StoreFrontÔ Third Party Payment Processor Integration Guide

Purpose

This guide is intended to give third party payment processor integrators a basic understanding of how to integrate payment processing services into StoreFront 5.0. This guide is a general overview of the process and is not intended to be specific to any particular payment processor or method of integration.

General Overview

StoreFront can use integrated third-party payment processing services to handle credit card transactions. A merchant would typically acquire an account with the payment processor and then configure their StoreFront web store to be able to process transactions using that service.

In a StoreFront web store, credit card processing begins in the **verify.asp** page. If a customer chooses to pay for their order using a credit card, this page will load and display input fields to collect the necessary credit card information: name, expiration date, card type, and card number. After this information has been collected, it will be sent to the appropriate payment processing service through a routine stored in the **processor.asp** page. The payment processing service then returns a result – either success or a failure with an error message– which is parsed and interpreted by processor.asp. The confirm.asp page then displays the result of the transaction to the customer. If the transaction failed, the customer will be allowed to resubmit their payment information.

Technical Overview

This section is intended to give an overview of the underlying technology used in StoreFront. This should help developers determine the feasibility of integration and what the integration process will require. Note that knowledge of ASP and VBScript will be prerequisite to performing many of the tasks described in this guide.

The StoreFront web application files are written in the Active Server Pages scripting language, and we recommend that any integration be performed using ASP with VBScript and a COM server component. Using other technologies such as Java, Perl, Shell Scripting, etc. is possible, but may require more work and will increase the likelihood that your custom integration will become incompatible with future releases of StoreFront. Most payment processors will provide a COM server to handle the communication between your web store and their gateway, or they will provide VBScript routines that perform request/response operations between two secure servers running SSL (HTTPS). StoreFront is best suited for these types of approaches to integration. **Note:** A COM server could be written in C++, Java or Visual Basic. This does not affect StoreFront as long as the component is COM compliant and can be instantiated in ASP.

Step 1: Adding the Payment Processing Service to your Database

Before you begin modifying your web store's application files to incorporate new code, you should add the name of the new payment processing service to the list that is stored in your web store's database. This will make the payment processing service selectable from the StoreFront development tools.

Open your database using Access or, for SQL Server, Enterprise Manager or an Access Data Link. Locate and open the sfTransactionMethods table. Create a new record in this table by entering the name of the new payment processing service in the trnsmthdName field. The trnsmthdID field will autopopulate. For the rest of the fields in the table, enter a single space in each. This will eliminate the risk of introducing null values into the database, which could potentially cause problems. Note the value you entered in the trnsmthdName field and the corresponding number in the trnsmthdID field; you will need these for the next step.

Step 2: Modify Confirm.asp
Confirm.asp detects which payment processing servi ce has been selected and then executes the appropriate routines for each. You must add a case to confirm.asp to detect the new payment processing service and perform the appropriate tasks.

1. Open confirm.asp using Notepad or another file editor.

2. Locate the following line:

```
' Process Order
```

Immediately following it will be a large block of conditionals:

```
If sTransMethod = "15" OR sPaymentMethod = "PayPal" Then
        sProcErrMsg = PayPal()
ElseIf sTransMethod = "1" Then
        sProcErrMsg =  CyberCash(proc_live)
ElseIf sTransMethod = "16" Then
        'sProcErrMsg = CyberCash(proc_live,"1")
        sProcErrMsg = CyberCash(proc_live)
ElseIf sTransMethod = "2" or sTransMethod = "11" or sTransMethod = "13" or sTransMethod = "19" Then
        sProcErrMsg =  AuthNet(proc_live,"1")
ElseIf sTransMethod = "3" or sTransMethod = "17" Then
        sProcErrMsg =  SignioPayProFlow(proc_live)
ElseIf sTransMethod = "4" Then
        sProcErrMsg =  SurePay(proc_live)
ElseIf sTransMethod = "7" Then
        sProcErrMsg =  LinkPoint(proc_live)
ElseIf sTransMethod = "8" Then
        sProcErrMsg = PSIGate(proc_live)
ElseIf sTransMethod = "10" Then
        sProcErrMsg = SecurePay(proc_live)
ElseIf sTransMethod = "18" OR sPaymentMethod = "WorldPay" Then
        sProcErrMsg = WorldPay(proc_live)
End If
```

This is where StoreFront detects which payment processing service is in use and executes the appropriate routine. Add a new case by inserting the following code after sProcErrMsg = WorldPay(proc_live), where [**] is the value from the trnsmthdID field of the record you added to sfTransactionMethods in Step 1:

```
ElseIf sTransMethod = “[**]” Then
        sProcErrMsg = NewPayProc(proc_live)
```

For example:

```
ElseIf sTransMethod = “20” Then
        sProcErrMsg = NewPayProc(proc_live)
```

Save the changes to the file and proceed to the next step.

Step 3: Add a New Payment Processing Routine to Processor.asp
Processor.asp is the page that contains the routines for all integrated payment processing services. The next step in the integration process is to add a routine to this page for your new payment processing service. Open processor.asp using Notepad or another file editor. Locate the following lines:

```
'------------------------------------------------------------------
' CyberCash subroutine
' Requirement: CYCHMCK.DLL 2.0 or higher
' Last edited : October 3, 2000
'------------------------------------------------------------------
```

Begin building your payment processor's routine by entering the following lines of code immediately before these:

```
Function NewPayProc(proc_live)

End Function
```

These lines are the delimiters that will contain the code for your function. Any code that will be executed for your payment processing service must be contained between these lines. At this point, you should consult the integration documentation for your chosen service to determine exactly what this code should be and how information should be passed to the service's gateway. The following table lists all pertinent variables that are available at this point:

Credit Card Information The following variables store the information entered on verify.asp.	
sCustCardNumber	customer's credit card number
sCustCardExpiry	customer's credit card expiration date
sCustCardName	name on the credit card
sCustCardType	customer's credit card type
Customer and Order Information With the exception of iOrderID and sGrandTotal, the following variables store the information entered under the Billing Information section of process_order.asp.	
iOrderID	the customer's Order ID, a unique identifying number assigned to each order.
sCustAddress1	the value entered by the customer in the Street Address 1 field.
sCustAddress2	the value entered by the customer in the Street Address 2 field.
sCustCity	the value entered by the customer in the City field.

sCustState	the value selected by the customer as their State.
sCustZip	the value entered by the customer in the Zip Code field.
sCustCountry	the value selected by the customer as their Country.
sGrandTotal	the grand total of the order. This variable is not formatted as currency.
sCustName	a combination of sCustFirstName and sCustLastName (see below)
sCustFirstName	the value entered by the customer in the First Name field.
sCustMiddleInitial	the value entered by the customer in the Middle Initial field.
sCustLastName	the value entered by the customer in the Last Name field
sCustEmail	the value entered by the customer in the Email field.
sCustCompany	the value entered by the customer in the Company field.
sCustPhone	the value entered by the customer in the Phone field.
sCustFax	the value entered by the customer in the Fax field.
--- The following variables store the information entered in the fields under the Shipping Information section of process_order.asp **---**	
sShipCustFirstName	the value entered by the customer in the First Name field.
sShipCustMiddleInitial	the value entered by the customer in the Middle Initial field.
sShipCustLastName	the value entered by the customer in the Last Name field.
sShipCustCompany	the value entered by the customer in the Company field.
sShipCustAddress1	the value entered by the customer in the Street Address 1 field
sShipCustAddress2	the value entered by the customer in the Street Address 2 field.
sShipCustCity	the value entered by the customer in the City field.

sShipCustState	the value selected by the customer as their State.
sShipCustZip	the value entered by the customer in the Zip code field.
sShipCustCountry	the value selected by the customer as their Country.
sShipCustPhone	the value entered by the customer in the Phone field.
sShipCustFax	the value entered by the customer in the Fax field.
sShipCustEmail	the value entered by the customer in the Email field.
sShipCustName	a combination of sShipCustFirstName and sShipCustLastName.
<u>Payment Processor Information</u> The following variables store login and general configuration information for the payment processing service that is in use. This information is stored in the web store's database, and is input and manipulated through the StoreFront Web Manager. See Step 4 for more information.	
sLogin	the value recorded in the trnsmthdLogin field of the sfTransactionMethods table; usually the username used to access a payment processing service.
sPassword	the value recorded in the trnsmthdPassword field of the sfTransactionMethods table; usually the password used to access a payment processing service.
sPaymentServer.	the value recorded in the trnsmthdServerPath field of the sfTransactionMethods table; usually the URL to a payment processing service's gateway.
sMercType	the value recorded in the adminMerchantType field of the sfAdmin table. This value tells the payment processing service how to process transactions: normal_auth (authorization only) or authcapture (authorize transaction and capture funds).

Use the variables listed above to pass your payment processing service the information it requires to process a transaction. Any additional variables and values you need for your integration can be created within the function as well (be sure to declare any variables using the Dim statement first). To see different ways in which these variables are used for other payment processing services, examine the rest of the code in processor.asp.

Step 3: Handling the Results of a Transaction
Most payment processing services will return a uniform set of name-value pairs listing the results of each transaction. The manner in which these values are parsed and handled is immaterial so long as the data they contain is preserved in usable variables. The following variables must be created at the end of your function and assigned the appropriate values. If your payment processing service does not return any values that can be assigned to a specific variable (for example, not all services will return a value suitable for the iMercTransNo variable) then set that variable equal to nothing (“”). The exceptions to this rule are iProcResponse and ProcResponse, as explained below:

iTransactionID	most payment processing services will return a unique transaction ID for each transaction. This value should be assigned to the iTransactionID variable
iMercTransNo	some payment processing services return a unique identifying code to identify the merchant that submitted each transaction. This value should be assigned to the iMercTransNo variable.
iAVSCode	most payment processing services support Address Verification, a system that checks a customer’s card number against the billing address registered to that card. AVS messages are returned in standardized one-character codes. iAVSCode should be assigned this value.
sAUXMsg	if your payment processing service supports AVS, they will return a single-character AVS Code (see iAVSCode above). The processor.asp file contains a AVSMsg function that you can use to interpret these codes to derive a meaningful error message. If your payment processing service supports AVS and returns an AVS code, set the sAUXMsg variable equal to AVSMsg(YourAVSResponse), where YourAVSResponse is the AVS Code returned by the processing service.
sActionCode	banks and payment processing services generally assign an action code to each transaction to describe the results. This value should be assigned to sActionCode.
iAuthNo	some payment processing services return an authorization code for each transaction, in addition to the transaction ID. This value should be assigned to the iAuthNo variable.
sErrorMessage	when a transaction fails for some reason, most payment processing services return an error messaging describing the reason for the failure. This value should be assigned to the sErrorMessage variable. **Note:** in order for a failed transaction to be recognized by StoreFront, sErrorMessage must have a value. Similarly, for a successful transaction to be recognized by StoreFront, sErrorMessage must be set to nothing, or contain no value. This is the variable that is evaluated to determine the result of a transaction.
iProcResponse :	this variable must be set to a value of 1 or 0 to indicate the

	overall success (1) or failure (0) of a transaction. Many payment processing services return a similar binary success code which can be assigned to this variable. If no such value is present, you must manually set the value of iProcResponse based on other information returned by your service.
sProcType	the name of the payment processing service you are using (keep this brief and abbreviate if necessary).
ProcResponse	this variable is evaluated in some situations to determine the result of a transaction. When a transaction fails, ProcResponse should be set to "fail."
iRetrievalCode	if your payment processing service returns a retrieval code of some sort for each transaction, assign that value to this variable.

After creating the variables shown above, place the following code below them. Although the code is wrapped here, it should be entered all on one line in processor.asp:

```
Call
setResponse(sProcType,iOrderID,iTransactionID,iMercTransNo,iAVSCode,sAUXMsg,sActionCode,iAuthNo,iRe
trievalCode,sErrorMessage,iProcResponse)
```

Following this, enter:

```
NewPayProc = sErrorMessage
```

This will complete the new processing routine.

Step 4: Configuring your Web Store to use the New Payment Processor
Prior to testing, the last step in integrating a new payment processing service into your web store is to select and configure it using the StoreFront Web Manager. Follow the instructions below:

1. Connect to the web store's database using the Web Manager.

2. Click on the Order Processing tab. Select your payment processing service from the **Payment Processing Method** drop-down. Click **Configure**.

3. The values you enter into the input fields in the Configure Payment Method dialog box will be written to the sfTransactionMethods table. In the User Name field, enter the value that should be assigned to the sLogin variable in your function.

4. In the Password field, enter the value that should be assigned to the sPassword variable.

5. In the Payment Server Path field, enter the value that should be assigned to the sPaymentServer variable.

6. Choose the Processing Type to set the sMercType variable to the desired value. "Authorization only" will set sMercType to "normal_auth", while "Authorize and Capture" will set it to "authcapture."

7. Save the settings and test the site.