client” and enter the WSDL URL.

# Web Service (SOAP)

## Create account for Web Service

To be able to use the API:s you need an account. In all the examples, we need to use the API createSession or createSessionEx call. In an production environment, we recommend to use the API CreateSessionEx instead and to have a separate account for the Web service session - this will avoid any problems associated with the fact that accounts might be locked out if misused or when login have failed more than 5 times.

To create a Web service API- account – log in into Payers “adminweb” at <https://secure.payer.se/adminweb> and create one under “Inställningar” -> “Konto” -> “Lägg till API-user”. When clicking on button – please remember the password/commands displayed – it will not be stored or viewed anywhere else. If password is lost – you need to remove and add the user again.

## Function debitByReference

The URL for the WSDL file is:   
<https://secure.payer.se/PostAPI_V1/services/PublicPayerCore?wsdl>

To perform a debit, you must use the Web Service API method called ”debitByReference”. You must also have a valid “SessionId”. The SessionId is used to check that the Merchant has a legal account at Payer and that Merchant is successfully logged in.

Example: to bill 49 SEK for a product called “Weekly service”:

1. sessionID = createSession (agentID, password); // see Create account
2. transID = debitByReference(sessionId, uniqSessionId, merchantReferenceId, 49, “SEK”, “Weekly Service”, 25);
3. destroySession(sessionId) ; // or perform more operations

The transID above can be used if you want to perform a refund (give the money back to the Buyer)

In the example above, at “step 1”, we use the API CreateSession call. In an production environment, we recommend to use the API CreateSessionEx instead and to have a separate account for the Web service session - this will avoid any problems associated with the fact that accounts might be locked out if misused or when login have failed more than 5 times.

To create a Web service API- account – log in into Payers “adminweb” at <https://secure.payer.se/adminweb> and create one under “Inställningar” -> “Konto” -> “Lägg till API-user”. When clicking on button – please remember the password/commands displayed – it will not be stored or viewed anywhere else. If password is lost – you need to remove and add the user again.

## Function commitTransaction

The Web Service API method called “commitTransaction” can be used to settle (commit) authorized only trransactions.

Example: To settle transaction with payer\_settle\_id = 33445566

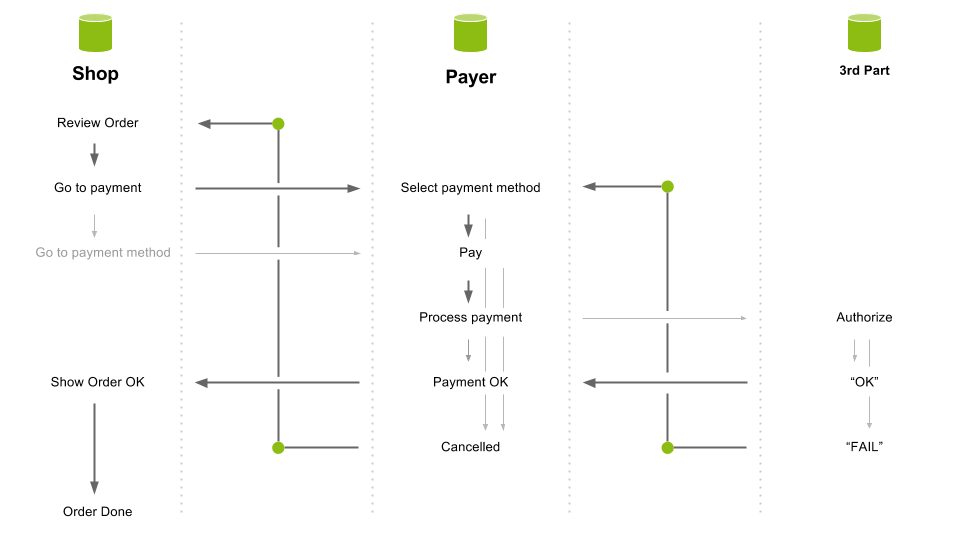
Example: to bill 49 SEK for a product called “Weekly service”:

1. sessionID = createSession (agentID, password); // see Create account
2. transID = commitTransaction(sessionId, 33445566);
3. destroySession(sessionId) ; // or perform more operations

During this call – the system will perform a settlement callback to the shop.

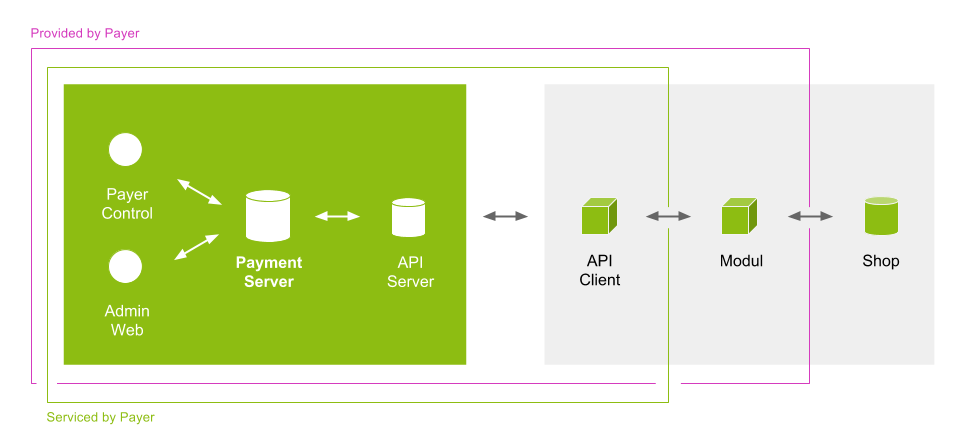
# Flowcharts

## Payment steps



Figur 2

## System schema



Figur 3