

SecurePayTech.com Developer Information

securepaytech.com

■ SECURE PAYMENT TECHNOLOGIES

Updated: 8 Aug 2011

Introduction

This document is intended to help you to integrate your website or system(s) with SecurePayTech's online credit card processing facilities.

Architecture

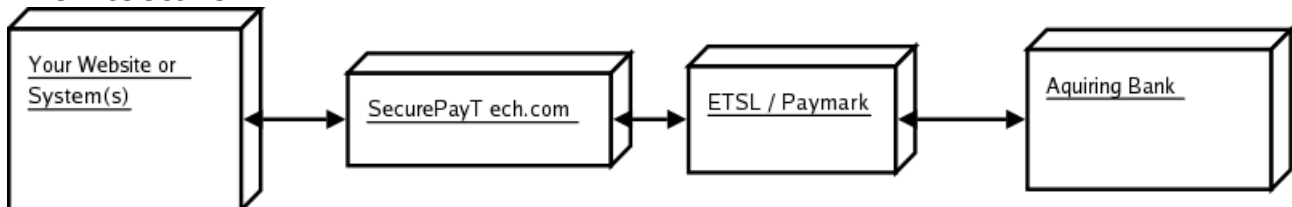


Figure 1: Overview of the SecurePayTech architecture

Figure 1 above, shows where SecurePayTech fits into the overall system of processing credit card transactions on-line. The typical flow of events is:

- Your system makes a request for a transaction to the SecurePayTech system
- The SecurePayTech system authenticates your request, and checks to see if it is a valid one.
- SecurePayTech sends the request to the acquiring bank, via the ETSL / Paymark network.
- The bank validates the credit card, and performs the transfer of funds into the acquiring merchant bank account. The results are then sent back to SecurePayTech via the ETSL / Paymark network.
- SecurePayTech processes these results, and returns the relevant result messages back to your system
- Your system deals with the returned results in the specific manner appropriate to your application.

Purchase-mode or Authorisation/Capture?

There are presently two different types of merchant account that can be used through SecurePayTech.com: Purchase-Mode, or Authorisation/Capture-mode.

- In Purchase-mode (the most common mode), when a purchase transaction is made through SecurePayTech, the card is authorised (i.e. the card is checked that it is valid, and has sufficient funds), and the money is debited immediately.
- In Authorisation/Capture-mode, the initial transaction only authorises the card, and “reserves” the requested amount of funds on the card, without debiting it. If no further action is taken, that “reserve” will be released after a certain period of time, determined by the bank.
- Once a transaction has been authorised in this way, further “capture” transactions can be issued against the authorisation, in order to debit funds from the card. Multiple captures can be issued against the authorisation, but the combined value of these cannot exceed the value of the authorisation.

Note: If you require Authorisation/Capture processing, it is important that you request it when setting up the merchant account through your bank.

SecurePayTech interfaces

There are currently two interfaces available for performing purchase transactions with SecurePayTech:

1. a SOAP (Simple Object Access Protocol) interface
2. an HTTPS POST interface, which may be of use should SOAP be inappropriate

Not all features of SecurePayTech are available through all interfaces due to lack of demand, which may be fixed if sufficient demand arises.

Interface	Purchase	Refund	Auth/Capture	Batch
SOAP	✓	✓	✓	✓
HTTPS POST	✓	✓	✗	✗

Table 1: SecurePayTech supported features by interface

Testing Accounts

In order to perform test transactions through the SecurePayTech beta testing system, you will need to be assigned a test “Merchant ID” code, and a “Merchant Transaction Key” Transactions performed by one of these test accounts behave slightly differently from a full, production account, in that the results returned from it are dependent on the monetary amount of the transaction.

Until you are assigned a VPS Merchant ID, you can use the following details to run test transactions through the system. Note that you will be unable to view these transactions through your login on the web interface. Once you have been assigned a VPS Merchant ID, and you start using that, then the web interface will show those transactions.

Purchase-mode Account:

Temporary Test VPS Merchant ID: **TESTDIGISPL1**

Temporary Test Passkey: **d557591484cb2cd12bba445aba420d2c69cd6a88**

Authorisation/Capture-mode Account:

Temporary Test VPS Merchant ID: **TESTDIGISPL3**

Temporary Test Passkey: **85d62c3b7b2de3c81f93b255b6c00a86dca5487d**

Testing Data

card type	card number	expiry date (MM/YY)
Visa	4987-6543-2109-8769	05/13
Mastercard	5123-4567-8901-2346	05/13
American Express	3456-7890-1234-564	05/13
Diners Club	3012-345678-9019	05/13

Table 2: fictional card numbers for testing

CSC result code	meaning	testing CSC code
S	CSC not provided	
M	CSC matched	100
P	CSC not processed	102
U	CSC unsupported by card issuer	103
N	CSC not matched	104
Unsupported	CSC unsupported by merchant bank	

Table 3: testing CSC codes and their effect on the transaction

code	card type
0	Unspecified
1	Visa
2	Mastercard
3	American Express
4	Diners Club

Table 4: card types

code	meaning	description
1	Transaction OK	
2	Insufficient Funds	
3	Card Expired	
4	Card Declined	Card is declined for a reason aside from insufficient funds or an expired card.
5	Server Error Occurred	A software error occurred either at SecurePayTech or at one of our provider's systems.
6	Communications Error	This usually indicates that a timeout has occurred while processing the transaction request.
7	Unsupported Transaction Type	
8	Bad or Malformed Request	This error will return a reason in the 'failReason' result field.
9	Invalid Card Number	Card number was incorrect length, or failed verification

Table 5: transaction result codes

The return code received from the test system depends upon the cent value in the transaction amount, for example an amount of \$30.00 returns “Transaction OK”, while \$30.10 returns “Insufficient Funds”.

cent value	response code	description
.00	1	Transaction OK
.10	4	Card Declined
.54	3	Card Expired
.57	7	Unsupported Transaction Type
.75	4	Card Declined
.91	6	Communications Error

Table 6: results depending on cent value while in testing

NOTE: Using any cent value *other than* .00, will result in a transaction error of some kind (usually a Server Error). In a live, production system, any cent value can be used, it will not effect the result in this way.

Explanation of Common Fields / Terms

- **VPS MerchantID:** Unique identifier assigned to you which ties transactions to your merchant bank account.
- **MerchantKey:** 40 character string that authenticates your transaction request with the SecurePayTech system.
- **OrderReference:** A reference number or string that has meaning in your own application that need not be unique. Example: “AcmeWidgetOrder_3424”. **This field should be no longer than 20 characters.** It forms part of the MerchTxnRef response field, which has a 40 character limit. If it needs to, the system will truncate your OrderReference in order to keep the MerchTxnRef within 40 characters.
- **MerchTxnRef:** Unique identifier assigned to each transaction by the SecurePayTech system, and can be used to retrieve information relating to that transaction later on.
- **TransactionNo:** Identifier assigned to the transaction by the ETSL's systems. This identifier can later be used to perform refunds against that transaction.
- **ReceiptNo:** Identifier for the receipt data generated by ETSL. Also known as the RRN.
- **AuthorizeID:** Code issued by the bank tracking the request to approve or deny the transaction
- **BatchNo:** Number identifying which batch job at the bank (when they actually perform funds transfer) that contains the transaction.

Purchase using HTTPS POST

Direct your HTTPS POST to <https://tx.securepaytech.com/web/HttpPostPurchase>. The parameters to this URL are:

- MerchantID
- MerchantKey
- OrderReference
- Amount
- Currency
- CardType
- CardNumber
- CardExpiry
- CardHolderName

..and optionally:

- EnableCSC
- CSC

These POST variables can be sent in any order. The results will be returned in the HTTP response to the POST; as comma-delimited fields (with a mime-type of `text/plain`). `EnableCSC` should be either `true` or `false`.

If the request failed because of “Communications Error”, then just the result string will contain the return code, plus optionally the rest of the receipt data, depending on at which point the communications failure occurred.

If the request is rejected for being incorrect, for example, the `MerchantKey` is incorrect, then it will be the result code, then a string explaining the reason as to why it failed, e.g.:

```
"8,Not all the required fields were filled out, or there was
invalid data in one of the fields"
```

or:

```
"8,Bad passkey"
```

Similarly, if a “Server Error” occurs, then the code, and a `failReason` will be returned, e.g.:

```
"5,Database Error"
```

Otherwise, it will return the fields in this order:

- `resultCode`, `merchTxnRef`, `receiptNo`, `transactionNo`, `authorizationID`,
`batchNo`[, `cscResultCode`]

so a typical (successful) response string would look something like:

```
"1,1-ACMEWIDGETORDER232-1100564515198-281,000000000453,904,000453,20041116"
```

Refund using HTTPS POST

Performing a refund against a previously successful transaction is relatively simple compared to performing that initial transaction. You do not need to store the credit card details (SecurePayTech doesn't store them either) in order to perform a refund in this way.

Direct your HTTPS POST to <https://tx.securepaytech.com/web/HttpPostRefund>. The parameters to this URL are:

- `MerchantID`
- `MerchantKey`
- `Amount`
- `OriginalTxNumber`

These POST variables can be sent in any order. The results will be returned in the HTTP response to the POST; as comma-delimited fields (with a mime-type of `text/plain`).

The original transaction is referenced by the `originalTxNumber` parameter which comes from the `transactionNo` field in the response from the original purchase transaction. Many refunds may be made against the original transaction but the total refunded amount cannot exceed the original amount of the purchase transaction. (Note: this `transactionNo` field is *not* the `merchTxnRef` code).

This method returns the same response string as the HTTPS POST Purchase response string given above.

Purchase using SOAP

Note: Authorisation/Capture merchants can use this action, but it will result in a separate authorisation, followed automatically by a capture for the full transaction amount.

The SOAP interface provides both WSDL and non-WSDL methods of calling the `purchase` RPC method. SecurePayTech uses Apache Axis 1.2 for providing SOAP web-services.

The WSDL for the purchase method on the server is located at:

<https://tx.securepaytech.com/web/SoapPurchase?wsdl> and the endpoint is at:
<https://tx.securepaytech.com/web/SoapPurchase> and provides these methods:

```
purchase( merchantID      : String,
          merchantKey    : String,
          orderReference  : String,
          amount         : float,
          currency       : String,
          cardType       : int,
          cardNumber     : String,
          cardExpiry     : String,
          cardHolderName : String )

purchaseWithCsc( merchantID      : String,
                 merchantKey    : String,
                 orderReference  : String,
                 amount         : float,
                 currency       : String,
                 cardType       : int,
                 cardNumber     : String,
                 cardExpiry     : String,
                 cardHolderName : String,
                 csc            : String )
```

The only available currency at this stage is New Zealand Dollars – this should be set as “NZD”. The `cardType` parameter is an integer representing the credit card type, as listed in Table 2. The `cardNumber` field can contain hyphens or spaces, or a combination of these, as non-numerical characters are stripped out before processing occurs. The `cardExpiry` parameter is in MMY format, so an expiry date of December 2006 would be “1206”.

The `purchase` method returns a SOAP complex type (`TransactionResult`) containing the following elements:

```
TransactionResult {
  resultCode      : int
  failReason     : String
  merchTxnRef    : String
  receiptNo     : String
  transactionNo  : String
  authorizationID : String
  batchNo       : String
}
```

..while the `purchaseWithCsc` method returns a type called `CscTransactionResult`, which is identical to `TransactionResult` but for an additional field: `cscResultCode : String`.

`failReason` is always `null` except for when the result code returns a “Bad Request”, or a “Server Error” code. In this case, it gives an explanation as to why the request failed.

The possible values for `resultCode` are listed in Table 5. The example code on the SecurePayTech.com website provides more information on using the SOAP interface.

Refund

Performing a refund against a previously successful transaction is relatively simple compared to performing that initial transaction. You do not need to store the credit card details (SecurePayTech doesn't store them either) in order to perform a refund in this way.

The WSDL for the refund method on the server is located at:

<https://tx.securepaytech.com/web/SoapRefund?wsdl> and the endpoint is at <https://tx.securepaytech.com/web/SoapRefund> and provides these methods:

```
refund( merchantID      : String,
        merchantKey    : String,
        amount         : float,
        originalTxNumber : String )
```

The original transaction is referenced by the `originalTxNumber` parameter which comes from the `transactionNo` field in the response from the original purchase transaction. Many refunds may be made against the original transaction but the total refunded amount cannot exceed the original amount of the purchase transaction. (Note: this `transactionNo` field is *not* the `merchTxnRef` code).

This method returns the same complex SOAP type (`TransactionResult`) that the `SoapPurchase` web-service returns.

Authorisation and Capture

(Available to Authorisation/Capture merchants only)

The authorisation service is very similar to the purchase service, in that it accepts the same parameters, and return the same kind of result type. The capture service is similar to the refund service, but instead of refunding against the transaction, it captures the funds from it.

The WSDL for the auth and capture methods on the server is located at:

<https://tx.securepaytech.com/web/SoapAuthCapture?wsdl> and the endpoint is at <https://tx.securepaytech.com/web/SoapAuthCapture> and provides these methods:

```
auth( merchantID      : String,
      merchantKey    : String,
      orderReference  : String,
      amount          : float,
      currency        : String,
      cardType        : int,
      cardNumber      : String,
      cardExpiry      : String,
      cardHolderName  : String )

authWithCsc( merchantID      : String,
             merchantKey    : String,
             orderReference  : String,
             amount          : float,
             currency        : String,
             cardType        : int,
             cardNumber      : String,
             cardExpiry      : String,
             cardHolderName  : String,
             csc             : String )

capture( merchantID      : String,
         merchantKey    : String,
         amount          : float,
         originalTxNumber : String )
```

Batch Processing

There are two ways of submitting sets of transactions through our batch processing facility. The first method is to upload a .csv file via the web interface, the other is to submit an XML document directly to our batch SOAP web-service.

The service (WSDL at: <https://tx.securepaytech.com/batch/ProcessBatch?wsdl>) is Document/Literal-style webservice, which accepts an XML formatted document as input, and returns another XML formatted document as a result. As the batch is processed asynchronously The result document contains the batch run ID number, initial result codes, and a list of transactions that failed the initial validation.

Example of the input document:

```
<BatchRequest_1>
  <processBatch xmlns="http://tx.securepaytech.com/types">
    <reportEmail>person@example.com</reportEmail>
    <reportEmail>person2@example.com</reportEmail>
    <merchantID>TESTDIGI</merchantID>
    <txPassKey>123344567788878787</txPassKey>
    <purchase>
      <amount>5.00</amount>
      <currency>NZD</currency>
      <orderReference>ACME WIDGET 42</orderReference>
      <cardHolder>Some Guy</cardHolder>
      <cardNumber>4987654321098769</cardNumber>
      <cardExpiry>0607</cardExpiry>
      <cardType>1</cardType>
    </purchase>
    <purchase>
      <amount>5.00</amount>
      <currency>NZD</currency>
      <orderReference>ACME WIDGET 43</orderReference>
      <cardHolder>Another Guy</cardHolder>
      <cardNumber>4987654321098769</cardNumber>
      <cardExpiry>9902</cardExpiry>
      <cardType>1</cardType>
    </purchase>
  </processBatch>
</BatchRequest_1>
```

Example of the response:

```
<ns1:processBatchResponse xmlns:ns1="http://tx.securepaytech.com/types">
  <result>
    <batchID>86</batchID>
    <error xsi:nil="1"/>
    <errorCode>0</errorCode>
    <invalid>
      <orderReference>ACME WIDGET 43</orderReference>
      <row>2</row>
    </invalid>
    <willProcess>true</willProcess>
  </result>
</ns1:processBatchResponse>
```

code	meaning
0	No Error
1	Empty Request
2	Invalid Merchant
3	Invalid Passkey
4	Server Error
5	All transactions were invalid

Table 7: batch processing error codes

ProcessBatch input document DTD

```

<!ELEMENT BatchRequest_1 (processBatch)>
<!ELEMENT processBatch (reportEmail*,merchantID,txPassKey,purchase*)>
<!ELEMENT reportEmail EMPTY>
<!ELEMENT merchantID EMPTY>
<!ELEMENT txPassKey EMPTY>
<!ELEMENT purchase
(amount,currency,orderReference,cardHolder,cardNumber,cardExpiry,cardType)>
<!ELEMENT amount EMPTY>
<!ELEMENT currency EMPTY>
<!ELEMENT orderReference EMPTY>
<!ELEMENT cardHolder EMPTY>
<!ELEMENT cardNumber EMPTY>
<!ELEMENT cardExpiry EMPTY>
<!ELEMENT cardType EMPTY>

```

ProcessBatch output document DTD

```

<!ELEMENT processBatchResponse (result)>
<!ELEMENT result (batchID,error,errorCode,invalid*,willProcess)>
<!ELEMENT batchID EMPTY>
<!ELEMENT error EMPTY>
<!ELEMENT errorCode EMPTY>
<!ELEMENT invalid (orderReference,row)>
<!ELEMENT orderReference EMPTY>
<!ELEMENT row EMPTY>
<!ELEMENT willProcess EMPTY>

```