CyberSourceCartridge

Version 16.9



10/27/2016

Contents

[Summary 5](#_Toc465445166)

[Component Overview 6](#_Toc465445167)

[Functional Overview 6](#_Toc465445168)

[Credit Card Authorization Service 6](#_Toc465445169)

[Taxes 7](#_Toc465445170)

[Address Verification Service (AVS) 8](#_Toc465445171)

[Delivery Address Verification Service (DAV) 8](#_Toc465445172)

[Decision Manager 8](#_Toc465445173)

[Conversion Detail Report 8](#_Toc465445174)

[Payment Tokenization 9](#_Toc465445175)

[Payer Authentication 9](#_Toc465445176)

[Full Authorization Reversal 10](#_Toc465445177)

[Retail Point-of-Sale (POS) 10](#_Toc465445178)

[AliPay Authorization 11](#_Toc465445179)

[Alipay Batch Job 12](#_Toc465445180)

[PayPal Express Authorization [From Cart Page and Mini Cart] 12](#_Toc465445181)

[PayPal Authorization [From Billing Page] 13](#_Toc465445182)

[Secure Acceptance Authorization 14](#_Toc465445183)

[Secure Acceptance Merchant Notification Post Batch Job 15](#_Toc465445184)

[Apple Pay 15](#_Toc465445185)

[Visa Checkout 15](#_Toc465445186)

[Use Cases Scenarios 16](#_Toc465445187)

[Credit Card / VisaCheckout / ApplePay Authorization 16](#_Toc465445188)

[Taxes 18](#_Toc465445189)

[Address Validation Service (AVS) 19](#_Toc465445190)

[Delivery Address Verification Service (DAV) 19](#_Toc465445191)

[Conversion Detail Report 19](#_Toc465445192)

[Payment Tokenization 20](#_Toc465445193)

[Payer Authorization 20](#_Toc465445194)

[Retail Point-of-Sale (POS) 21](#_Toc465445195)

[Alipay Authorization 21](#_Toc465445196)

[PayPal Express Checkout and Authorization 22](#_Toc465445197)

[Credit Card Order Status Mapping with Demandware Order 24](#_Toc465445198)

[Alipay PayPal Order Status Mapping with Demandware Order 24](#_Toc465445199)

[Demandware – Cybersource Service Response Field Mapping for Alipay and Paypal Services 25](#_Toc465445200)

[Secure Acceptance Authorization 29](#_Toc465445201)

[VISA Checkout Decrypt 30](#_Toc465445202)

[Limitations, Constraints 30](#_Toc465445203)

[Compatibility 33](#_Toc465445204)

[Implementation Guide 33](#_Toc465445205)

[Custom Code 33](#_Toc465445206)

[Generic Section 33](#_Toc465445207)

[Credit Card Auth 56](#_Toc465445208)

[Tax Service 56](#_Toc465445209)

[Address Verification Service 61](#_Toc465445210)

[Delivery Address Validation Service 62](#_Toc465445211)

[Full Authorization Reversal 63](#_Toc465445212)

[Payer Authentication Service 63](#_Toc465445213)

[Payment Tokenization Service 63](#_Toc465445214)

[Alipay Authorization 72](#_Toc465445215)

[PayPal Express Checkout [From Cart Page and Mini Cart] 73](#_Toc465445216)

[PayPal Checkout [From Billing Page] 80](#_Toc465445217)

[Retail POS 86](#_Toc465445218)

[ApplePay REST Interface Integration ways with Device/APP 99](#_Toc465445219)

[Visa Checkout 102](#_Toc465445220)

[Secure Acceptance 109](#_Toc465445221)

[Device Fingerprint 126](#_Toc465445222)

[How does it work? 126](#_Toc465445223)

[Setup: 126](#_Toc465445224)

[Hints for the CsDeviceFingerprintRedirectionType: 126](#_Toc465445225)

[Site Configuration 130](#_Toc465445226)

[Configure Payment Processor 130](#_Toc465445227)

[Import Meta Data 130](#_Toc465445228)

[Import Payment Methods 131](#_Toc465445229)

[Configure Services 131](#_Toc465445230)

[Configure Site Preferences 135](#_Toc465445231)

[Batch Jobs 151](#_Toc465445232)

[Secure Acceptance Profile Configuration into CyberSource Business Manager 154](#_Toc465445233)

[Testing 156](#_Toc465445234)

[Authorize Credit Card 156](#_Toc465445235)

[Tax Service 156](#_Toc465445236)

[Address Verification Service (AVS) 156](#_Toc465445237)

[Delivery Address Verification Service (DAV) 157](#_Toc465445238)

[Payment Tokenization 157](#_Toc465445239)

[Full Authorization reversal 157](#_Toc465445240)

[Device Fingerprint 158](#_Toc465445241)

[Payer Authentication 158](#_Toc465445242)

[Retail POS Authorization Request 158](#_Toc465445243)

[Alipay Initiate Request 159](#_Toc465445244)

[Alipay Check Status Request 159](#_Toc465445245)

[Credit Card Capture Request 159](#_Toc465445246)

[Paypal Capture Request 160](#_Toc465445247)

[Secure Acceptance Redirect Create Token Test Service Request 160](#_Toc465445248)

[Apple Pay 161](#_Toc465445249)

[Cartridges Structure and Reference 163](#_Toc465445250)

[Typical Project Plan 164](#_Toc465445251)

[Roles, Responsibilities 164](#_Toc465445252)

[Typical Efforts and Timelines 164](#_Toc465445253)

[Pre-Production Steps 166](#_Toc465445254)

[Known Issues 166](#_Toc465445255)

[CyberSource document links 167](#_Toc465445256)

[Release History 167](#_Toc465445257)

# Summary

This document provides technical overview and implementation details for each CyberSource service integrated within Demandware platform. The CyberSource cartridge extends the functionality of Demandware Storefront, enabling real time access to CyberSource eCommerce transaction services listed below.

* Credit Card Authorization
* CyberSource Address Verification
* Tax Service
* Delivery Address Verification
* Payment Tokenization
* Payer Authentication
* AliPay
  + Domestic
  + International
* PayPal Express checkout
* Credit Card Secure Acceptance Web /Mobile
  + Redirect method
  + Iframe Method
* Credit Card Secure Acceptance Silent Order Post
* Visa Checkout
* ApplePay REST Based Interface
  + To Process Authorization for Encrypted PayLoad
  + To Process Authorization for Cryptogram
* Retail POS
* AliPay Batch Job
* Secure Acceptance Batch Job

# Component Overview

## Functional Overview

### Credit Card Authorization Service

The credit card authorization service pipeline allows storefront application to request for credit authorization for the total order amount. The pipeline makes the credit card authorization web service call to CyberSource authorization service and receive confirmation about the availability of the funds.

The Demandware Cybersource–AuthorizeCreditCard pipeline populates the authorization request with ship-to, bill-to, credit card data, and purchase total data from the basket and invokes the authorization web service call using CyberSource web service API.

Credit Card Authorization sequence flow:

1. Creates CyberSource authorization request using ship-to, bill-to, credit card data, and purchase total data from the current basket.
2. If authorize Payer is configured, then make the authorize payer request, if not ignore and continue with the authorization request.
3. Create credit card authorization request.
4. If DAV is enabled, then set up DAV business rules, as needed.
5. Set up AVS Ignore Result business rule for request with AVS Ignore Flags specification, as needed.
6. Make actual service call to CyberSource Simple Order API.
7. If Delivery Address Verification is enabled, then:
   1. Capture pertinent DAV result information & DAV Reason Code
   2. If DAV fails and DAV On Failure is set to ‘REJECT’, then exit immediately with rejection response
8. If DAV On Failure is set to ‘APPROVE’ and the DAV Reason Code is a fail code (not 100), then:
   1. Exit immediately with declined or review response, as merchant defines
9. Capture pertinent AVS information
10. Validate authorization reason code and set corresponding end node, based on Auth response code.

The list of activities depicted in the following diagram takes place when API request is made for an online credit card authorization: [**Source, CyberSource Credit Card Service, and October 2009**]

**Figure 1** Processing an Online Authorization [**Source, CyberSource Credit Card Service, October 2009**]



1. The customer places an order and provides the credit card number, the card expiration date, and other information about the card.

2   You send a request for authorization over a secure Internet connection. If the customer buys a digitally delivered product or service, you can request both the authorization and the capture at the same time. If the customer buys a physically fulfilled product, do not request the capture until you ship the product.

3   CyberSource validates the order information, and then contacts your payment processor and requests authorization.

4   The processor sends the transaction to the card association, which routes it to the issuing bank for the customer’s credit card. Some card companies, including Discover and American Express, act as their own issuing banks.

5   The issuing bank approves or declines the request. Depending on the card type, the bank could also use the Address Verification Service (AVS) to determine whether the customer provided the correct billing address. For more information about AVS, refer to AVS service documents via the CyberSource Services Documentation at <http://www.cybersource.com/support_center/support_documentation/services_documentation/payment.php> or as described in this integration guide.

6   CyberSource runs its own tests, and then tells you if the authorization succeeded.

7   Response is sent back to the client.

### Taxes

Online Customer adds Product(s) to Cart and proceeds to Checkout.

As soon as shipping information is entered and validated, taxes are updated to reflect current tax rates based on six basic criteria:

1. Customer ship to address
2. Merchant ship from address
3. Merchant point of order origin (POO)
4. Merchant point of order acceptance (POA)
5. Product code
6. Merchant nexus

Product information is provided on an individual line item basis and all merchant/request IDs are captured for future reference. When the customer enters in shipping information, the Tax Service is called to calculate taxes.

### Address Verification Service (AVS)

AVS does not exist as a stand-alone callable service. Please refer to the Credit Card Authorization Service walkthrough for high level walkthrough.

### Delivery Address Verification Service (DAV)

DAV service may be run as a stand-alone callable service, as well as be performed as a part of other services. Please refer to Credit Card Authorization Service for more information regarding the DAV service, as an integral part of payment auth.

As a stand-alone service, the process is defined as:

* Customer enters shipping information
* Shipping information passes client-side validation (required elements filled in)
* Shipping information passes basic server-side validation (syntactically correct)
* Request is made to CyberSource DAV Service
* Response returns DAVReasonCode (100=Success)
* End node returns either: authorized, declined or error (authorized==success, declined==failure)
* Captured validation information is extracted from pipeline dictionary to present user with choices to correct problems, confirm “standardized” formatting or try again
* If service is successful, allow Shipping Address save operation to continue

### Decision Manager

The CyberSource Decision Manager provides Merchant and ability to set business rules, provide case management, and Reporting.

The CyberSource Decision Manager Business rule engine allows Merchant to analyze the order data based on predefined or custom rules. The business rules can be set by orders, by category, or by SKU.

The Demandware CyberSource Cartridge is using an alternate “Conversion Detail Report” Job

### Conversion Detail Report

Cyber Source Conversion report contains the results of the modified orders which were initially in review state. This information gives you an overview of all orders that were not immediately accepted. For each order that is initially marked review and later modified to accept or reject, the report contains below information:

* Request ID
* Status before and after review
* Name of reviewer
* Queue assignment
* Reviewer comments and notes
* Order profile

Request this report at any time during the day, starting up to 24 hours in the past and ending at the present time

The section “Configure Services” has details to configure conversion detail report cybersource service.

The section “Business Manager Changes for batch Jobs” has details about the conversion detail report batch Job, which fetches the last 24 hours updated order status from REVIEW to ACCEPT/REJECT within cybersource and further updates the order status in Demandware accordingly.

### Payment Tokenization

Tokenization is the replacement of sensitive data with a unique identifier that cannot be mathematically reversed. In your environment, tokens take the place of sensitive credit card data. Typically, the token will retain the last four digits of the card as a means of accurately matching the token to the payment card owner. The remaining numbers are generated using proprietary tokenization algorithms.

How It Works

* To make a purchase on your website, the customer will enter their payment card information into the designated payment fields on the order page. These payment fields will be hosted by CyberSource using [Hosted Payment Acceptance](http://www.cybersource.com/products_and_services/payment_security/hosted_payment_acceptance/). When the customer hits the ‘submit’ button, the data is immediately encrypted and transmitted directly to CyberSource for storing, processing, and token generation. The payment data never enters your environment.
* The encrypted primary account number (PAN) is decrypted when it enters CyberSource’s Level 1, PCI-compliant data vault, where it is securely stored. The payment data is then passed on to the processing channel (bank) and returned to CyberSource with an accepted or denied result.
* CyberSource returns the result to you but substitutes the PAN data with a uniquely generated token. You store the token in your database of record system (such as ERP system) for future transactions or chargeback resolution on that account. Customer service representatives can easily verify customers as the custom token will retain the last four digits of the original PAN.

### Payer Authentication

CyberSource Payer Authentication services enable you to add support to your web store for card authentication services, including Visa Verified by VisaSM, MasterCard® and Maestro® SecureCode™ (UK Domestic and international), American Express SafeKeySM, and JCB J/Secure™.

These card authentication services deter unauthorized card use and protect you from fraudulent chargeback activity referred to as *liability shift*.

How It Works

Payer Authentication provides the following services:

* + **Check Enrollment**: Determines whether the customer is enrolled in one of the card authentication programs.
  + **Validate Authentication**: Ensures that the authentication that you receive from the issuing bank is valid.



The Check Enrollment service determines whether the customer is enrolled in one of the

Card authentication services:

* **No**: If the card is not enrolled, you can process the authorization immediately.
* **Yes**: If the card is enrolled, the customer’s browser displays a window where the customer can enter the password associated with the card. This is how the customer authenticates their card with the issuing bank.
* If the password matches the password stored by the bank, you need to verify that the information is valid with the Validate Authentication service. If the identity of the sender is verified, you can process the payment with the Card Authorization service.
* If the password does not match the password stored by the bank, the customer may be fraudulent. You must refuse the card and can request another form of payment.

### Full Authorization Reversal

A full authorization reversal is a follow-on transaction that uses the request ID returned from a previous authorization. The request ID links the full authorization reversal to the authorization. CyberSource uses the request ID to look up the customer’s billing and account information from the original authorization, which means merchant is not required to include those fields in full authorization reversal request.

### Retail Point-of-Sale (POS)

This service of CyberSource enables a merchant to process a credit card for retail point-of-sale transaction at their stores. The integration takes inputs for the API service and provides CyberSource API response for later use. This integration takes care for terminal which has manual entry for credit card details and terminal with a magnetic stripe where a credit card can be swiped and enter amount for the transaction.

### AliPay Authorization

The Alipay authorization service pipeline allows storefront application to request for authorization for total ordered amount along with the currency. The pipeline make the web service call to CyberSource Alipay initiate service to initiate payment request and authorize the amount and after successful initiation pipeline make the web service call to check the payment status of initiated request.

The Demandware CyberSource- AuthorizeAlipay pipeline populates the payment initiate request with purchase total data, product name, product description and Alipay Payment type such as APD (Domestic payment for China based merchant to trade in China) and APY (International payment for International merchant to trade from outside China) and invoke the initiate web service call using CyberSource web service API.

**Alipay Authorization Sequence Flow:**

1. Create CyberSource Alipay Initiate request using purchase total data, product name, and product description (optional) from the current order object
2. Set Alipay payment type to domestic or international in site preference
3. After configuration make actual service call to Alipay Initiate request
4. Validate Reason code and Decision of Initiate request and accordingly set the corresponding end node.
5. If initiation is successful, then assign the required values in Demandware Payment Transaction object and add a node to create CyberSource Alipay Check Status Request using Request ID of Initiate service response
6. Make service call to Alipay Check Status request to return the payment status of initiated request
7. Validate Reason Code and Payment status of check status service response and set the corresponding end node
8. If ReasonCode = 100 then check the payment status. If payment status is COMPLETED for service call then complete the checkout flow and place the order with “New” as order status and “Paid” as order payment status.
9. If ReasonCode = 100 and PaymentStatus = PENDING, complete the checkout flow with order status as “Created” and order payment status as “Not Paid”.
10. If ReasonCode = 100 and PaymentStatus = ABANDONED or PaymentStatus = TRADE\_NOT\_EXIST, fail the order and show message on the screen.
11. If Decision = REJECT and ReasonCode = 102 or ReasonCode = 233, fail the order and show message on the screen.
12. If Decision = ERROR and ReasonCode = 150, fail the order and show message on the screen.

**Note:** As Alipay live environment is not available, so for Alipay Domestic and International scenarios, Site Preference configuration for Reconciliation ID needs to configure to test various scenarios of Alipay Initiate and Check Status service. Also, If shopper does not return from the AliPay then Demandware order status shall remain the same as “Created” and shall be updated once Batch Job for Check Payment Status service runs from scheduler

### Alipay Batch Job

Alipay batch job process Demandware orders placed by Alipay as payment method by making web service call to Alipay Check Status Service.

The Demandware Cybersource-AlipayCheckPaymentStatusWorkflow pipeline node is called from batch job that populates the check status request with Request ID generated and stored in Demandware Payment Transaction custom attribute after Alipay Initiate request service call for every order placed by Alipay as payment method and invoke the Check Status web service call using CyberSource web service API.

**Alipay Batch Job Sequence Flow:**

1. Query on all the Demandware orders placed through Alipay with New, Open and Created as order status and get the Request Id stored in Order Payment Transaction custom object attribute after Alipay Initiate web service call.
2. Pass the Request Id and Payment Type to Alipay Check Status Service and make the actual service call.
3. Validate Reason Code and Payment status of check status service response and set the corresponding end node
4. If ReasonCode = 100 then check the payment status. If payment status is COMPLETED for service call then update the order status to “New”, Order Payment Status to “Paid”, Alipay Payment Status to “COMPLETED” in Order Payment Transaction custom object attribute and set the export status to “Ready For Export”
5. If ReasonCode = 100 and PaymentStatus = PENDING, no need to update any Demandware status in case of PENDING Alipay Payment Status.
6. If ReasonCode = 100 and PaymentStatus = ABANDONED or PaymentStatus = TRADE\_NOT\_EXIST, fail the order.
7. If Decision = REJECT and ReasonCode = 102 or ReasonCode = 233, fail the order.
8. If Decision = ERROR and ReasonCode = 150, fail the order.

Note: Please refer to the order status mapping mentioned in section Alipay Paypal Order Status Mapping with Demandware Order on page 37.

### PayPal Express Authorization [From Cart Page and Mini Cart]

PayPal Express Authorization service pipeline allow storefront application to request for authorization of ordered amount by validating credential of the user without using billing and shipping information from Demandware storefront. The pipeline makes the PayPal authorization web service call to CyberSource authorization service to get confirmation about availability of funds and authorize the amount.

The Demandware Cybersource - AuthorizePaypal pipeline populates the authorization request with item and purchase total data from order object and invokes the authorization web service call using CyberSource web service API.

PayPal Express Authorization Sequence Flow:

1. Create CyberSource Authorization request using item object, purchase total data object and required values as a result of set service, get service and order setup service call.
2. Site preference “PayPal Payment Option” has been provided for either Authorization or Authorization and Capture service calls. If user has selected Authorization from the site preference then only amount will be authorized but order payment status would be “Not Paid” and in case of Authorize and Capture service calls, payment status would become “Paid”.
3. After adding the product into cart, user can either choose express checkout from mini cart or from cart. On clicking express checkout user will be redirected to PayPal to authenticate its credentials. A call to CyberSource PayPal set and get service would be made and user will redirect back to merchant site and shopper basket billing/shipping information shall be updated with PayPal Get Service response and shopper redirected to Review page to complete the order.
4. On placing the order, actual service call to CyberSource Order setup, Authorize and Capture service would be made to authorize the amount.
5. If Decision Manager is enabled from Site Preference and CyberSource console, create CyberSource Authorization request using bill-to, ship-to, item object and purchase data object to make CyberSource PayPal Authorization service call.
6. Validate Reason Code and set corresponding end node, based on Auth response code.

### PayPal Authorization [From Billing Page]

PayPal Authorization service pipeline allow storefront application to request for authorization of ordered amount by validating credential of the user by using billing and shipping information from Demandware storefront. The pipeline makes the PayPal authorization web service call to CyberSource authorization service to get confirmation about availability of funds and authorize the amount.

The Demandware Cybersource - AuthorizePaypal pipeline populates the authorization request with item and purchase total data from order object and invokes the authorization web service call using CyberSource web service API.

PayPal Express Authorization Sequence Flow:

1. Create CyberSource Authorization request using item object, purchase total data object, bill-to, ship-to objects from the order object.
2. Site preference “PayPal Payment Option” has been provided for either Authorization or Authorization and Capture service calls. If user has selected Authorization from the site preference then only amount will be authorized but order payment status would be “Not Paid” and in case of Authorize and Capture service calls, payment status would become “Paid”.
3. After adding the product into cart, user chooses normal checkout flow by choosing PayPal as payment method on billing page. Now “Set CyberSource PayPal service” is invoked to validate the user address incase address issue encountered user returned back to billing page with error message displayed. On successful “Set CyberSource PayPal service” response will return PayPal token that will be passed to get service to get the address details, address verification detail, payer details etc. and also update the shopper basket object payment instrument with corresponding details. On placing an order from review page, a sequential call will be made to all the services. CyberSource Order setup service will setup an order and Authorize and Capture CyberSource PayPal service will authorize the payment and send back the user to merchant site from PayPal site to complete the order.
4. If Decision Manager is enabled from Site Preference and CyberSource console, create CyberSource Authorization request using bill-to, ship-to, item object and purchase data object to make CyberSource PayPal Authorization service call.
5. Validate Reason Code and set corresponding end node, based on Auth response code.

### Secure Acceptance Authorization

Secure Acceptance payment gateway is used to process transaction requests directly from the customer browser so that sensitive payment data does not pass through Demandware servers.

Secure Acceptance feature is implemented using these secure acceptance payment methods:

1. Secure Acceptance – Redirect
2. Secure Acceptance – Iframe
3. Secure Acceptance – Silent Post

All the above secure Acceptance methods provide a common feature of handling the secure information on secure pages only.

**Secure Acceptance Redirect:** Customer will get redirect to secure Acceptance payment gateway when clicking on Place Order from Review Page

**Secure Acceptance Iframe:** Customer will get redirect to secure Acceptance payment gateway within an Iframe embedded in a new summary page added into checkout flow

**Secure Acceptance Silent Order Post:** Credit Card form data is posted to secure acceptance silent post URL and token is generated and user is redirected on review page and normal card authorization flow is being used to further process the transaction.

**Secure Acceptance Web/Mobile Authorization Sequence flow:**

1. Secure Acceptance Authorize pipeline create Request signature using signed and unsigned field names to validate the request on secure pages
2. Pipeline post the request data[i.e: billing/shipping/card details, signature in signed and unsigned fields] in to selected APM form Action
3. Secure Acceptance validate the request using signature and open the secure payment pages to complete the checkout flow
4. After successful checkout completion ,Customer is return back to Demandware custom pipeline[configured in Cybersource profile]
5. Secure Acceptance response pipeline get the response in CurrentHttpParameterMap,again signature is created using the response data and matched with the response signature, once validated response is parsed within the pipeline
6. Based on Decision and reason code Order will get placed or failed in Demandware.

**Secure Acceptance Silent Order Post Authorization Sequence flow:**

1. An Ajax function is created to call Secure Acceptance silent post pipeline to form request data except card details
2. Card details are populated within Ajax to prevent security breach , further the details are posted to selected APM form action URL
3. Silent post will create or update token based on request details and return the response on Demandware custom pipeline which parse the response of CurrentHttpParameterMap and return back to corresponding pages[summary/billing/cart]
4. On Place Order ,Secure Acceptance authorization pipeline is called which internally completed the flow using CyberSource Authorization[refer Credit Card Authorization service]

### Secure Acceptance Merchant Notification Post Batch Job

The batch job process merchant post notifications arrived from cybersource secure acceptance web/mobile. These notification response data get stored in demandware custom object “SA\_MerchantPost”.

Further when batch job runs it update those orders which did not got updated in regular customer checkout journey due to network issues. The job process below scenarios

1.       Order already updated in the checkout journey itself then custom object entry removed for order

2.       Order not updated in checkout journey then merchant post response read from custom object in JSON form and information updated in the order

a.       Billing/shipping address

b.      Order status as New/Failed

c.       Payment authorization response

d.      Card get saved for logged in user if customer opted in checkout journey

Note: It is recommended to have the batch job frequency every 15 min to update order status and release inventory

### Apple Pay

Developed REST Interface as standalone services and cartridge does not have end-to-end direct integration with DW native ApplePay Web/APP functionality. However interface has mechanisms to integrate individual methods with DW Native ApplePay web/APP

There are two type of REST interface supported where information expected in request are:

1.         Payload and order Number data

2.         Network Token, Order Number, Card type, Token Expiration Date, and cryptogram data

### Visa Checkout

Visa Checkout and the CyberSource credit card services work together as an integrated offering. CyberSource provides the following services to assist with your Visa Checkout integration

* Get Visa Checkout data: this service retrieves Visa  checkout data, which  enables you to display payment and shipping details to the customer during checkout.
* Authorization: this service enables you to send an authorization request to your processor using the Visa Checkout payment data



1.       Your web site integrates directly to Visa Checkout to display the Visa Checkout button on your checkout page.

2.       CyberSource provides the get Visa Checkout data service, which retrieves the Visa Checkout payment data, except the PAN. You can use the retrieved data to help the customer confirm the purchase.

3.       You submit an authorization request to CyberSource for credit card processing. Instead of including payment information in the authorization request, you include the Visa Checkout order ID.

4.       At various points in the transaction cycle, you notify the customer of the transaction status.

## Use Cases Scenarios

### Credit Card / VisaCheckout / ApplePay Authorization

The following table outlines the possible Demandware actions based on the response of the CyberSource gateway. Each client may choose to handle the response code differently. As of release 2.10, all errors logged as “fatal”, can activate an email alert to recipients identified in business manager.

|  |  |  |  |
| --- | --- | --- | --- |
| **Response** | **DW Storefront Action** | **Cyber-**  **Source Code** | **CyberSource suggested response** |
|  |  |  |  |
| Successful transaction. | Continue Checkout | 100 |  |
|  |  |  |  |
| **Validation Errors** | | | |
| Request is missing one or more fields | Should not occur as validation should catch this Show user “denied” error message Log fatal error (email alert) | 101 | See the reply fields missingField\_0...N for which fields are missing. Resend the request with the complete information. |
|  |  |  |  |
|  |  |  |  |
| One or more fields in the request contain invalid data. | Should not occur as validation should catch this Show user “denied” error message Log fatal error (email alert) | 102 | See the reply fields invalidField\_0...N for which fields are invalid. Resend the request with the correct information. |
|  |  |  |  |
| **System Errors** | | | |
| General system failure. | Show user “Unable to process – Call Cust Service” error message Log fatal error (email alert) | 150 | Wait a few minutes and resend the request. |
|  |  |  |  |
|  |  |  |  |
| The request was received but there was a server time-out. | Show user “Unable to process – Call Cust Service” error message Log fatal error (email alert) | 151 | Wait a few minutes and resend the request. |
|  |  |  |  |
|  |  |  |  |
| The request was received but there was a service time-out. | Show user “Unable to process – Call Cust Service” error message Log fatal error (email alert) | 152 | Wait a few minutes and resend the request. |
|  |  |  |  |
| The request just wait and then timeout, ends up as exception on the Demandware script | This could be one of the unique scenarios where CyberSource waits for the Merchant’s bank to authorize the order and exceeds timeout sets at the Demandware. This ends up into SOAP exception. Client code can handle this scenario differently. | Script sets Reason Code to 999 | Handle at client’s end depending on business rules associated with this scenario. |
|  |  |  |  |
| **Authorization denied errors** | | | |
| Declined the request because the card has expired. | Show user “Auth denied” error message | 202 | Request a different card or another form of payment. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| The account number is invalid. | Show user “Auth denied” error message | 231 | Request a different card or other form of payment. |
|  |  |  |  |
| **Gateway Account problem** | | | |
| There is a problem with your merchant configuration. | Show user “Unable to process – Call Cust Service” error message Log fatal error (email alert) | 234 | Do not resend the request. Contact Customer Support to correct the configuration problem. |
|  |  |  |  |
| **Fraud Management** | | | |
| The fraud score exceeds your threshold. | Show user “Unable to process – Call Cust Service” error message Log fatal error (email alert) | 400 |  |
|  |  |  |  |
| The order is marked for review by Decision Manager. | Proceed with checkout Leave DW order “unconfirmed” | 480 |  |
|  |  |  |  |
| The order is rejected by Decision Manager. | Show user “Unable to process – Call Cust Service” error message Log fatal error (email alert) | 481 |  |

### Taxes

List of use cases and appropriate action taken listed below:

|  |  |
| --- | --- |
| **Use case scenarios** | **Result** |
| If shipping information is specified, then request is made to the Tax Service | If successful, the contents of the Basket are taxed and price totals are adjusted.  If failed, because of service outage or failed address verification then don’t update anything. Other services must handle AVS/DAV/Service outages before successful checkout and final sales tax calculation. Failure is logged for email notification. |
| Since cybersource charges per request to the tax service, the cartridge has been modified to reduce the number of tax requests. Subsequent tax requests in the current session are only made to cybersource if the line item’s products id, quantity or price has changed or if the basket merchandise price total (including order level and product level), adjusted shipping price totals or adjusted basket total price has changed. | If the basket state that would affect tax has changed then a tax call will be made to cybersource and the basket will be updated with the new tax prices.  If the basket state that would affect tax has not change, the request to cybersource is skipped. |

### Address Validation Service (AVS)

Note that AVS does not run as an independent process, but is instead an optional, integrated aspect of payment authorization. List of use cases and appropriate action taken listed below:

|  |  |
| --- | --- |
| **Use case scenarios** | **Result** |
| AVS Ignore Result set to true | AVS Information is captured, but does not affect authorization response. |
| AVS Ignore Result set to false | AVS information is captured and if result from AVS is error or declined, then propagates that result up to the calling service. |
| AVS Ignore Result is set to false & AVS Decline Flags is defined | Seed request with additional result codes which should also result in a declined response. |

### Delivery Address Verification Service (DAV)

List of use cases and appropriate action taken listed below:

|  |  |
| --- | --- |
| **Use case scenarios** | **Result** |
| DAV Enable is set to false | No DAV information will be requested. No correction/validation information will be collected from the response. |
| DAV Enable is set to true, DAV On Failure set to REJECT | DAV information will be requested from the calling service. DAV related corrections and validation information is captured, and a DAV-related failure will be propagated to the calling service. |
| DAV Enable is set to true, DAV On Failure set to APPROVE | DAV information will be requested from the calling service. DAV related corrections and validation information is captured, but the result does not affect Authorization result. |

### Conversion Detail Report

This job pick orders which are initially having decision as REVIEW in cybersource and later their decision modified as “ACCEPT” or “REJECT” in last 24 hours. The order status is updated in Demandware through the incoming xml of conversion detail report.

List of use cases and appropriate action taken listed below:

|  |  |
| --- | --- |
| **Use case scenarios** | **Result** |
| Incoming order status is set to “ACCEPT” | Read order from the order table; Update the status in demandware  The order statuses modified after conversion detail report ran successfully  Order Confirmation Status as CONFIRMED  Order Status as NEW/OPEN  Export Status as Ready For Export |
| Incoming order status is set to “REJECT” | Read order from the order table; update the status in demandware  The order statuses modified after conversion detail report ran successfully  Order Status as CANCELLED |

### Payment Tokenization

Payment Tokenization service stores the customer and card related sensitive data for future reuse. Updates order object with the subscription id received from Cybersource. Now tokenization will work along with Payer Authorization as well.

List of use cases and appropriate action taken listed below:

|  |  |
| --- | --- |
| **Use case scenarios** | **Result** |
| Create subscription response is set to “ACCEPT” | Place the order and update the order object with subscription id.  The subscription ID to be updated in field creditCardToken, this field not visible in BM order , though it is visible under order export XML file inside tag <CrdetCardToken> |
| Create subscription response is set to “REJECT” | Place the order but leave the subscription field empty. Make entry in log files to record the event. |

### Payer Authorization

List of use cases and appropriate action taken listed below:

|  |  |
| --- | --- |
| **Use case scenarios** | **Result** |
| Enrolment Check Error | Merchant proceeds to authorization (optional) |
| Cardholder Not Participating | Merchant proceeds to authorization |
| Unable To Verify Enrolment | Merchant proceeds to authorization (optional) |
| Successful Authentication | Merchant proceeds to authorization |
| Authentication Failure | Merchant asks for another form of payment |
| Attempted Authentication | Merchant proceeds to authorization |
| Authentication Unavailable | Merchant proceeds to authorization (optional) |
| Invalid Authentication Response | Merchant asks for another form of payment |
| PARes Signature Error | Merchant asks for another form of payment |
| Whitespace in PARes | Merchant proceeds to authorization |

### Retail Point-of-Sale (POS)

The use case for POS can be achieved by two scenarios:

1. Hardware - swipe credit card – (A Bluetooth scanning device must be paired to the iPad device.)

On Payments page, we listen for credit card swipes only after the user has entered the amount for Credit Card and tapped enter.

**Expected Result**: The swiped credit card is read and payment is made to the order

1. **Hardware - manually enter credit card with keypad**: (A Bluetooth scanning device must be paired to the iPad device.)

From Payments page, enter amount to be applied to credit card.

**Expected Result:** Manually enter credit card number on device and payment is accepted

### Alipay Authorization

The following table outlines the possible Demandware actions based on the response of the CyberSource gateway. Each client may choose to handle the response code differently.

|  |  |  |  |
| --- | --- | --- | --- |
| **Response** | **DW Storefront Action** | **Cyber-**  **Source Code** | **CyberSource suggested response** |
| Successful transaction. | Continue Checkout | 100 |  |
| **Validation Errors** | | | |
| Request is missing one or more fields | Should not occur as validation should catch this Show user “denied” error message Log error message into Demandware logs | 101 | See the reply field’s missingField\_0...N for which fields are missing. Resend the request with the complete information. |
| One or more fields in the request contain invalid data. | Should not occur as validation should catch this Show user “denied” error message Log error message into Demandware logs | 102 | See the reply field’s invalidField\_0...N for which fields are invalid. Resend the request with the correct information. |
| General decline by the processor | Should not occur as validation should catch this Show user “denied” error message Log error message into Demandware logs | 233 | Request that the customer select a different form of payment. |
| **System Errors** | | | |
| General system failure. | Show user “Unable to process – Call Customer Service” error message Log fatal error | 150 | Wait a few minutes and resend the request. |
| The request just wait and then timeout, ends up as exception on the Demandware script | This could be one of the unique scenarios where CyberSource waits for the Merchant’s bank to authorize the order and exceeds timeout sets at the Demandware. This ends up into SOAP exception. Client code can handle this scenario differently. | Script sets Reason Code to 999 | Handle at client’s end depending on business rules associated with this scenario. |

### PayPal Express Checkout and Authorization

The following table outlines the possible Demandware actions based on the response of the CyberSource gateway. Each client may choose to handle the response code differently.

|  |  |  |  |
| --- | --- | --- | --- |
| **Response** | **DW Storefront Action** | **Cyber-**  **Source Code** | **CyberSource suggested response** |
|  |  |  |  |
| Successful transaction. | Continue Checkout | 100 |  |
|  |  |  |  |
| **Validation Errors** | | | |
| Request is missing one or more fields | Should not occur as validation should catch this Show user “denied” error message Log error message into Demandware logs | 101 | See the reply field’s missingField\_0...N for which fields are missing. Resend the request with the complete information. |
|  |  |  |  |
|  |  |  |  |
| One or more fields in the request contain invalid data. | Should not occur as validation should catch this Show user “denied” error message Log error message into Demandware logs | 102 | See the reply field’s invalidField\_0...N for which fields are invalid. Resend the request with the correct information. |
|  |  |  |  |
| **System Errors** | | | |
| General system failure. | Show user “Unable to process – Call Customer Service” error message Log fatal error | 150 | Wait a few minutes and resend the request. |
|  |  |  |  |
|  |  |  |  |
| The request was received but there was a server time-out. | Show user “Unable to process – Call Customer Service” error message Log fatal error | 151 | Wait a few minutes and resend the request. |
|  |  |  |  |
| The request just wait and then timeout, ends up as exception on the Demandware script | This could be one of the unique scenarios where CyberSource waits for the Merchant’s bank to authorize the order and exceeds timeout sets at the Demandware. This ends up into SOAP exception. Client code can handle this scenario differently. | Script sets Reason Code to 999 | Handle at client’s end depending on business rules associated with this scenario. |
|  |  |  |  |
| **Authorization denied errors** | | | |
| PayPal rejected the transaction. | Show user “Unable to process – Call Customer Service” error message Log error message into Demandware logs | 223 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| General decline by PayPal. | Show user “Unable to process – Call Customer Service” error message Log error message into Demandware logs | 233 | Request a different form of payment option at PayPal Website. |
|  |  |  |  |
| **Gateway Account problem** | | | |
| There is a problem with your CyberSource merchant configuration. | Show user “Unable to process – Call Customer Service” error message Log error message into Demandware logs | 234 | Do not resend the request. Contact Customer Support to correct the configuration problem. |
| PayPal rejected the transaction. A successful transaction was already completed for this PayPal Token value. | Show user “Unable to process – Call Customer Service” error message Log error message into Demandware logs | 238 |  |
|  |  |  |  |
| **Fraud Management** | | | |
| The order is marked for review by Decision Manager. | Proceed with checkout Leave DW order “unconfirmed” | 480 |  |
|  |  |  |  |
| The order is rejected by Decision Manager. | Show user “Unable to process – Call Customer Service” error message Log error message into Demandware logs | 481 |  |

**CyberSource PayPal Transactional Flow**:

**Step 1:** Set Service request and reply— accept item object, bill to, ship to objects, purchase data to generate the PayPal token.

**Step 2:** Get Service request and reply — accept request id, request token and PayPal token generated by set service and return address verification response, payer details and address details.

**Step 3:** Order Setup request and reply— accept payer id and order details to generate order setup response required to authorize the request.

**Step 4:** Authorization service request and reply — accept order related details and authorize the order amount.

**Step 5:** Capture service request and reply — capture the amount authorized by Authorization service.

**Use Case 1:** Checkout using PayPal Express Checkout on Cart Page

“PayPal Checkout” button has been added on Demandware reference SiteGenesis.

**Use Case 2:** Checkout using “PayPal Checkout” button on mini cart

**Use case 3**: Checkout using Pay Pal as payment method on Payment page.

### Credit Card Order Status Mapping with Demandware Order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CYB Status** | **Order Status** | **Confirmation status** | **Payment status** | **Export status** |
| Auth/Accept | New/Open | Confirmed | Not Paid | Ready for export |
| Capture | New/Open | Confirmed | Paid |  |
| Pending/Review | Created | Not Confirmed | Not Paid | Not Exported |
| Reject/Decline | Failed | Not confirmed | Not Paid | Not Exported |

### Alipay PayPal Order Status Mapping with Demandware Order

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Payment Method** | **Cybersource Service Call** | **Payment Status or Reason Code** | **DW Order Status Mapping** | **DW Payment Status Mapping** | **DW Export Status Mapping** |
| Alipay | apCheckStatusService | COMPLETED | New | Paid | Ready For Export |
| Alipay | apCheckStatusService | PENDING | Created | Not Paid | Not Exported |
| Alipay | apCheckStatusService | ABANDONED | Failed | Not Paid | Not Exported |
| Alipay | apCheckStatusService | TRADE\_NOT\_EXIST | Failed | Not Paid | Not Exported |
| Alipay | apCheckStatusService | REJECT | Failed | Not Paid | Not Exported |
| Alipay | apCheckStatusService | ERROR | Failed | Not Paid | Not Exported |
| Paypal | payPalAuthorizationService | ReasonCode=100 | New | Not Paid | Ready For Export |
| Paypal | payPalAuthorizationService | ReasonCode!=100 | Failed | Not Paid | Not Exported |
| Paypal | payPalCaptureService | Completed | New | Paid | Ready For Export |

### Demandware – Cybersource Service Response Field Mapping for Alipay and Paypal Services

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Service Response Field** | **Demandware Field** |
| **apInitiateService** | apInitiateService.returnURL | PaymentInstrument.paymentTransaction.custom.apMerchantURL |
|  | reasonCode | PaymentInstrument.paymentTransaction.custom.approvalStatus |
|  | apInitiateReply.reconciliationID | PaymentInstrument.paymentTransaction.custom.apInitiatePaymentReconciliationID |
|  | requestID | PaymentInstrument.paymentTransaction.custom.apInitiatePaymentRequestID |
|  | requestToken | PaymentInstrument.paymentTransaction.custom.requestToken |
|  | PaymentProcessor | PaymentInstrument.paymentTransaction.paymentProcessor |
| **apCheckStatusService** | apCheckStatusService.paymentStatus | Order.paymentTransaction.custom.apPaymentStatus |
|  | reasonCode | Order.paymentTransaction.custom.apCheckStatusServiceApprovalStatus |
|  | apCheckStatusService.reconciliationID | Order.paymentTransaction.custom.apCheckStatusReconciliationID |
|  | requestID | Order.paymentTransaction.custom.requestId |
|  | requestToken | Order.paymentTransaction.custom.apCheckStatusRequestToken |
|  | apCheckStatusService.processorTransactionID | Order.paymentTransaction.custom.apCheckStatusProcessTransactionID |
| **payPalEcSetService** | payPalEcSetService.correlationID | PaymentInstrument.paymentTransaction.custom.paypalSetRequestCorrelationID |
| **payPalEcGetDetailsService** | requestToken | PaymentInstrument.paymentTransaction.custom.paypalEcSetRequestToken |
|  | requestID | PaymentInstrument.paymentTransaction.custom.paypalEcSetRequestID |
|  | payPalEcGetDetailsService.payerId | PaymentInstrument.paymentTransaction.custom.payPalPayerId |
|  | payPalEcGetDetailsService.paypalToken | PaymentInstrument.paymentTransaction.custom.paypalToken |
|  | payPalEcGetDetailsService.avsCode | PaymentInstrument.paymentTransaction.custom.paypalAvsCode |
|  | payPalEcGetDetailsService.payerStatus | PaymentInstrument.paymentTransaction.custom.paypalGetDetailsPayerStatus |
|  | payPalEcGetDetailsService.addressStatus | PaymentInstrument.paymentTransaction.custom.paypalGetDetailsAddressStatus |
|  | payPalEcGetDetailsService.correlationID | PaymentInstrument.paymentTransaction.custom.paypalGetDetailsCorrelationId |
|  | payPalEcGetDetailsService.paypalTaxAmount | PaymentInstrument.paymentTransaction.custom.paypalGetDetailsTaxAmount |
|  | payPalEcGetDetailsService.payerFirstname | PaymentInstrument.paymentTransaction.custom.paypalPayerFirstName |
|  | payPalEcGetDetailsService.payerLastname | PaymentInstrument.paymentTransaction.custom.paypalPayerLastName |
|  | payPalEcGetDetailsService.payerCountry | PaymentInstrument.paymentTransaction.custom.paypalPayerCountry |
|  | payPalEcGetDetailsService.paypalBillingAgreementAcceptedStatus | PaymentInstrument.paymentTransaction.custom.paypalBillingAgreementAcceptedStatus |
|  | payPalEcGetDetailsService.payer | Basket.customerEmail |
|  | payPalEcGetDetailsService.shipToName | OrderAddress.firstName + OrderAddress.lastName |
|  | payPalEcGetDetailsService.payerPhone | OrderAddress.phone |
|  | payPalEcGetDetailsService.shipToAddress1 | OrderAddress.address1 |
|  | payPalEcGetDetailsService.shipToAddress2 | OrderAddress.address2 |
|  | payPalEcGetDetailsService.shipToCity | OrderAddress.city |
|  | payPalEcGetDetailsService.shipToZip | OrderAddress.postalCode |
|  | payPalEcGetDetailsService.shipToCountry | OrderAddress.countryCode |
|  | payPalEcGetDetailsService.shipToState | OrderAddress.state |
|  | payPalEcGetDetailsService.payerFirstname | OrderAddress.firstName |
|  | payPalEcGetDetailsService.payerMiddlename | OrderAddress.secondName |
|  | payPalEcGetDetailsService.payerLastname | OrderAddress.lastName |
|  | payPalEcGetDetailsService.payerPhone | OrderAddress.phone |
|  | payPalEcGetDetailsService.street1 | OrderAddress.address1 |
|  | payPalEcGetDetailsService.street2 | OrderAddress.address2 |
|  | payPalEcGetDetailsService.city | OrderAddress.city |
|  | payPalEcGetDetailsService.postalCode | OrderAddress.postalCode |
|  | payPalEcGetDetailsService.countryCode | OrderAddress.countryCode |
|  | payPalEcGetDetailsService.state | OrderAddress.state |
| **payPalEcOrderSetupService** | requestID | PaymentInstrument.paymentTransaction.custom.orderSetupRequestId |
|  | requestToken | PaymentInstrument.paymentTransaction.custom.orderSetupRequestToken |
|  | payPalEcOrderSetupService.transactionId | PaymentInstrument.paymentTransaction.custom.orderSetupTransactionId |
|  | payPalEcOrderSetupService.paypalPaymentStatus | PaymentInstrument.paymentTransaction.custom.paypalOrderSetupPaymentStatus |
|  | payPalEcOrderSetupService.paypalTransactiontype | PaymentInstrument.paymentTransaction.custom.paypalOrderSetupTransactionType |
|  | payPalEcOrderSetupService.correlationID | PaymentInstrument.paymentTransaction.custom.paypalOrderSetupCorrelationID |
|  | payPalEcOrderSetupService.paypalPendingReason | PaymentInstrument.paymentTransaction.custom.paypalOrderSetupPendingReason |
| **payPalAuthorizationService** | requestID | PaymentInstrument.paymentTransaction.custom.requestId |
|  | requestToken | PaymentInstrument.paymentTransaction.custom.requestToken |
|  | reasonCode | PaymentInstrument.paymentTransaction.custom.approvalStatus |
|  | payPalAuthorizationService.transactionId | PaymentInstrument.paymentTransaction.custom.paypalAuthTransactionID |
|  | payPalAuthorizationService.amount | PaymentInstrument.paymentTransaction.custom.authAmount |
|  | payPalAuthorizationService.paypalAmount | PaymentInstrument.paymentTransaction.custom.paypalAuthorizedAmount |
|  | payPalAuthorizationService.protectionEligibility | PaymentInstrument.paymentTransaction.custom.paypalProtectionEligibility |
|  | payPalAuthorizationService.protectionEligibilityType | PaymentInstrument.paymentTransaction.custom.paypalProtectionEligibilityType |
|  | payPalAuthorizationService.correlationID | PaymentInstrument.paymentTransaction.custom.paypalCorrelationId |
| **payPalCaptureService** | payPalCaptureService.paypalPaymentStatus | PaymentInstrument.paymentTransaction.custom.paypalPaymentStatus |
|  | payPalCaptureService.parentTransactionId | PaymentInstrument.paymentTransaction.custom.paypalParentTransactionId |
|  | payPalCaptureService.authorizationId | PaymentInstrument.paymentTransaction.custom.paypalAutorizationId |
|  | payPalCaptureService.paypalReceiptId | PaymentInstrument.paymentTransaction.custom.paypalReceiptId |
|  | payPalCaptureService.transactionId | PaymentInstrument.paymentTransaction.custom.paypalCaptureTransactionID |
|  | requestID | PaymentInstrument.paymentTransaction.custom.paypalCaptureRequestId |
|  | requestToken | PaymentInstrument.paymentTransaction.custom.paypalCaptureRequestToken |
|  | payPalCaptureService.paypalFeeAmount | PaymentInstrument.paymentTransaction.custom.paypalCaptureFeeAmount |
|  | payPalCaptureService.correlationID | PaymentInstrument.paymentTransaction.custom.paypalCaptureCorrelationID |

### Secure Acceptance Authorization

Following are the list of reason codes received for Secure Acceptance payment service response. System shall be handling these codes and change the Demandware status accordingly.

|  |  |  |
| --- | --- | --- |
| **Decision** | **Description** | **CyberSource hosted Decision** |
| Successful transaction. | Successful transaction. Reason codes 100 and 110. | 100 |
| Request is missing one or more fields | Authorization was declined; however, the capture may still be possible. Review payment details. See reason codes 200, 201, 230, 480, and 520. | 101 |
| One or more fields in the request contain invalid data. | Transaction was declined. See reason codes 102, 202, 203, 204, 205, 207, 208, 210, 211, 221, 222, 231, 232, 233, 234, 236, 240, 475, 476, and 481. | 102 |
| General decline by the processor | Access denied, page not found, or internal server error. See reason codes 102, 104, 150, 151 and 152 | 233 |
| General system failure. | The customer did not accept the service fee conditions. ν The customer cancelled the transaction. | 150 |
| Create Token Service | Silent Post Service for create token when user enter card details on billing page on merchant site | 100 |
| Update Token Service | Silent Post Service for create token when user choose existing saved cards on billing page on merchant site | 100 |
| Authorization and Create Token Service | Redirect or Iframe service for Authorization and create token when no saved card is choosen | 100 or 480 |
| Authorization and update Token Service | Redirect or Iframe service for Authorization and create token when user choose saved card | 100 or 480 |
| Authorization Service | Redirect or Iframe service for Authorization when tokenization is disabled from BM | 100 or 480 |

### VISA Checkout Decrypt

List of use cases and appropriate action taken listed below:

|  |  |  |
| --- | --- | --- |
| **Service Flow** | **Description** | **CyberSource hosted Decision** |
| Decrypt | Accept – review page displayed decrypted details | 100 |
| Decrypt | Error -  System – user redirect to cart page with standard error message | 150 |
| Authorization | Behavior would remain same as card flow, where confirmation page displayed on successful authorization and review page with error message in case of error | Same as Credit Card Reason code |

## Limitations, Constraints

Not currently implemented:

* Multiple shipments.  Tax rates are only calculated for a single shipment per order.  To implement tax service calculation for multiple shipments, a separate web service call must be made for each distinct “ship to” location.
* Custom User Interface components to correct address validation (DAV/AVS) errors and/or omissions or to confirm “standardized” address format corrections.  All pertinent data is collected, but because each merchant will have customized specifications how to deal with such information (or use other 3rd party solutions to play the same role); no default user interface is provided.
* Custom user interface for view, update and delete subscription. All functionalities are created and working in stand-alone mode in **Cybersource\_Subscription.xml** pipeline. They have to customized and integrated as per the merchant specific needs.
* Custom user interface for Full Authorization Reversal. Full Authorization reversal is created and working in stand-alone mode in **Cybersource\_Services.xml** pipeline. It has to customized and integrated as per the merchant specific needs.
* Cartridge does not provide changes to support the styling of error and validation messages. Merchant need to make the required changes to meet the style guide for error and validation messaging as per their storefront implementation
* Cartridge support  DW provided form field validations only

Currently implemented with limitations and constraints:

* Testing of Alipay is possible only with Test data provided by CyberSource such as Reconciliation ID that is getting passed to Alipay Initiate Service to get the response back. We don’t have Alipay simulator and access to Alipay live environment.
* CNY is the only hardcoded currency value that has been used for Alipay Domestic requests.
* Incase user has enabled Decision Manager from CyberSource console for cards, its mandatory to enable Decision Manager from Business Manager Site Preference path: Site -> Site Preferences -> Custom Preferences -> Cybersource -> Decision Manager Enable for Card -> check/uncheck as per decision manager enabled/disabled in CyberSource console.
* Incase user has enabled Decision Manager from CyberSource console for PayPal, its mandatory to enable Decision Manager from Business Manager Site Preference path: Site -> Site Preferences -> Custom Preferences -> Cybersource\_paypal -> check/uncheck as per decision manager enabled/disabled in CyberSource console.
* Merchant to decide “Master Card Auth Indicator” as “Pre  Authorization” “FinalAuthorization ” or “Undefined” for site payments done via master card.
* Order should remain in same state if user closes the browser while transaction is in progress. For example: For Alipay, if user closes the browser while coming back from simulator and before coming to order confirmation page, order will remain in created state. Similar is the case with PayPal and rest of the payment methods.
* In case of PayPal Express checkout, site specific default shipment method would be passed while placing an order. For example if Ground is the shipment method default selected in Business Manager for a particular site, then for PayPal express checkout Ground method would be passed by default as Shipment method
* Since Alipay and PayPal are dealing with different order and payment status, merchant need to take care of different order messaging on order confirmation page after successful completion of order.
* Inactivate the step link “PlaceOrder” inside checkout journey for PayPal Express checkout and PayPal Billing. This constraint added to ensure user should move from billing page to review page only via continue button, otherwise gift certificate won’t be redeemed when multiple payment method available on billing page.
* Unit Test Services are developed for the standalone testing purpose only and should not be used directly into production
* Cross Site request forgery checks are not implemented on Redirect APM return URLs, as it is not advisable to pass the CSRF token out of DW
* Limitations for secure Acceptance :
  1. Secure Acceptance silent post profile MUST use option of single page form under payment forms section.
  2. Merchant need to provide explicit field validation if required as Demandware Cartrige support native validations only
  3. Limit storefront order setting must be disable if Merchant post URL is configured
  4. It is recommended to enable to CVN check on Secure Acceptance profiles on CyberSource business manager console to ensure extended security
  5. Merchant need to ensure that Billing/Payment pages should be on HTTPS for secure acceptance
  6. Cartridge supports five types of cards in secure acceptance (Visa, master card, amex, maestro international, discover)
  7. Billing Address fields passed in secure acceptance services are limited to the mandatory fields specified in the cybersource documentation
  8. Iframe journey depicted in cartridge has new page similar to summary page where edit links in summary section is removed, further merchant can make other site links disabled as per need.
  9. Current cartridge is developed and tested for en\_US locale
  10. Cartridge supports two generic error message (denied, unable to process call cust service), merchant can further may customize on need basis. Error messages are coming from resource bundles for localization support.
* Limitations for Visa Checkout:

1. “Save Card” option will not be available in the demandware checkout journey, which means tokenization will not be applicable for Visa Transactions
2. Cartridge will send the payload for decrypt/authorization where cybersouce must interpret the required information out of it
3. Cybersource must take into account Fraud and Risk details, AVS and card security codes available in PayLoad during transaction authorization, Cartridge will not be performing any additional security/risk checks except the existing CC Auth response handling

* Limitations for Apple Pay  REST Interface

1. Cartridge provides only REST based interfaces to process the ApplePay Transactions which accepts a predefined inputs in request and provides a customized Transaction response in JSON format
2. Tokenization and Payer authentication is not supported with ApplePay Transactions
3. Developed REST Interface  are just standalone services only and does not support direct integration with DW native ApplePay Web functionality, however interface is developed in such a way that Merchant can use individual methods to integrate with DW Native ApplePay web
4. Cybersource must take into account Fraud and Risk details, AVS and card security codes available in PayLoad during transaction authorization, Cartridge will not be performing any additional security/risk checks except the existing CC Auth response handling

## Compatibility

This cartridge is supported under Demandware Site genesis release 16.9

# Implementation Guide

## Custom Code

**Note:** While integration either implement pipeline changes or controller file changes depending on the storefront site developed on pipeline or controller.

Modify the references of actual storefront cartridges in cybersource cartridges during cybersource integration

### Generic Section

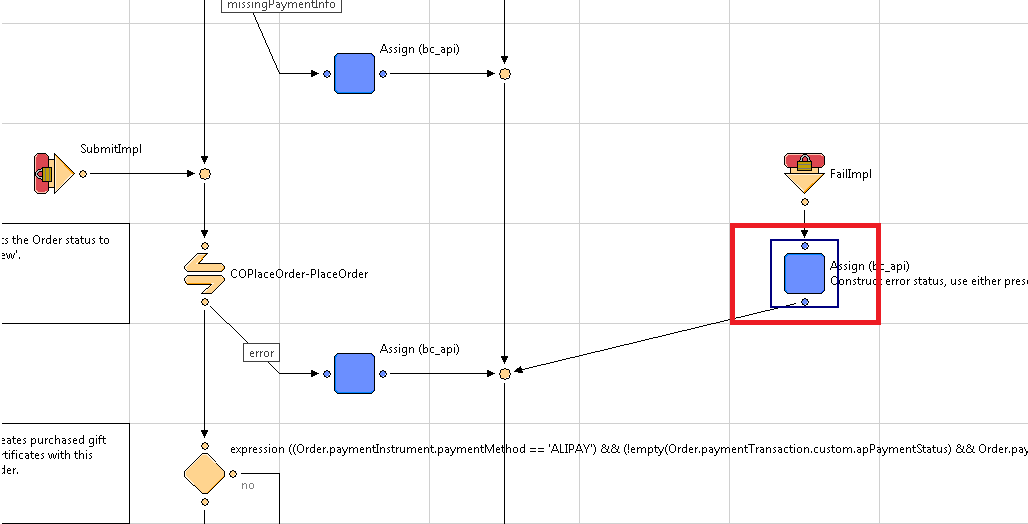
#### Pipeline - COPlaceOrder.xml

##### Update “start” node

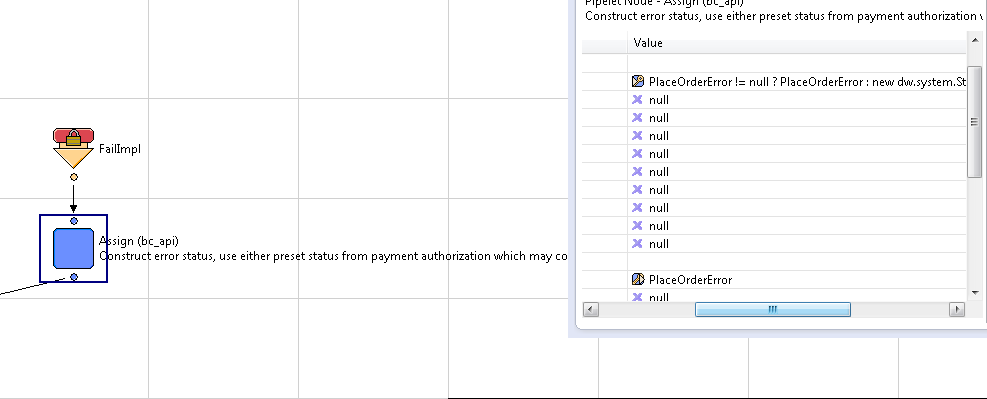
Remove call node CSRF-Validate after place order

##### Update “FailImpl” node

Add an assign node in COPlaceOrder-FailImpl to show the error message on screen in case of error scenarios.

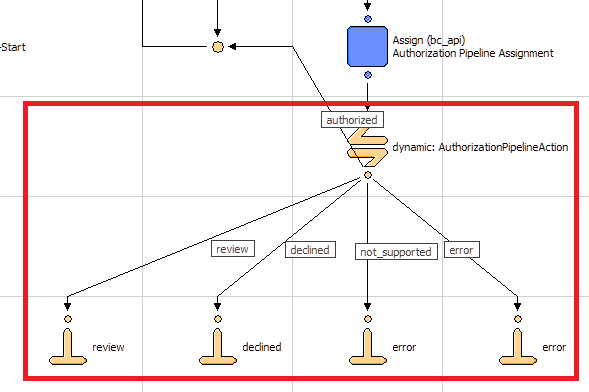


* Assign PlaceOrderError != null ? PlaceOrderError : new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined") value in the pipeline Input Dictionary as shown below.



##### Update “HandlePayments” node

Update HandlePayments pipeline to handle response code returned by CyberSource. Add different end nodes to handle various response returned byservice call.

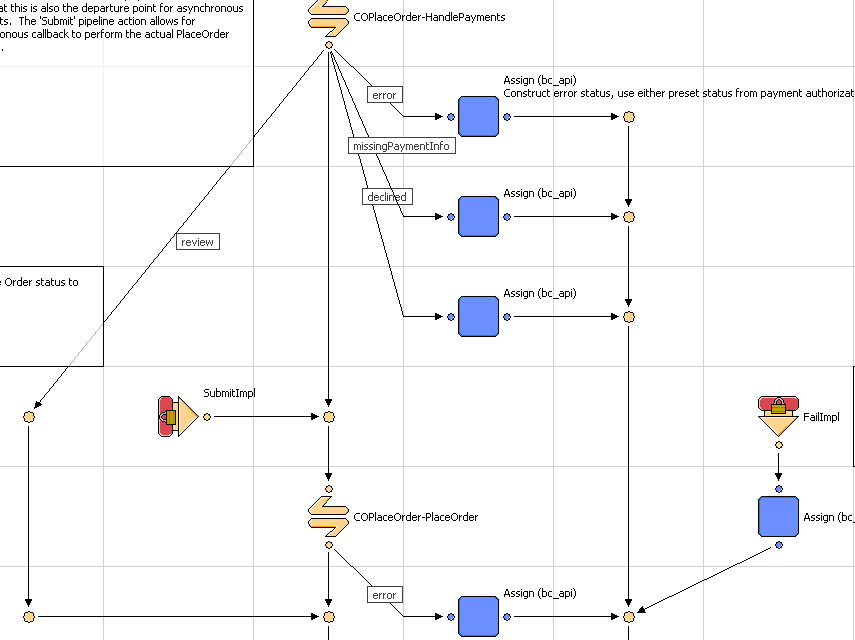


##### Update “start” node

Handle different responses returned from CyberSource along with tracing error in case of different scenarios

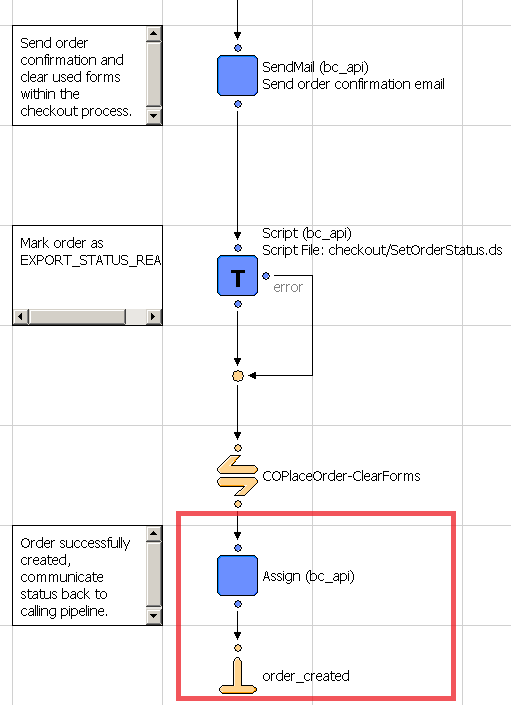
* Add assign node and set the value of PlaceOrderError as PlaceOrderError != null ? PlaceOrderError : new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined") for error connector pipelet
* Add assign node and set the value of PlaceOrderError as new dw.system.Status(dw.system.Status.ERROR, "confirm.error.session") for missingPaymentInfo connector pipelet
* Add assign node and set the value of PlaceOrderError as new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined") for declined connector pipelet
* Add a new transition for “review” and connect it just after COPlaceOrder-PlaceOrder and before COPlaceOrder-CreateGiftCertificates

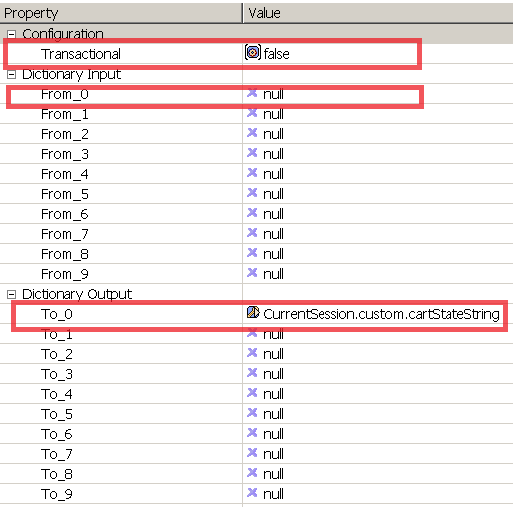
Refer below



##### Update “Start” node with cartstateString in Session

Set cartStateString parameter (CurrentSession.custom.cartStateString) in current session to null after order has been placed, just before order\_createdend node in the COPlaceOrder-Start pipeline. Refer to the following screenshots:





##### Add “ReviewOrder” Node

###### Add new private node “ReviewOrder”

###### Add send Mail pipelet:

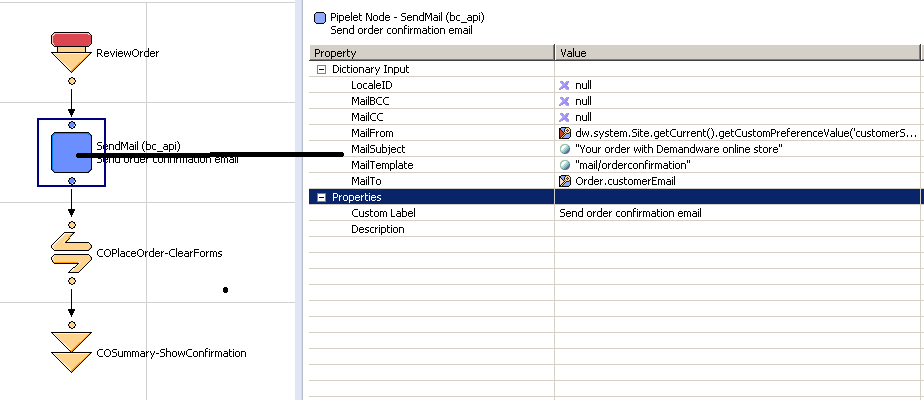
Value of the fields are:

* MailSubject : "Your order with Demandware online store"
* MailTemplate : "mail/orderconfirmation"
* MailTo : Order.customerEmail
* MailFrom : dw.system.Site.getCurrent().getCustomPreferenceValue('customerServiceEmail')

###### Add call node COPlaceOrder-ClearForms

###### Add jump Node COSummary-ShowConfirmation

Kindly refer to following screen shot:



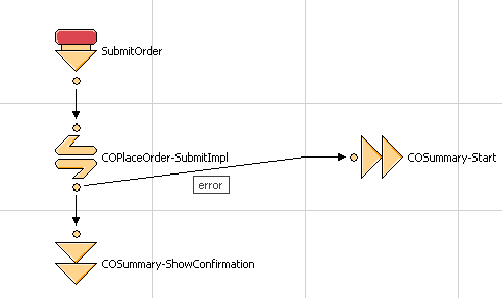
##### Add “SubmitOrder” Node

###### Add new private node “SubmitOrder”

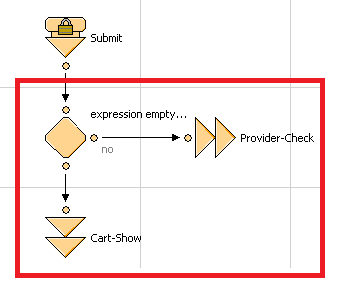
###### Add call node with value COPlaceOrder-SubmitImpl

1. Add jump Node COSummary-Start on error transition
2. Add jump Node COSummary-ShowConfirmation on success

Kindly refer following screen shot

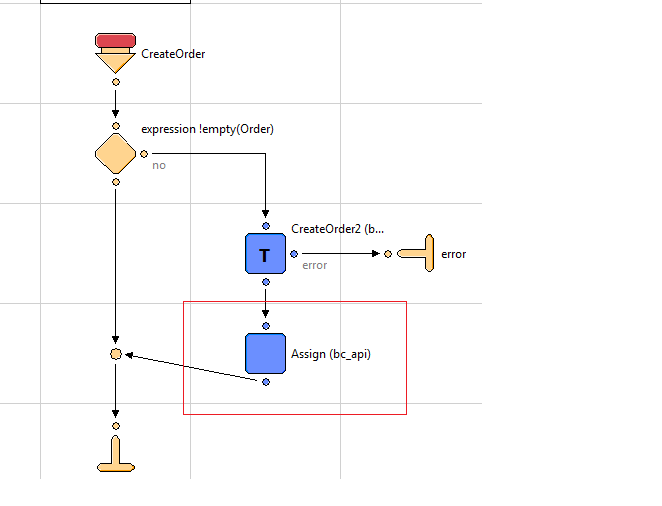
****

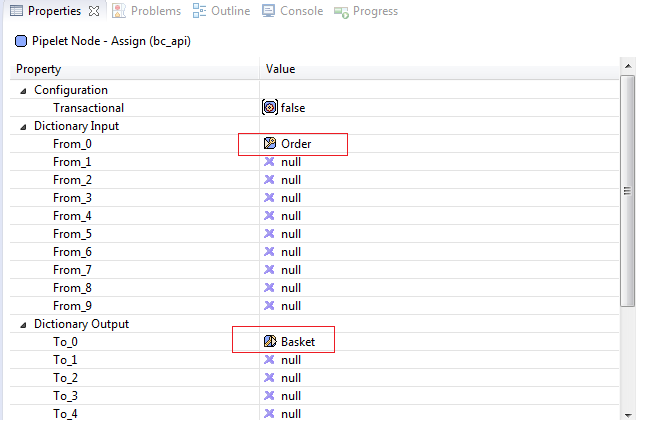
##### Update”Submit” node

1. Remove existing logic (if any)
2. Add decision node to check “provider” exists in CurrentHttpParameterMap empty(CurrentHttpParameterMap.provider.stringValue)
3. If exists then call Provider-Check
4. If does not exists call Cart-Show to redirect user to cart page [no error message displayed]. This indicates either some wrong configuration or user tries to modify the URL

##### Update “CreateOrder” node

* Include an assign node just after the createorder2 pipelet. Refer to the screenshot below for more details:

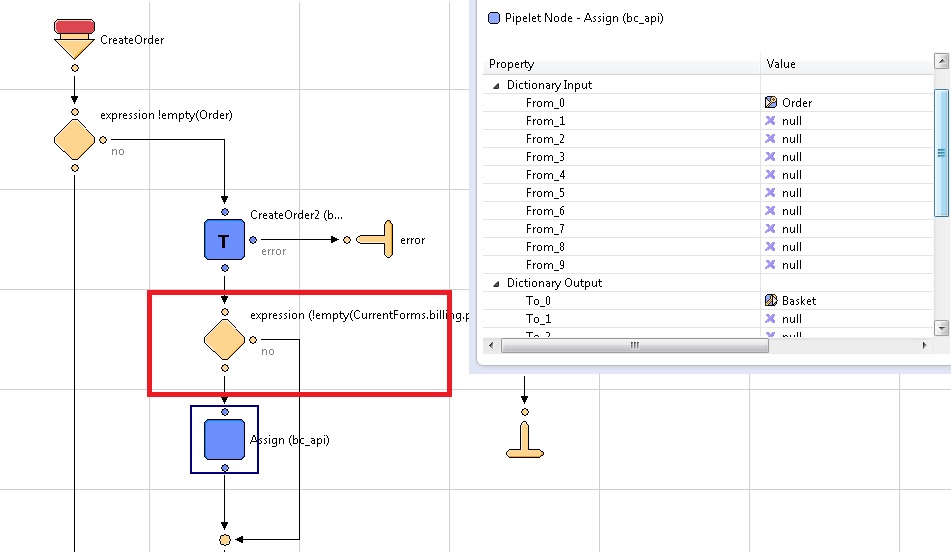




Note: Below mentioned change is required if merchant is using Paypal, Alipay payment methods along with Credit Card Payment methods. COPlaceOrder-CreateOrder node need to update as mentioned below

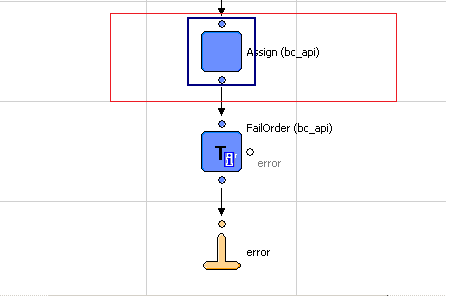
* Add below condition in the expression node to set the value of Order into Basket Object for Credit Card as payment methods.

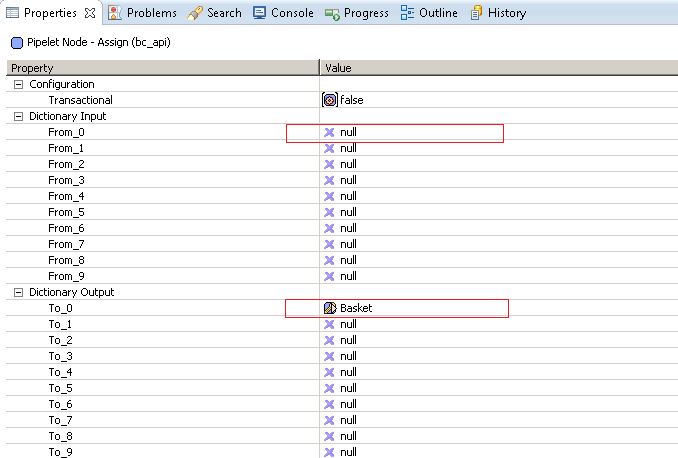
(!empty(CurrentForms.billing.paymentMethods.selectedPaymentMethodID) && (CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals('CREDIT\_CARD')))



* At the end of “Start” node Add assign node just before FailOrder pipelet and null the Basket instance created at the time of createOrder2 pipelet.

NOTE: Refer to the screen below for changes:



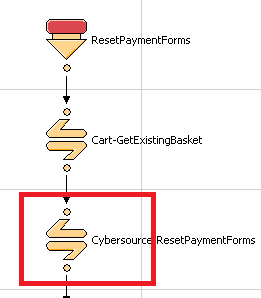


#### Pipeline - COBilling.xml

##### Update “resetPaymentForms()” function

Invoke Cybersource-ResetPaymentForms after Cart-GetExistingBasket

1. This pipeline is used to remove al l the payment intruments except the selected one.

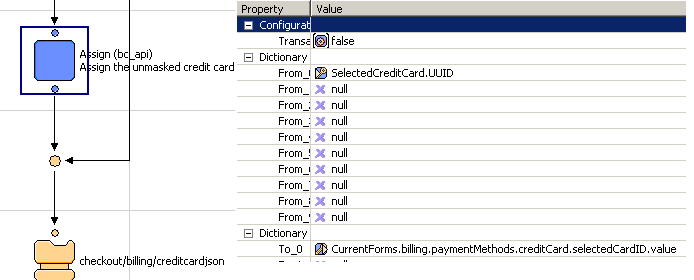


1. Delete pipelets called to remove payment instruments of BML or clear form elements of BML

##### Update “SelectCreditCard” node

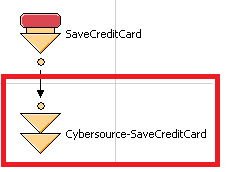
Add assign node just above interaction node checkout/billing/creditcardjson

|  |  |
| --- | --- |
| Output | Input |
| CurrentForms.billing.paymentMethods.creditCard.number.value | SelectedCreditCard.UUID |



##### Update “SaveCreditCard” Node

1. Remove existing logic of save credit card
2. Call Cybersource-SaveCreditCard



#### Replace DuplicateCheck.ds for duplicate card in My Account

Replace DuplicateCheck.ds with below code:

|  |
| --- |
| /\*\*  \* SetDefaultAddress.ds  \*  \* This script checks for a duplicate card number against the list  \* of payment instruments for the customer. If there is a duplicate,  \* return the payment instrument that already exist.  \*  \* @input NoIn : String  \* @input expMonth : Number  \* @input expYear : Number  \* @input cardType : String  \* @input PIList : dw.util.Collection  \* @output Duplicate : Boolean  \* @output OldCard : dw.order.PaymentInstrument  \*  \*/  **importPackage**( dw.system );  **function** execute( args : PipelineDictionary ) : Number  {  args.Duplicate = false;  **var** ccNumber = args.NoIn;  **for** **each**(**var** card in args.PIList) {  **var** creditcardNo = card.getCreditCardNumber();  **if** (card.creditCardExpirationMonth === args.expMonth && card.creditCardExpirationYear === args.expYear  && card.creditCardType === args.cardType && creditcardNo.substring(creditcardNo.length-4).equals(ccNumber.substring(ccNumber.length-4))) {  args.Duplicate = true;  args.OldCard = card;  **break**;  }  }    **return** PIPELET\_NEXT;  } |

#### Controller - COPlaceOrder.js

##### Update “start” node

Remove call node CSRF-Validate after place order

##### Update “handlePayments” Function

1. This function use for invoke payment processor Authorize function and check the result
2. Check for the result of authorization as failed
3. Return authorization result rather than empty when there is no error occurred

|  |
| --- |
| **function** handlePayments(order) {  **var** authorizationResult ={};  **if** (order.getTotalNetPrice() !== 0.00) {  **var** paymentInstruments = order.getPaymentInstruments();  **if** (paymentInstruments.length === 0) {  **return** {  missingPaymentInfo: **true**  };  }  /\*\*  \* Sets the transaction ID for the payment instrument.  \*/  **var** handlePaymentTransaction = **function** () {  paymentInstrument.getPaymentTransaction().setTransactionID(order.getOrderNo());  };  **for** (**var** i = 0; i < paymentInstruments.length; i++) {  **var** paymentInstrument = paymentInstruments[i];  **if** (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor() === **null**) {  Transaction.wrap(handlePaymentTransaction);  } **else** {  authorizationResult = PaymentProcessor.authorize(order, paymentInstrument);  **if** (authorizationResult.not\_supported || authorizationResult.error || authorizationResult.failed) {  **return** {  error: **true**  };  } else if(authorizationResult.returnToPage){  return {  returnToPage :true,  order : order  };  }  }  }  }  **return** authorizationResult;  } |

##### Update “start” function to handle payment results

Add below snippet to handle payment different results

|  |
| --- |
| var handlePaymentsResult = handlePayments(order);  if (handlePaymentsResult.error) {  session.custom.SkipTaxCalculation=false;  return Transaction.wrap(function () {  OrderMgr.failOrder(order);  return {  error: true,  PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')  };  }); |

|  |
| --- |
| } else if (handlePaymentsResult.declined) {  session.custom.SkipTaxCalculation=false;  return Transaction.wrap(function () {  OrderMgr.failOrder(order);  return {  error: true,  PlaceOrderError: new Status(Status.ERROR, 'confirm.error.declined')  };  });  } else if (handlePaymentsResult.end) {  return {};  } else if (handlePaymentsResult.review) {  ReviewOrder({Order:order});  return {};  } else if (handlePaymentsResult.pending) {  ReviewOrder({Order:order});  return {};  }  var orderPlacementStatus = Order.submit(order);  if (!orderPlacementStatus.error) {  clearForms();  }  return orderPlacementStatus;  } |

##### Update “clearforms” function

Add below snippet at end of function

|  |
| --- |
| session.custom.cartStateString=null; |

##### Add” fail” function

1. Add new method to handle the failed order

|  |
| --- |
| /\*  \* Identifies if an order exists, submits the order, and shows a confirmation message.  \*/  **function** fail(args) {  **var** Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');  **var** orderResult = Cybersource.GetOrder({Order:args.Order});  **if** (orderResult.error) {  app.getController('COSummary').Start({PlaceOrderError:orderResult.PlaceOrderError});  **return**;  }  **var** order = orderResult.Order;  **var** PlaceOrderError = args.PlaceOrderError!= **null** ? args.PlaceOrderError : **new** dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined");  session.custom.SkipTaxCalculation=**false**;  **var** failResult = Transaction.wrap(**function** () {  OrderMgr.failOrder(order);  **return** {  error: **true**,  PlaceOrderError: PlaceOrderError  };  });  **if** (failResult.error){  app.getController('COSummary').Start({PlaceOrderError:failResult.PlaceOrderError});  **return**;  }  **return**;  } |

##### Add “ReviewOrder” function

Add below lines at script module section

|  |
| --- |
| var Email = app.getModel('Email');  var Resource = require('dw/web/Resource'); |

Add the review order function with the code snippet below

|  |
| --- |
| /\*\*  \* Leave order in created state in demandware and send order confirmation email  \* **@param** args  \*/  **function** ReviewOrder(args) {  **var** order = args.Order;  // Send order confirmation and clear used forms within the checkout process.  Email.get('mail/orderconfirmation', order.getCustomerEmail())  .setSubject((Resource.msg('order.orderconfirmation-email.001', 'order', **null**) + ' ' + order.getOrderNo()).toString())  .send({  Order: order  });  // Clears all forms used in the checkout process.  clearForms();  app.getController('COSummary').ShowConfirmation(order);  **return**;  } |

##### Add “submitOrder” function

Add the submit order function with the code snippet below

|  |
| --- |
| /\*\*  \* Submit the order and send order confirmation email  \* **@param** args  \*/  **function** SubmitOrder(args) {  **var** orderPlacementStatus = Order.submit(args.Order);  **if** (!orderPlacementStatus.error) {  clearForms();  app.getController('COSummary').ShowConfirmation(args.Order);  **return**;  }    app.getController('COSummary').Start();  } |

##### Update “submit” function

Replace the submit function with the code snippet below

|  |
| --- |
| /\*  \* Asynchronous Callbacks for SiteGenesis.  \* Identifies if an order exists, submits the order, and shows a confirmation message.  \*/  **function** submit(args) {  **var** Provider = require('int\_cybersource\_controllers/cartridge/controllers/Provider');  **var** providerParam = request.httpParameterMap.provider.stringValue;  **if**(!empty(providerParam)) {  **return** Provider.Check(args);  }  app.getController('Cart').Show();  **return**;  } |

##### Export functions

|  |
| --- |
| exports.Fail = guard.ensure(['https'], fail);  exports.ReviewOrder = ReviewOrder;  exports.SubmitOrder = SubmitOrder; |

#### Controller - COBilling.js

##### Update Export Function

|  |
| --- |
| exports.ReturnToForm = guard.ensure(['https'], returnToForm);  exports.handleBillingAddress = handleBillingAddress;  exports.initAddressForm = initAddressForm;  exports.initEmailAddress = initEmailAddress;  exports.validateBilling = validateBilling; |

##### Update “resetPaymentForms()” function

This fuction invoke cybersource cartridge “ResetPaymentForms” function after cart basket retrieved.

Remove BML payment instruments from paypal and credit card condition

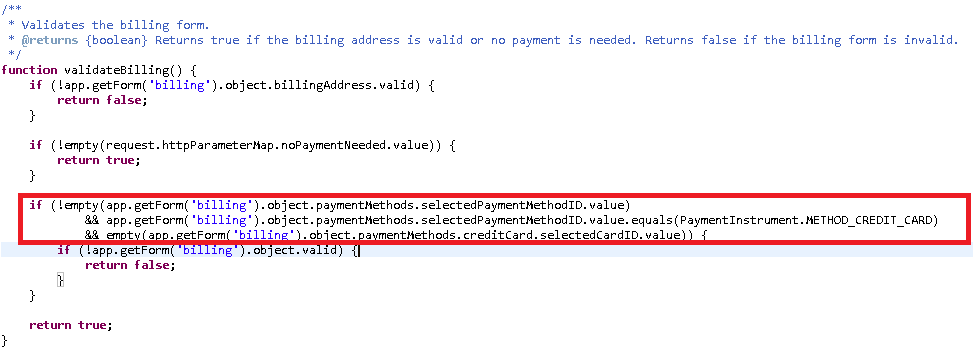
Add if condition after cart object

|  |
| --- |
| /\*\*  \* Clears the form element for the currently selected payment method and removes the other payment methods.  \*  \* **@return** {Boolean} Returns true if payment is successfully reset. Returns false if the currently selected payment  \* method is bml and the ssn cannot be validated.  \*/  **function** resetPaymentForms() {  **var** cart = app.getModel('Cart').get();  **if** (**null** != cart && !empty(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID)) {  **var** Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');  Cybersource.ResetPaymentForms({Basket:cart.object, PaymentType: app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value});  }  **var** status = Transaction.wrap(**function** () {  **if** (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals('PayPal')) {  app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD));  } **else** **if** (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_CREDIT\_CARD)) {  cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));  } **else** **if** (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_BML)) {  app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();  **if** (!app.getForm('billing').object.paymentMethods.bml.ssn.valid) {  **return** **false**;  }  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD));  }  **return** **true**;  });  **return** status;  } |

##### Update the validateBilling() function

Update the if condition Code add highlighted section

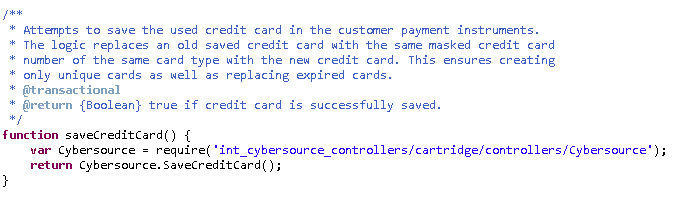
|  |
| --- |
| **if** (!empty(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value)  && app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_CREDIT\_CARD)  && empty(app.getForm('billing').object.paymentMethods.creditCard.selectedCardID.value)) {  **if** (!app.getForm('billing').object.valid) {  **return** **false**;  }  } |



##### Update “saveCreditCard” function

Replace the entire function with the below snippet,.

|  |
| --- |
| **function** saveCreditCard() {  **var** Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');  **return** Cybersource.SaveCreditCard();  } |



##### Update “selectCreditCard” function

Add below code snippet inside the condition for selectedCreditCard to update selectedcarduuid in Credit card

|  |
| --- |
| **if** (selectedCreditCard) {  app.getForm('billing').object.paymentMethods.creditCard.number.value = selectedCreditCard.getCreditCardNumber();  app.getForm('billing').object.paymentMethods.creditCard.selectedCardID.value = selectedCreditCard.UUID;  } |

#### JS file – billing.js [compiled to app.js]

##### Update “populateCreditCardForm” function

Add new parameter “selectedPaymentMethod” and add Switch condition to handle Credit card related APM’s as below:

[Note: all app.js changes are similar to billing.js, please refer the below section for app.js changes]

|  |
| --- |
| **function** populateCreditCardForm(cardID,selectedPaymentMethod) {  // load card details  **var** url = util.appendParamToURL(Urls.billingSelectCC, 'creditCardUUID', cardID);  ajax.getJson({  url: url,  callback: **function** 0028data) {  **if** (!data) {  window.alert(Resources.CC\_LOAD\_ERROR);  return false;  }  **switch** (selectedPaymentMethod) {  **case** "SA\_REDIRECT":  $('.payment-method-expanded .saCCToken .field-wrapper').val(data.selectedCardID); $("#dwfrm\_billing\_paymentMethods\_creditCard\_selectedCardID").val(data.selectedCardID);  **break**;  **case** "SA\_IFRAME":  $('.payment-method-expanded .saIframeCCToken .field-wrapper').val(data.selectedCardID); $("#dwfrm\_billing\_paymentMethods\_creditCard\_selectedCardID").val(data.selectedCardID);  **break**;  **case** "CREDIT\_CARD":  setCCFields(data);  **break**;  default:  setCCFields(data);  }  }  });  } |

##### Update “creditCardList” on change function

* Update the method by adding parameter “selectedPaymentMethod”:

|  |
| --- |
| // select credit card from list  $('#creditCardList').on('change', **function** () {  **var** cardUUID = $(**this**).val();  **if** (!cardUUID) {$($checkoutForm).find('input[name$="\_selectedCardID"]').val(''); **return**;}  populateCreditCardForm(cardUUID,selectedPaymentMethod);  // remove server side error  $('.required.error').removeClass('error');  $('.error-message').remove();  }); |

##### Update “setCCFields “function

* Add highlighted lines as shown in function

|  |
| --- |
| **function** setCCFields(data) {  **var** $creditCard = $('[data-method="CREDIT\_CARD"]');  $creditCard.find('input[name$="creditCard\_owner"]').val(data.holder).trigger('change');  $creditCard.find('select[name$="\_type"]').val(data.type).trigger('change');  $creditCard.find('input[name\*="\_creditCard\_number"]').val(data.maskedNumber).trigger('change');  **var** selectedPaymentMethodID = $('input[name$="\_selectedPaymentMethodID"]:checked').val();  **if**(selectedPaymentMethodID == 'SA\_SILENTPOST'){  $creditCard.find('[name$="\_month"]').val(data.expirationMonth);  $creditCard.find('[name$="\_year"]').val(data.expirationYear);  }  **else**{  $creditCard.find('[name$="\_month"]').val(data.expirationMonth).trigger('change');  $creditCard.find('[name$="\_year"]').val(data.expirationYear).trigger('change');  }  $creditCard.find('input[name$="\_cvn"]').val('').trigger('change');  $creditCard.find('[name$="creditCard\_selectedCardID"]').val(data.selectedCardID).trigger('change');  $creditCard.find("input[name$='\_cvn']").val('');  } |

##### Update “updatePaymentMethod “function

* Add highlighted lines at end of function before validateform

|  |
| --- |
| **function** updatePaymentMethod(paymentMethodID) {  **var** $paymentMethods = $('.payment-method');  $paymentMethods.removeClass('payment-method-expanded');  **var** dataMethod = paymentMethodID;  **if** (paymentMethodID=='SA\_SILENTPOST') {  dataMethod = 'CREDIT\_CARD';  }  **var** $selectedPaymentMethod = $paymentMethods.filter('[data-method="' + dataMethod + '"]');  **if** ($selectedPaymentMethod.length === 0) {  $selectedPaymentMethod = $('[data-method="Custom"]');  }  **if** (paymentMethodID=="CREDIT\_CARD" || paymentMethodID=="SA\_SILENTPOST") {  $(".spsavecard").show();  } **else** **if** ((paymentMethodID=="SA\_REDIRECT" || paymentMethodID=="SA\_IFRAME") && SitePreferences.TOKENIZATION\_ENABLED) {  $(".spsavecard").show();  }  **else** {  $(".spsavecard").hide();  }  $selectedPaymentMethod.addClass('payment-method-expanded');  // ensure checkbox of payment method is checked  $('input[name$="\_selectedPaymentMethodID"]').removeAttr('checked');  $('input[value=' + paymentMethodID + ']').prop('checked', 'checked');  formPrepare.validateForm();  } |

##### Update “export.init “function

* Remove creditCardList onchange section and updatePaymentMethod section
* Add below code snippet after formPrepare.init

|  |
| --- |
| formPrepare.init({  formSelector: 'form[id$="billing"]',  continueSelector: '[name$="billing\_save"]'  });  **var** $ccContainer = $($checkoutForm).find(".payment-method").filter(**function**(){  **return** $(**this**).data("method")=="CREDIT\_CARD";  });  **var** $ccNum = $ccContainer.find("input[name$='\_number']");  $($checkoutForm).find('input[name$="\_selectedCardID"]').val('');  $($checkoutForm).find('input[name\*="\_number"]').val('');  $ccContainer.find('input[name\*="\_number"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_selectedCardID"]').val('');  });  $ccContainer.find('input[name$="\_owner"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_selectedCardID"]').val('');  });  $ccContainer.find('select[name$="creditCard\_type"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_selectedCardID"]').val('');  });  $ccContainer.find('select[name\*="expiration"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_selectedCardID"]').val('');    **var** selectedPaymentMethodID = $('input[name$="\_selectedPaymentMethodID"]:checked').val();  **var** cardNumber = $($checkoutForm).find('input[name\*="\_number"]').val();  **if**(cardNumber.indexOf('\*\*\*\*') != -1 && selectedPaymentMethodID == 'SA\_SILENTPOST'){  $($checkoutForm).find('input[name\*="\_number"]').val('');  }    });  // default payment method to 'CREDIT\_CARD'  updatePaymentMethod((selectedPaymentMethod) ? selectedPaymentMethod : 'CREDIT\_CARD');  $selectPaymentMethod.on('click', 'input[type="radio"]', **function** () {  updatePaymentMethod($(**this**).val());  }); |

#### Form - creditcard.xml

1. Include the following form field after saveCard field in the form:

|  |
| --- |
| <!-- field for credit card subscription -->  <field formid="selectedCardID" type="string" /> |

1. Remove max-length="16" from credit card number field to allow cards numbers of varied length.

|  |
| --- |
| <field formid=*"number"* label=*"creditcard.number"* type=*"string"* mandatory=*"true"* masked=*"4"* max-length=*"16"* description=*"creditcard.numberexample"* binding=*"creditCardNumber"* missing-error=*"creditcard.numbermissingerror"* value-error=*"creditcard.numbervalueerror"*/> |

#### Form - customeraddress.xml

Include the following code just above the action events

|  |
| --- |
| <!-- email field is contained in separate form group to enable binding to customer profile -->  <group formid=*"email"*>  <field formid=*"*emailAddress*"* label=*"profile.email"* type=*"string"* mandatory=*"true"* regexp=*"^[\w.%+-]+@[\w.-]+\.[\w]{2,6}$"* binding=*"email"* max-length=*"50"* missing-error=*"forms.address.email.invalid"* range-error=*"forms.address.email.invalid"* parse-error=*"forms.address.email.invalid"* value-error=*"forms.address.email.invalid"*/>  </group>  <!-- all possible actions performed on an address -->  <action formid="cancel" valid-form="false"/>  <action formid="remove" valid-form="false"/> |

#### Form - paymentinstruments.xml

Include address fromId just below new credit card Id

|  |
| --- |
| <include formid="address" name="customeraddress"/> |

#### Template - creditcardjson.isml

Retrieve subscription token of saved card to be used further

|  |
| --- |
| expirationYear:pdict.SelectedCreditCard.creditCardExpirationYear,  selectedCardID:pdict.SelectedCreditCard.UUID |

#### Template - paymentmethods.isml

##### Remove Credit card and bml payment method section and include custom logic for fields

* Include the following code

|  |
| --- |
| <isscript>  var currentCountry = require('~/cartridge/scripts/util/Countries').getCurrent(pdict);  </isscript>  <isinclude template="custom/paymentmethods"/> |

#### Template - paymentinstrumentsdetail.isml

##### Remove attributes property fromcreditcardnumber field

|  |
| --- |
| <isinputfield formfield="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.number}" dynamicname="true" type="input" attributes="${numberAttributes}"/> |

#### Script - Resource.ds

##### Update ResourceHelper.getPreferences

|  |
| --- |
| ,TOKENIZATION\_ENABLED: (Site.getCurrent().getCustomPreferenceValue('CsTokenizationEnable') == 'YES')? true : false |

#### Script - ValidatePaymentInstruments.ds

##### Update the excute method to not display expired cards during checkout

// check, whether payment card is still applicable

|  |
| --- |
| if( card != null && contains(card, cards) && !pi.isCreditCardExpired())  {  continue;  } |

#### Controller- common.js

##### Update validatePaymentInstruments function

Update below if condition so that expired card is not shown in saved credit card list during checkout.

|  |
| --- |
| if (card !== null && cards.contains(card) && !pi.isCreditCardExpired()) { |

### Credit Card Auth

#### Hooks.json in Controller

##### Replace hook entry for CYBERSOURCE\_CREDIT

{

"name": "app.payment.processor.CYBERSOURCE\_CREDIT",

"script": "./../../../int\_cybersource\_controllers/cartridge/controllers/CYBERSOURCE\_CREDIT"

},

### Tax Service

#### Script - calculate.js

##### Avoid CalculateTax builtin Function Call for NET taxation

Move the built in tax calculation function call in cart/calculate.js inside condition below which is getting called from a custom hook in CalculateCart.ds.

|  |
| --- |
| **// ===================================================**  **// ===== CALCULATE TAX =====**  **// ===================================================**  **if** (dw.order.TaxMgr.taxationPolicy!=dw.order.TaxMgr.TAX\_POLICY\_NET) {  calculateTax(basket);  } |

#### Controller Cartridge – Script CartModel.js

##### Update Calculate function

Add condition for calling calculate taxes of cybersource after hook call

|  |
| --- |
| **if** (dw.order.TaxMgr.taxationPolicy==dw.order.TaxMgr.TAX\_POLICY\_NET) {  **var** Cybersource = require('int\_cybersource\_controllers/cartridbasicge/controllers/Cybersource');  Cybersource.CalculateTaxes({Basket:**this**.object});  }  },  addProductToCart: function() { |

#### Controller Cartridge – Script OrderModel.js

##### Update placeOrder function

1. Set variable session.custom.SkipTaxCalculation=false; before failOrder

|  |
| --- |
| **function** placeOrder(order) {  **var** placeOrderStatus = OrderMgr.placeOrder(order);  **if** (placeOrderStatus === Status.ERROR) {  session.custom.SkipTaxCalculation=**false**;  OrderMgr.failOrder(order);  **throw** **new** Error('Failed to place order.');  }  order.setConfirmationStatus(Order.CONFIRMATION\_STATUS\_CONFIRMED);  order.setExportStatus(Order.EXPORT\_STATUS\_READY);  } |

#### Controller -Cart.js

##### Update submitForm function

UpdatedeleteProct sdection by clear cartstatestring

|  |
| --- |
| 'deleteProduct': **function** (formgroup) {  Transaction.wrap(**function** () {  cart.removeProductLineItem(formgroup.getTriggeredAction().object);  session.custom.cartStateString = **null**;  });  **return** {  cart: cart  };  }, |

##### Controller – COShipping.js

###### Update “updateShippingMethodList“ function

1. Set session variable session.custom.SkipTaxCalculation=true;before cart calculate

|  |
| --- |
| applicableShippingMethods = cart.getApplicableShippingMethods(address);  shippingCosts = **new** HashMap();  currentShippingMethod = cart.getDefaultShipment().getShippingMethod() || ShippingMgr.getDefaultShippingMethod();    // Transaction controls are for fine tuning the performance of the data base interactions when calculating shipping methods  Transaction.begin();  **for** (i = 0; i < applicableShippingMethods.length; i++) {  method = applicableShippingMethods[i];  cart.updateShipmentShippingMethod(cart.getDefaultShipment().getID(), method.getID(), method, applicableShippingMethods);  session.custom.SkipTaxCalculation=true;  cart.calculate();  shippingCosts.put(method.getID(), cart.preCalculateShipping(method));  } |

##### Controller – COPlaceOrder.js

###### Update “start” function

1. Set session variable SkipTaxCalculation set as false in payment ERROR scenarios

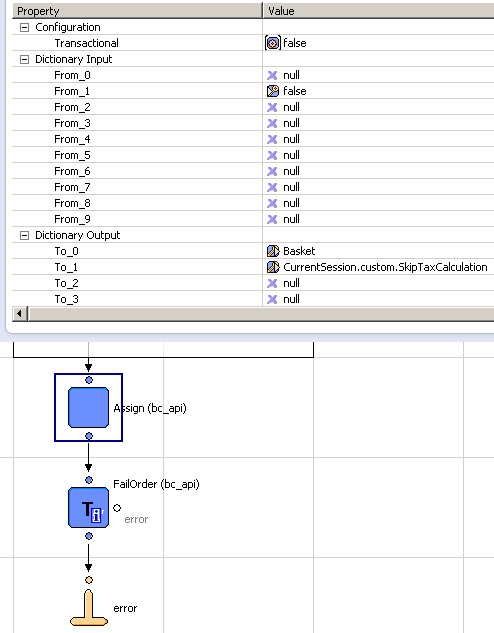
session.custom.SkipTaxCalculation=false;



#### Pipeline –COPlaceOrder.xml

##### Update Start Node

1. Add assign node before FailOrder pipelet where set variable in session SkipTaxCalculation as false and Basket as null



#### Pipeline – COShipping.xml

##### Update UpdateShippingMethodList node

Add assign node before Cart-Calculate in the COShipping-UpdateShippingMethodList pipeline. Refer to the following screenshot:

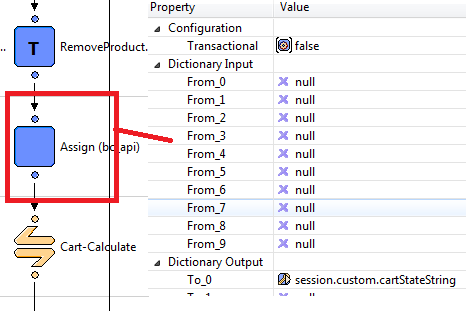




#### Pipeline –Cart.xml

##### Update deleteProduct transition flow

1. Add assign node with session.custom.cartStateString as null

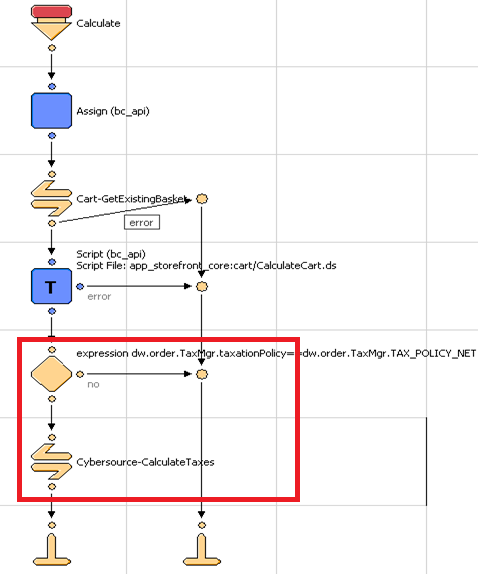


##### Update Cart-Calculate Node

1. Add condition after script call cart/calculateCart.ds script this will determine site taxation policy is NET in Demandware

dw.order.TaxMgr.taxationPolicy==dw.order.TaxMgr.TAX\_POLICY\_NET

1. On condition met add Cybersource-CalculateTaxes call node



### Address Verification Service

Provide Site Preference values for 2 AVS-related business rules:

User can change the site preference value by following [Merchant Tools](https://cybersource04.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/ViewApplication-SelectSite?MenuGroupID=ChannelMenu&ChannelID=bcbcIiaagtq3oaaac631602PJ3&SelectedSiteID=bcbcIiaagtq3oaaac631602PJ3) >  [Site Preferences](https://cybersource04.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/SiteNavigationBar-ShowMenuitemOverview?CurrentMenuItemId=site-prefs) >  [Custom Site Preferences](https://cybersource04.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/CustomPreferences-View?PreferenceType=SITE) > Cybersource path for a selected site as shown below.

CsAvsIgnoreResult – Determines whether AVS failures will force an auth failure.

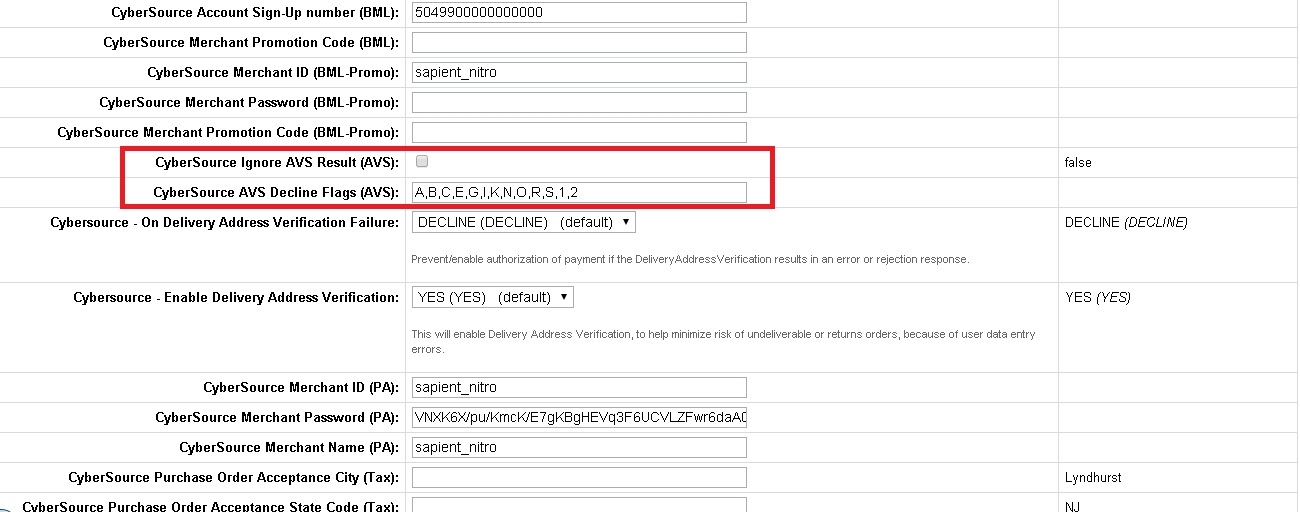
Default value would be false and if user checks this checkbox then in case of address verification failure corresponding to AVS decline flags, order will be placed but considering the default value, in case of address verification failure corresponding to decline flags application will not allow user to place the order.

CsAvsDeclineFlags –Determines how “correct” an address must be to produce a failure result

Augment UI interaction nodes to deal with AVS failure or correction confirmation dialogs, wherever Payment Authorization takes place, typically within the COPlaceOrder-Start and COSummary-Submit pipelines.

Merchant can define the value of decline flags in the business manager Cybersource site preference and when address verification service is enabled and while placing the order if that service returns any of the flag mentioned in site preference, system will decline the order.

Screen shot to change the site preference value:



### Delivery Address Validation Service

Provide Site Preference values for 2 DAV-related business rules:

User can change the site preference value by following [Merchant Tools](https://cybersource04.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/ViewApplication-SelectSite?MenuGroupID=ChannelMenu&ChannelID=bcbcIiaagtq3oaaac631602PJ3&SelectedSiteID=bcbcIiaagtq3oaaac631602PJ3) >  [Site Preferences](https://cybersource04.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/SiteNavigationBar-ShowMenuitemOverview?CurrentMenuItemId=site-prefs) >  [Custom Site Preferences](https://cybersource04.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/CustomPreferences-View?PreferenceType=SITE) > Cybersource path for a selected site as shown below.

CsDavEnable – Determines whether DAV features are enabled for payment auth requests.

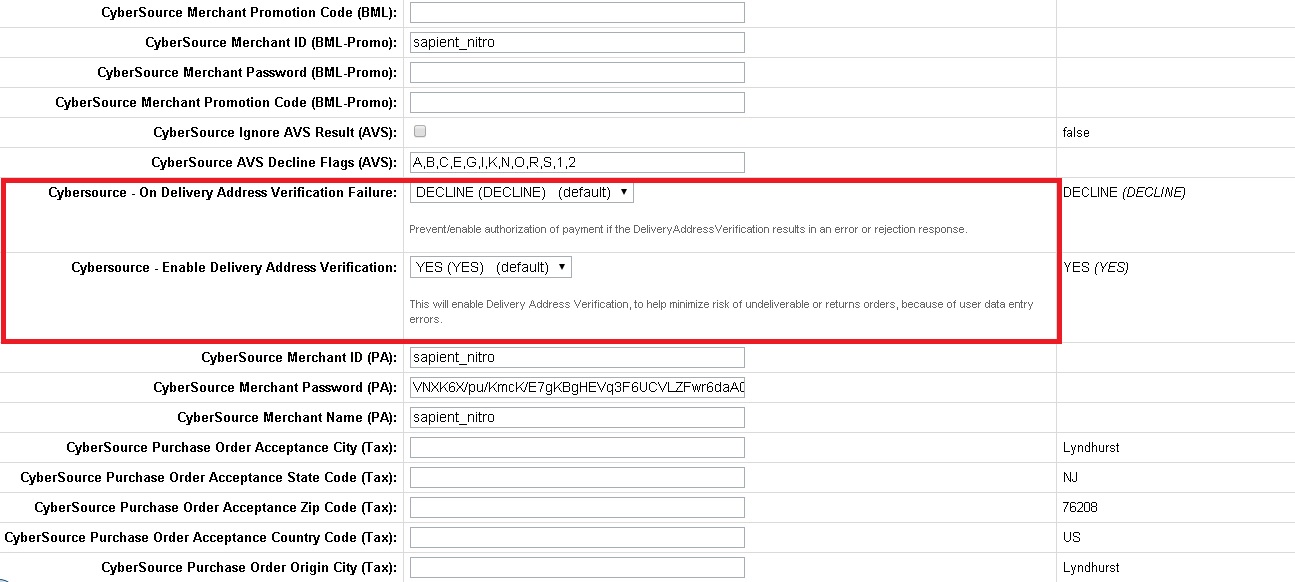
Default value would be DECLINE and if user selects APPROVE from dropdown then in case of shipping or delivery address validation failure corresponding to enable delivery address verification value mentioned below, order will be placed but considering the default value i.e. DECLINE, in case of shipping or delivery address validation failure corresponding to enable delivery address verification value, application will not allow user to place the order. This will Prevent/enable authorization of payment if the DeliveryAddressVerification results in an error or rejection response.

CsDavOnAddressVerificationFailure –Determines whether a DAV failure will result in a payment auth failure

Merchant can set the value of this field in the business manager Cybersource site preference. This will enable Delivery Address Verification, to help minimize risk of undeliverable or returns orders, because of user data entry errors. When user selects YES from the drop down and corresponding CsDavEnable site preference value is DECLINE and in case of delivery address verification failure, system will not allow process the order.

Augment UI interaction nodes to deal with AVS failure or correction confirmation dialogs, wherever Payment Authorization takes place, typically within the COPlaceOrder-Start and COSummary-Submit pipelines.

Screen shot to change the site preference value:



### Full Authorization Reversal

Full Authorization reversal is created and working in stand-alone mode in Cybersource\_Services.xml pipeline/controller. It has to customized and integrated as per the merchant specific needs.

### Payer Authentication Service

##### Configure Site preference

Provide Site Preference values for 5 Payment Authorization related business rules:

CyberSource Merchant ID (PA):– Determines which Cybersource merchant id to be used for payer authentication. It can be same as default merchant id as well.

CyberSource Merchant Password (PA): –Password corresponding to the merchant account.

CyberSource Merchant Name (PA): –Merchant Name to be used for service.

CyberSource Save Proof.xml (PA): –Determines whether to save proof.xml (received from Cybersource response) as part of order object.

CyberSource Save ParesStatus(PA): –Determines whether to pass ParesStatus received as input parameter from Pa authorization request as input parameter to ccAuthorization request.

### Payment Tokenization Service

#### My Account - Template - paymentinstrumentdetails.isml

##### Display Subscription Error Message

Include the following code block just after the <h1> tag to display the title message

|  |
| --- |
| <isif condition=*"${pdict.SubscriptionError != null}"*>  <div class=*"error-form"*>  ${Resource.msg('account.subscription','account',null)}  </div>  </isif> |

##### Add Billing Address Fields

Include the below code right after <isdynamicform> form object

|  |
| --- |
| <!-- code comments for adding new billing fields..-->  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.firstname}"* type=*"input"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.lastname}"* type=*"input"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.address1}"* type=*"input"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.address2}"* type=*"input"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.country}"* type=*"select"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.states.state}"* type=*"select"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.city}"* type=*"input"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.postal}"* type=*"input"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.phone}"* type=*"input"*/>  <isinputfield formfield=*"${pdict.CurrentForms.paymentinstruments.creditcards.address.email.emailAddress}"* xhtmlclass=*"email"* type=*"input"*/>  <!-- end code changes for billing fields. --> |

#### My Account - Template - paymentinstrumentlist.isml

##### Display Subscription Error

Add below code just after <h1> tag to show delete subscription message

|  |
| --- |
| <isif condition=*"${pdict.SubscriptionError != null}"*>  <div class=*"error-form"*>  ${Resource.msg('paymentinstrumentlist.deletesubscription','account',null)}  </div>  </isif> |

#### My Account - Properties File - account.properties

Add below resource bundle properties at end of account.properties file to display error messages

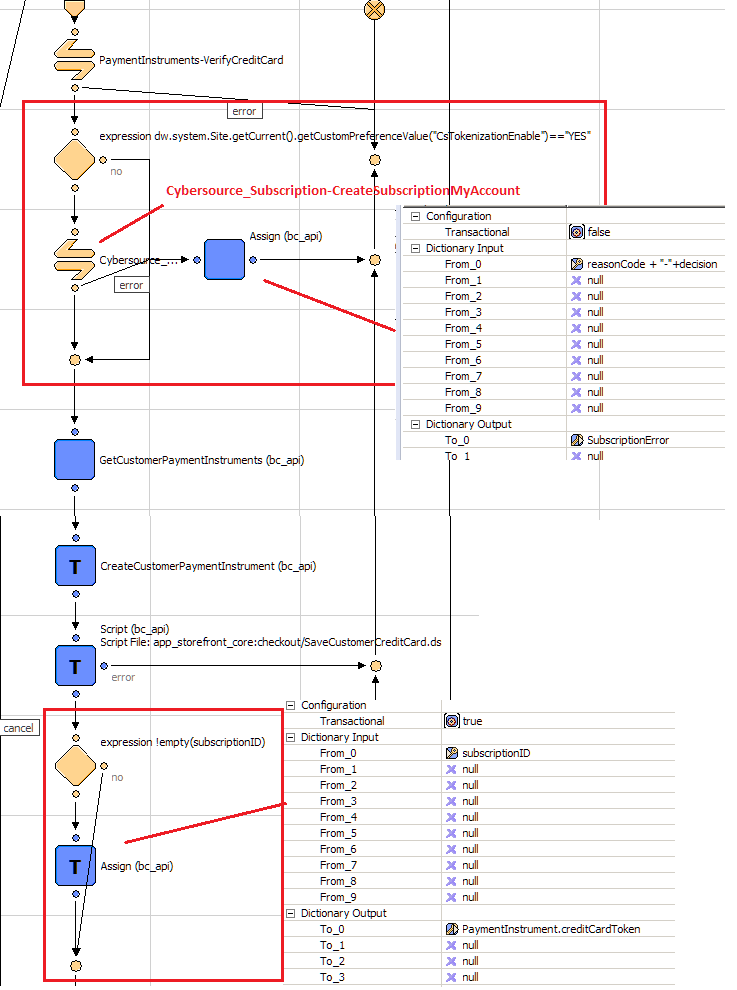
|  |
| --- |
| ##############################################  # cybersource  ##############################################  paymentinstrumentlist.deletesubscription=An error occurred while deleting subscription.  account.subscription = Subscription is not created. Please check your card details. |

#### My Account - Pipeline – PaymentInstruments.xml

##### Update “Add” node

1. Add a Decision node as **dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable")=="YES"** just after PaymentInstruments-VerifyCreditCard to check whether tokenization is enabled from BM
   1. If tokenization is enabled , a call to Cybersource\_Subscription-CreateSubscriptionMyAccount will be made for subscription creation
   2. If not, continue with the next step without Subscription creation
2. Add and assign node to trace the error in “SubscriptionError” in case of failure for Subscription Creation with value as **reasonCode+”-”+decision**
3. Add decision node **as !empty(subscriptionID)** and just after SaveCustomerCreditCard.ds and for existing Subscription ID
   1. In Case Subscription Id is not empty , Add Transactional assign node and set the value of “subscriptionID” into “PaymentInstrument.creditCardToken”

[Note]: Please refer below screen shot for all steps mentioned above



1. Further update the existing script pipelet of DuplicateCheck.ds with below I/O parameters:

The DuplicateCheck.ds now existing in cybersource cartridge use “utils/DuplicateCheck.ds”

Input

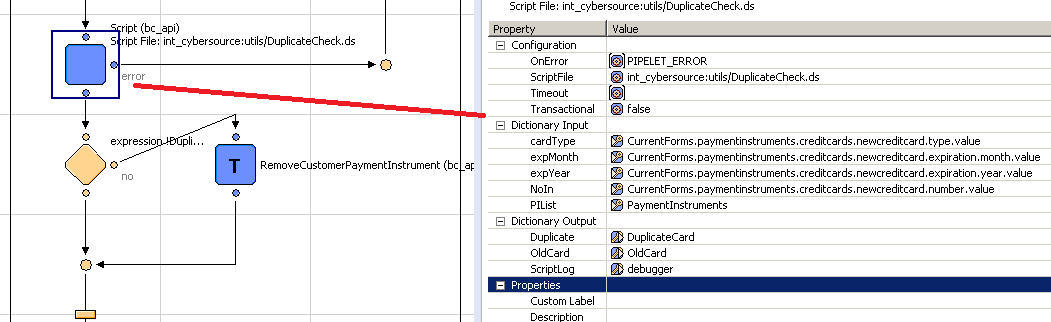
|  |  |
| --- | --- |
| cardType | CurrentForms.paymentinstruments.creditcards.newcreditcard.type.value |
| expMonth | CurrentForms.paymentinstruments.creditcards.newcreditcard.expiration.month.value |
| expYear | CurrentForms.paymentinstruments.creditcards.newcreditcard.expiration.year.value |
| NoIn | CurrentForms.paymentinstruments.creditcards.newcreditcard.number.value |
| PIList | PaymentInstruments |

Output:

|  |  |
| --- | --- |
| Duplicate | DuplicateCard |
| OldCard | OldCard |

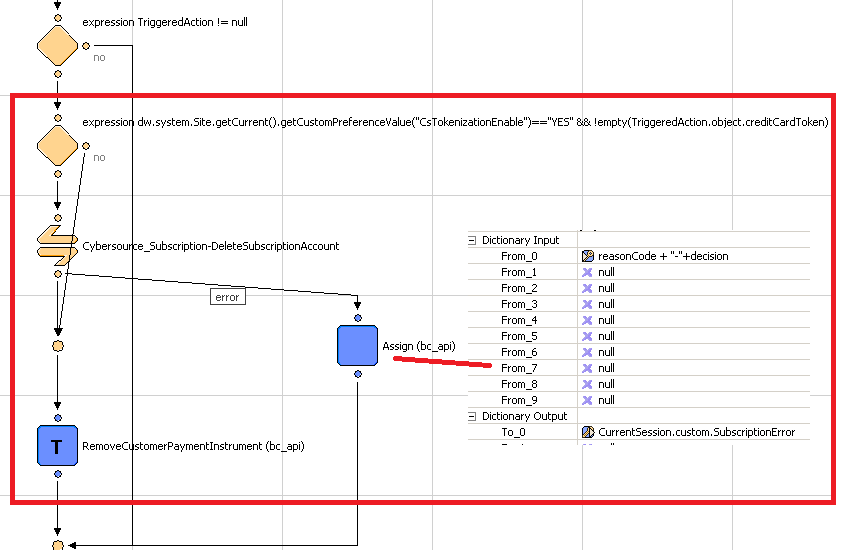
Continue with the existing flow further.

Refer below:



##### Update “Delete” node

1. Add a Decision node as dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable")=="YES" && !empty(TriggeredAction.object.creditCardToken) just after TriggeredAction to check whether tokenization is enabled from BM
2. Call Cybersource\_Subscription-DeleteSubscriptionAccount
3. On Error Add and assign node and set the input as **reasonCode + "-"+decision** and set the corresponding output as **CurrentSession.custom.SubscriptionError**
4. Join to PaymentInstruments-List on error



All functionalities related to Cybersource Payment Tokenization are created and working in stand-alone mode in Cybersource\_Subscription.xml pipeline. They have to customized and integrated as per the merchant specific needs

#### My Account - Controller - PaymentInstruments.js

##### Update “list” function

Update as below to handle SubscriptionError

|  |
| --- |
| **function** list() {  **var** SubscriptionError=**null**;  **var** wallet = customer.getProfile().getWallet();  **var** paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  **var** pageMeta = require('~/cartridge/scripts/meta');  **var** paymentForm = app.getForm('paymentinstruments');  paymentForm.clear();  paymentForm.get('creditcards.storedcards').copyFrom(paymentInstruments);  pageMeta.update(dw.content.ContentMgr.getContent('myaccount-paymentsettings'));  **if** (('SubscriptionError' **in** session.custom) && !empty(session.custom.SubscriptionError)) {  SubscriptionError = session.custom.SubscriptionError;  session.custom.SubscriptionError = **null**;  }  app.getView({  PaymentInstruments: paymentInstruments,  SubscriptionError : SubscriptionError  }).render('account/payment/paymentinstrumentlist');  **return**;  } |

##### Update “Add” function

Update as below to handle SubscriptionError

|  |
| --- |
| **function** add(clearForm, subscriptionError) {  **var** paymentForm = app.getForm('paymentinstruments');  **if** (clearForm !== **false**) {  paymentForm.clear();  }  paymentForm.get('creditcards.newcreditcard.type').setOptions(dw.order.PaymentMgr  .getPaymentMethod(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD).activePaymentCards.iterator());  app.getView({  ContinueURL: URLUtils.https('PaymentInstruments-PaymentForm'),  SubscriptionError: subscriptionError  }).render('account/payment/paymentinstrumentdetails');  } |

##### Update “handlePaymentForm” function

Update this code with **if** (!create()) {add(**false**); with below code to handle SubscriptionError

|  |
| --- |
| **function** handlePaymentForm() {  **var** paymentForm = app.getForm('paymentinstruments');  paymentForm.handleAction({  create: **function** () {  **var** createResult = create();  **if** (createResult.error) {  add(**false**, createResult.SubscriptionError);  **return**;  } **else** {  response.redirect(URLUtils.https('PaymentInstruments-List'));  }  },  error: **function** () {  add(**false**);  }  });  } |

##### Update “create” function

Update create function with below changes done for subscription and error handling

|  |
| --- |
| **function** create() {  **var** SubscriptionError;  **if** (!verifyCreditCard()) {  **return** {  error: **true**,  SubscriptionError: SubscriptionError  };  }  **var** subscriptionID;  **var** enableTokenization : String = dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable");  **if** (enableTokenization=='YES') {  **var** Cybersource\_Subscription = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource\_Subscription')  **var** createSubscriptionMyAccountResult = Cybersource\_Subscription.CreateSubscriptionMyAccount();  **if** (createSubscriptionMyAccountResult.error) {  SubscriptionError = createSubscriptionMyAccountResult.reasonCode + "-" + createSubscriptionMyAccountResult.decision;  **return** {  error: **true**,  SubscriptionError: SubscriptionError  };  }  subscriptionID = createSubscriptionMyAccountResult.subscriptionID;  }  **var** paymentForm = app.getForm('paymentinstruments');  **var** newCreditCardForm = paymentForm.get('creditcards.newcreditcard');  **var** ccNumber = newCreditCardForm.get('number').value();  **var** wallet = customer.getProfile().getWallet();  **var** paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  Transaction.begin();  **var** paymentInstrument = wallet.createPaymentInstrument(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  **try** {  save({  PaymentInstrument: paymentInstrument,  CreditCardFormFields: newCreditCardForm.object  });  } **catch** (err) {  Transaction.rollback();  **return** {  error: **true**,  SubscriptionError: SubscriptionError  };  }  **if** (!empty(subscriptionID)) {  paymentInstrument.setCreditCardToken(subscriptionID);  }  **var** isDuplicateCard = **false**;  **var** oldCard;  **for** (**var** i = 0; i < paymentInstruments.length; i++) {  **var** card = paymentInstruments[i];  **if** (card.creditCardExpirationMonth === newCreditCardForm.get('expiration.month').value() && card.creditCardExpirationYear === newCreditCardForm.get('expiration.year').value()  && card.creditCardType === newCreditCardForm.get('type').value() && card.getCreditCardNumber().indexOf(ccNumber.substring(ccNumber.length-4))) {  isDuplicateCard = **true**;  oldCard = card;  **break**;  }  }  **if** (isDuplicateCard) {  wallet.removePaymentInstrument(oldCard);  }  Transaction.commit();  paymentForm.clear();  **return** {  success: **true**  };  } |

##### Update “Delete” function

Update remove action handling to have deletion of subscription handling as per code snippet below

|  |
| --- |
| **function** Delete() {  **var** paymentForm = app.getForm('paymentinstruments');  **var** SubscriptionError;  paymentForm.handleAction({  remove: **function** (formGroup, action) {  **var** enableTokenization : String = dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable");  if (enableTokenization=='YES' && !empty(action.object.creditCardToken)) {  **var** Cybersource\_Subscription = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource\_Subscription')  **var** deleteSubscriptionBillingResult = Cybersource\_Subscription.DeleteSubscriptionAccount();  **if** (deleteSubscriptionBillingResult.error) {  SubscriptionError = deleteSubscriptionBillingResult.reasonCode + "-" + deleteSubscriptionBillingResult.decision;  session.custom.SubscriptionError = SubscriptionError;  **return** {  error: **true**  };  }  }  Transaction.wrap(**function** () {  **var** wallet = customer.getProfile().getWallet();  wallet.removePaymentInstrument(action.object);  });  },  error: **function** () {  // @TODO When could this happen  }  });    **if** (empty(SubscriptionError)) {  response.redirect(URLUtils.https('PaymentInstruments-List'));  }  } |

### Alipay Authorization

#### ValidatePaymentInstruments.ds

* Add Import package specified below

importPackage( dw.web );

* Replace the GIFT\_CERTIFICATE payment instrument check

|  |
| --- |
| // ignore gift certificate payment instrument if(!PaymentInstrument.METHOD\_GIFT\_CERTIFICATE.equals(pi.paymentMethod) )  With below code:  if(PaymentInstrument.METHOD\_GIFT\_CERTIFICATE.equals(pi.paymentMethod) || Resource.msg("paymentmethod.Alipay", "checkout", null).equals(pi.paymentMethod) || Resource.msg("paymentmethod.Paypal", "checkout", null).equals(pi.paymentMethod)) |

#### checkout.properties

Add following values in checkout.properties file for alipay changes.

|  |
| --- |
| ##############################################  # alipay Checkout  ##############################################  alipaycheckout.domesticpaymenttype=APD  alipaycheckout.currency=CNY  alipaycheckout.internationalpaymenttype=APY  ##############################################  # payment methods  ##############################################  paymentmethod.Alipay= ALIPAY |

#### Controller Cartridge - hooks.json

##### ALIPAY payment processor hook addition

1. Add a hook for payment processor as CYBERSOURCE\_ALIPAY at the end of hooks.json in cartridge app\_storefront\_controllers

|  |
| --- |
| ,  {  "name": "app.payment.processor.CYBERSOURCE\_ALIPAY",  "script": "./../../../int\_cybersource\_controllers/cartridge/controllers/CYBERSOURCE\_ALIPAY"  } |

### PayPal Express Checkout [From Cart Page and Mini Cart]

#### ValidatePaymentInstruments.ds

* Add Import package specified below

importPackage( dw.web );

* Replace GIFT\_CERTIFICATE payment instrument check existing code

// ignore gift certificate payment instrument if(!PaymentInstrument.METHOD\_GIFT\_CERTIFICATE.equals(pi.paymentMethod) )

With below code:

if(PaymentInstrument.METHOD\_GIFT\_CERTIFICATE.equals(pi.paymentMethod) || Resource.msg("paymentmethod.Alipay", "checkout", null).equals(pi.paymentMethod) || Resource.msg("paymentmethod.Paypal", "checkout", null).equals(pi.paymentMethod))

#### Template - minicart.isml

To enable express checkout from minicart .isml, add the code snippet just after the below line:

<a class=*"button mini-cart-link-cart"* href=*"${URLUtils.https('Cart-Show')}"* title=*"${Resource.msg('minicart.viewcart.label','checkout',null)}"*>${Resource.msg('minicart.viewcart','checkout',null)}</a>

|  |
| --- |
| <form class=*"minicart-action-expresscheckout"* action=*"${URLUtils.https('Cart-MiniCartExpressCheckout')}"* method=*"post"* name=*"${pdict.CurrentForms.cart.dynamicHtmlName}"* id=*"checkout-form"*>  <fieldset>  <isif condition=*"${dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}"*>  <input type=*"image"* src=*"https://www.paypal.com/en\_US/i/btn/btn\_xpressCheckout.gif"* alt=*"Paypal Express"* name=*"${pdict.CurrentForms.cart.paypalExpressCheckout.htmlName}"* align=*"left"* style="margin-right:*7px*;"/>  </isif>  </fieldset>  </form> |

#### CSS - \_checkout.scss [compiled into style.css]

Add a style just in end of the file to add an Express checkout button on minicart

|  |
| --- |
| .minicart-action-expresscheckout{  margin:20px 37px;  } |

#### Template - minibillingInfo.isml

Update file  to conditionally show the edit button for Billing Address and Payment Method on Order Summary Screen. Add the highlighted code at two places just after **<h3 class="section-header">** as mentioned below.

|  |
| --- |
| <isif condition=*"${!empty(billingAddress)}"*>  <div class=*"mini-billing-address order-component-block"*>  <h3 class=*"section-header"*>  <isif condition=*"${( empty(pdict.CurrentHttpParameterMap.paypalOrigin.value) || pdict.CurrentHttpParameterMap.paypalOrigin.value.equals('cart')) && ( empty(pdict.paypalOrigin) || !(pdict.paypalOrigin.equals('cart')))}"*>  <a href=*"${URLUtils.https('COBilling-Start')}"* class=*"section-header-note"*>${Resource.msg('global.edit','locale',null)}</a>  </isif>  ${Resource.msg('minibillinginfo.billingaddress','checkout',null)}  </h3>  <div class=*"details"*>  <isminicheckout\_address p\_address=*"${billingAddress}"*/>  </div>  </div>  </isif> |
| <h3 class=*"section-header"*>  <isif condition=*"${( empty(pdict.CurrentHttpParameterMap.paypalOrigin.value) || pdict.CurrentHttpParameterMap.paypalOrigin.value.equals('cart')) && ( empty(pdict.paypalOrigin) || !(pdict.paypalOrigin.equals('cart')))}"*>  <a href=*"${URLUtils.https('COBilling-Start')}"* class=*"section-header-note"*>${Resource.msg('global.edit','locale',null)}</a>  </isif>  <isif condition=*"${loopstate.first}"*><span>${Resource.msg('minibillinginfo.paymentmethod','checkout',null)}</span></isif>  </h3> |

#### Template - minishipments.isml

Replace the code

<isset name=*"editUrl"* value=*"${URLUtils.https('COShipping-Start')}"* scope=*"page"*/>

<isif condition=*"${pdict.CurrentForms.multishipping.entered.value}"*>

<isset name=*"editUrl"* value=*"${URLUtils.https('COShippingMultiple-Start')}"* scope=*"page"*/>

</isif>

With below code to edit shipping addresses depends on the checkout scenario

|  |
| --- |
| <iscomment>the url to edit shipping addresses depends on the checkout scenario</iscomment>  <isif condition="${!empty(pdict.paypalOrigin) && (pdict.paypalOrigin.equals('cart'))}">  <isset name="editUrl" value="${URLUtils.https('COShipping-Start', 'paypalOrigin', 'cart')}" scope="page"/>  <isif condition="${pdict.CurrentForms.multishipping.entered.value}">  <isset name="editUrl" value="${URLUtils.https('COShippingMultiple-Start', 'paypalOrigin', 'cart')}" scope="page"/>  </isif>  <iselse>  <isset name="editUrl" value="${URLUtils.https('COShipping-Start')}" scope="page"/>  <isif condition="${pdict.CurrentForms.multishipping.entered.value}">  <isset name="editUrl" value="${URLUtils.https('COShippingMultiple-Start')}" scope="page"/>  </isif>  </isif> |

#### Template - paymentmethods.isml

Update the file to handle paypal set service error

|  |
| --- |
| <isif condition="${pdict.OrderTotal > 0}">  <fieldset> <legend>  ${Resource.msg('billing.paymentheader','checkout',null)}  <div class="dialog-required"> <span class="required-indicator">&#8226; <em>${Resource.msg('global.requiredfield','locale',null)}</em></span></div>  </legend>  <isif condition="${pdict.PaypalSetServiceError != null}">  <div class="error-form">${Resource.msg('confirm.error.declined','checkout',null)}</div>  </isif>  <div class="payment-method-options form-indent"> |

#### Template - cart.isml

Add below mentioned code to display the error on cart page while placing an order.

<

|  |
| --- |
| <isslot id=*"cart-banner"* description=*"Banner for Cart page"* context=*"global"* />  <isif condition=*"${pdict.PlaceOrderError != null}"*>  <div class=*"error-form"*>${Resource.msg(pdict.PlaceOrderError.code,'checkout',null)}</div>  </isif> |

Please refer the screen shot below.

Add below mentioned code at two places inside “enableCheckout” condition to display Paypal Express checkout button in cart page.

|  |
| --- |
| <isif condition=*"${enableCheckout}"*>  <button class=*"button-fancy-large"* type=*"submit"* value=*"${Resource.msg('global.checkout','locale',null)}"* name=*"${pdict.CurrentForms.cart.checkoutCart.htmlName}"*>  ${Resource.msg('global.checkout','locale',null)}  </button>  <isif condition=*"${dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}"*>  <input type=*"image"* src=*"https://www.paypal.com/en\_US/i/btn/btn\_xpressCheckout.gif"* alt=*"Paypal Express"* name=*"${pdict.CurrentForms.cart.paypalExpressCheckout.htmlName}"* align=*"left"* style="margin-right:*7px*;"/>  </isif>  <iselse/>  <button class=*"button-fancy-large"* disabled=*"disabled"* type=*"submit"* value=*"${Resource.msg('global.checkout','locale',null)}"* name=*"${pdict.CurrentForms.cart.checkoutCart.htmlName}"*>  ${Resource.msg('global.checkout','locale',null)}  </button>  </isif> |

#### Template - singleshipping.isml

Add below if condition to check if the customer comes from Paypal Origin

|  |
| --- |
| <iscomment>display select box with stored addresses if customer is authenticated and there are saved addresses</iscomment>  <isif condition=*"${empty(pdict.CurrentHttpParameterMap.paypalOrigin.value) || !(pdict.CurrentHttpParameterMap.paypalOrigin.value.equals('cart'))}"*>  <isif condition=*"${pdict.CurrentCustomer.authenticated && pdict.CurrentCustomer.profile.addressBook.addresses.size() > 0}"*>  <div class=*"select-address form-row"*>  <label for=*"${pdict.CurrentForms.singleshipping.addressList.htmlName}"*>  ${Resource.msg('global.selectaddressmessage','locale',null)}  </label>  <div class=*"field-wrapper"*>  <isaddressselectlist p\_listId=*"${pdict.CurrentForms.singleshipping.addressList.htmlName}"* p\_listaddresses=*"${productListAddresses}"* />  </div>  </div>  </isif>  </isif> |

#### Properties File - checkout.properties

Add following values in checkout.properties file for PayPal changes.

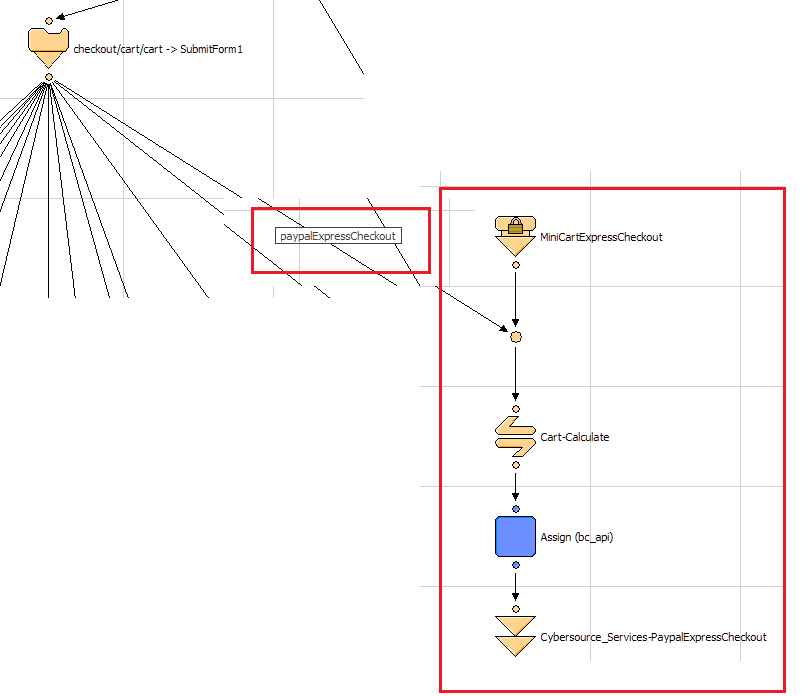
|  |
| --- |
| ##############################################  # payment methods  ##############################################  paymentmethod.Paypal =Paypal |

#### Pipeline - Cart.xml

Step to create flow for MiniCart express checkout from Cart page;

* 1. Add a new transition “**paypalExpressCheckout**” in Cart-Show pipeline to support express checkout from cart as well as mini cart page
  2. Add a new start node as “**MiniCartExpressCheckout**”, make it public and secured
  3. Add a call node” **Cart-Calculate**” to calculate cart items
  4. Add an assign node to set “**cart**” as “**paypalOrigin**”
  5. Further add jump node to call Cybersource\_Service-PaypalExpressCheckout

Below screen shot recognize the express checkout from cart page.



#### Controller -Cart.js

##### Update submitForm function

Add below code to add mini cart express checkout feature

|  |
| --- |
| 'error': function () {  return null;  },  'paypalExpressCheckout':function(){  Transaction.wrap(function() {  cart.calculate();  });  var Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource\_Services')  return Cybersource.PaypalExpressCheckout(cart.object);  }  });  if (formResult) {  cartAsset = app.getModel('Content').get('cart'); |

##### New miniCartExpressCheckout function

1. Write export for the new function as

exports.MiniCartExpressCheckout = guard.ensure(['https'],miniCartExpressCheckout);

1. Create a new miniCartExpressCheckout function in the Cart.js above Exposed methods

|  |
| --- |
| function miniCartExpressCheckout(){  var cart = app.getModel('Cart').get();  Transaction.wrap(function() {  cart.calculate();  });  var Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource\_Services')  return Cybersource.PaypalExpressCheckout(cart.object);    } |

#### hooks.json in Controller

##### Replace hook entry for PAYPAL\_EXPRESS

{

"name": "app.payment.processor.PAYPAL\_EXPRESS",

"script": "./../../../int\_cybersource\_controllers/cartridge/controllers/PAYPAL\_EXPRESS"

},

### PayPal Checkout [From Billing Page]

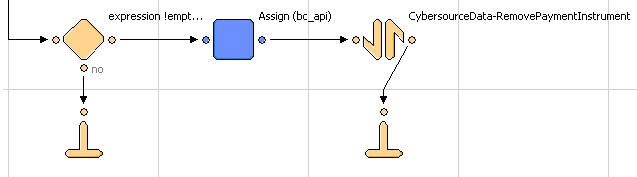
#### ValidatePaymentInstruments.ds

|  |
| --- |
| Add Import package specified below  importPackage( dw.web );  Replace the existing code  // ignore gift certificate payment instrument if(!PaymentInstrument.METHOD\_GIFT\_CERTIFICATE.equals(pi.paymentMethod) )  With below code:  if(PaymentInstrument.METHOD\_GIFT\_CERTIFICATE.equals(pi.paymentMethod) || Resource.msg("paymentmethod.Alipay", "checkout", null).equals(pi.paymentMethod) || Resource.msg("paymentmethod.Paypal", "checkout", null).equals(pi.paymentMethod)) |

#### Pipeline - COShipping.xml

##### Update “InitForms” Node

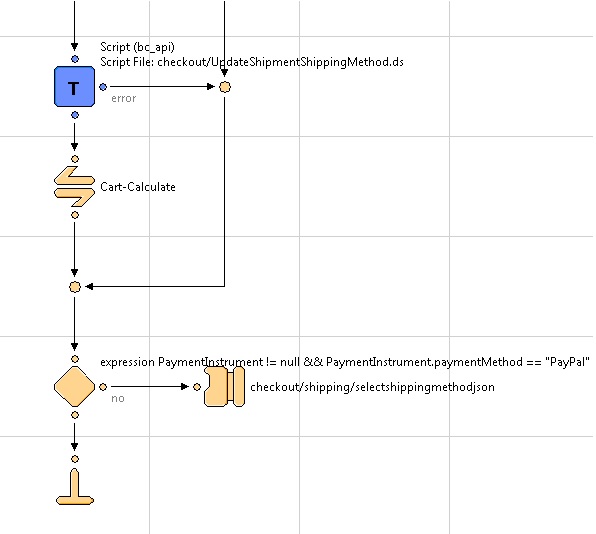
1. In COShipping pipeline also change the COShipping-InitForms at the bottom and add a condition for PayPal as !empty(Basket.getPaymentInstruments()) && !empty(Basket.getPaymentInstruments('PayPal'))
2. Assign node set variable “PaymentMethod” as “PayPal”
3. CallCybersourceData-RemovePaymentInstrument



##### Update “selectShippingMethod” Node

1. In COShipping pipeline also change the COShipping-SelectShippingMethod at the bottom and add a condition for PayPal to continue the checkout flow without using shipping method JSON object. Please refer the screen shot below for the changes in the pipeline.
2. Add below mentioned condition in the expression node.

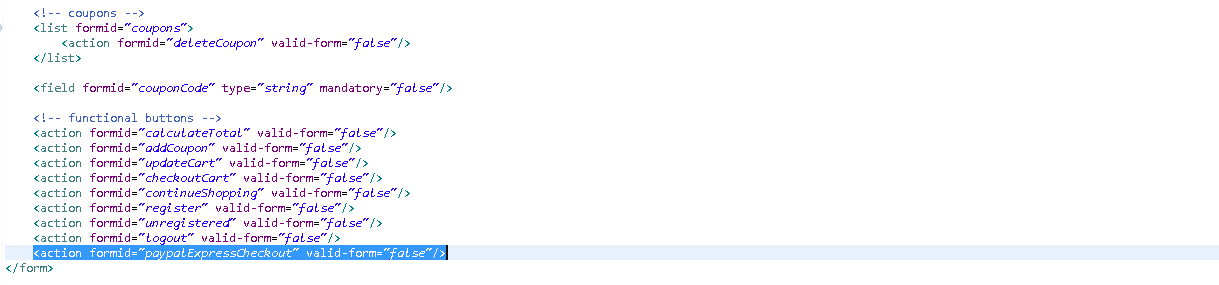
PaymentInstrument != null && PaymentInstrument.paymentMethod == "PayPal"



#### Form - cart.xml

Add an action entry in cart.xml as mentioned below to run Paypal from cart page.

<action formid="paypalExpressCheckout" valid-form="false"/>



Note: Update HandlePayment pipeline node of COPlaceOrder same as mentioned above for Alipay to handle response from service call.

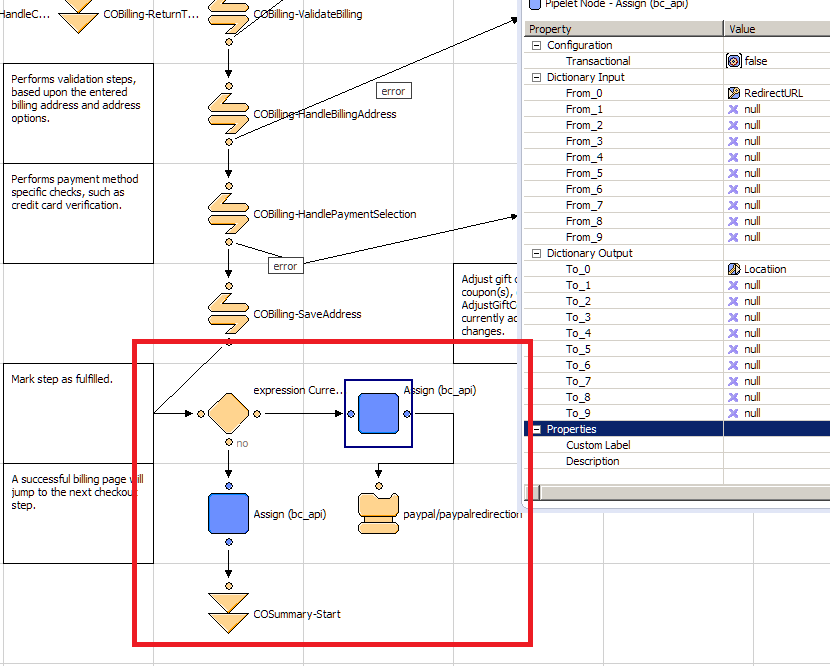
Note: Update PAYPAL\_EXPRESS-Handle and PAYPAL\_EXPRESS-Authorize node same as mentioned above for PayPal Express Checkout.

#### Pipeline - COBilling.xml

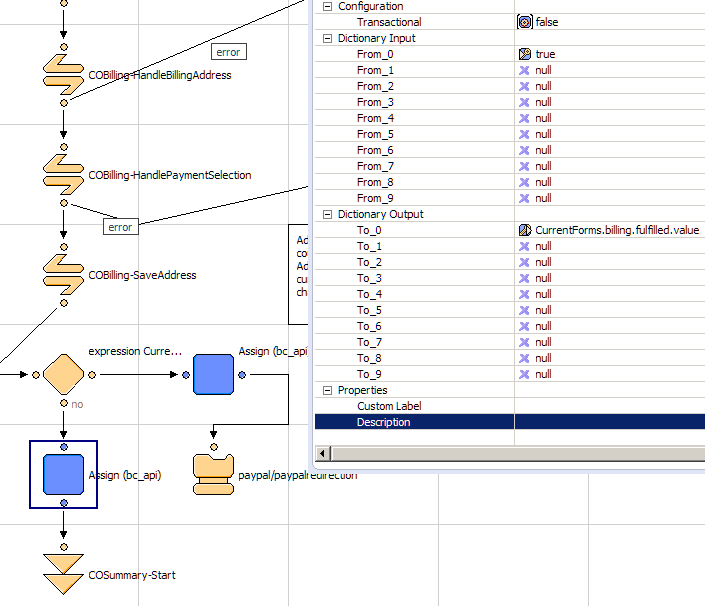
##### Update Start Node

Add decision node just after COBilling-SaveAddress to check for PayPal payment method and set PayPal redirection

CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals('PayPal') && !empty(RedirectURL)



Set the value in assign node if selected payment method is not PayPal



#### Controller - COBilling.js

##### Update “handlePaymentSelection” function

To redirect on PayPal in case of PayPal payment processor selected.

|  |
| --- |
| if (!result) {  result = app.getModel('PaymentProcessor').handle(cart.object, app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value);  }  if(result.OK && app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value =='PayPal' && !empty(result.paypalRedirectUrl))  {  response.redirect(result.paypalRedirectUrl);  }    return result;  } |

##### Update “billing” function

To handle PayPal set service error

|  |
| --- |
| save: function () {  var cart = app.getModel('Cart').get();  if (!resetPaymentForms() || !validateBilling() || !handleBillingAddress(cart) || // Performs validation steps, based upon the entered billing address  // and address options.  handlePaymentSelection(cart).error) {// Performs payment method specific checks, such as credit card verification.  if(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value =='PayPal'){    returnToForm(cart,{PaypalSetServiceError:true});  }else{  returnToForm(cart);  }    } else { |

##### Update “validatePayment” function

Add transaction.wrap to remove existing site genesis issue while validating the existing payment instruments

|  |
| --- |
| /\*\*  \* Revalidates existing payment instruments in later checkout steps.  \*  \* **@param** {module:models/CartModel~CartModel} cart - A CartModel wrapping the current Basket.  \* **@return** {Boolean} true if existing payment instruments are valid, false if not.  \*/  **function** validatePayment(cart) {  **var** paymentAmount, countryCode, invalidPaymentInstruments, result;  **if** (app.getForm('billing').object.fulfilled.value) {  paymentAmount = cart.getNonGiftCertificateAmount();  countryCode = Countries.getCurrent({  CurrentRequest: {  locale: request.locale  }  }).countryCode;    Transaction.wrap(**function** () {  invalidPaymentInstruments = cart.validatePaymentInstruments(customer, countryCode, paymentAmount.value).InvalidPaymentInstruments;  **if** (!invalidPaymentInstruments && cart.calculatePaymentTransactionTotal()) {  result = **true**;  } **else** {  app.getForm('billing').object.fulfilled.value = **false**;  result = **false**;  }  });  } **else** {  result = **false**;  }  **return** result;  } |

#### Controller - COShipping.js

##### Update “start” function

To remove Paypal Payment instrument

|  |
| --- |
| function start() {  var cart, physicalShipments, pageMeta, homeDeliveries;  cart = app.getModel('Cart').get();  if (cart) {  // Redirects to multishipping scenario if more than one physical shipment is contained in the basket.  physicalShipments = cart.getPhysicalShipments();    if(!empty(cart.getPaymentInstruments()) && !empty(cart.getPaymentInstruments('PayPal'))){  Transaction.wrap(function() {  cart.removePaymentInstrument(cart.getPaymentInstruments('PayPal')[0]);    });  }  if (!(Site.getCurrent().getCustomPreferenceValue('enableMultiShipping') && physicalShipments && physicalShipments.size() > 1)) { |

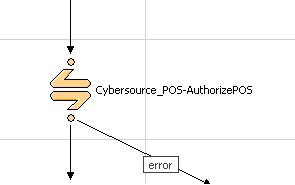
##### Update “selectShippingMethod” function

Update the exisiting method with the code below

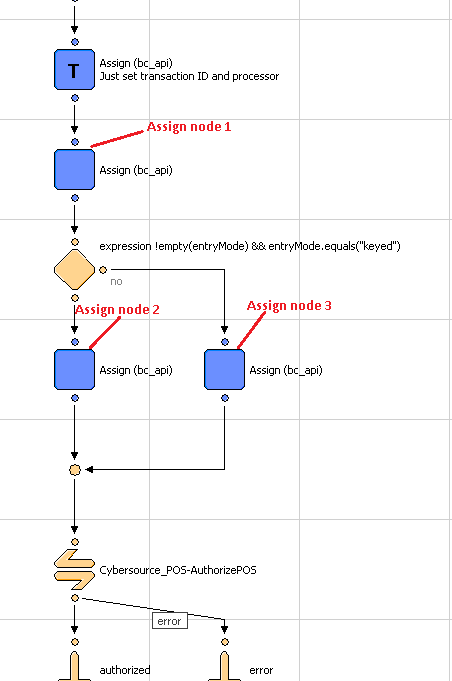
|  |
| --- |
| **function** selectShippingMethod() {  **var** cart, address, applicableShippingMethods, TransientAddress;  TransientAddress = app.getModel('TransientAddress');  cart = app.getModel('Cart').get();  **if** (cart) {  address = **new** TransientAddress();  address.countryCode = request.httpParameterMap.countryCode.stringValue;  address.stateCode = request.httpParameterMap.stateCode.stringValue;  address.postalCode = request.httpParameterMap.postalCode.stringValue;  address.city = request.httpParameterMap.city.stringValue;  address.address1 = request.httpParameterMap.address1.stringValue;  address.address2 = request.httpParameterMap.address2.stringValue;  applicableShippingMethods = cart.getApplicableShippingMethods(address);  Transaction.wrap(**function** () {  cart.updateShipmentShippingMethod(cart.getDefaultShipment().getID(), request.httpParameterMap.shippingMethodID.stringValue, **null**, applicableShippingMethods);  cart.calculate();  });    **if**(!empty(cart.getPaymentInstruments()) && !empty(cart.getPaymentInstruments('PayPal')))  {  **return** **true** ;  }**else**{  app.getView({  Basket: cart.object  }).render('checkout/shipping/selectshippingmethodjson');  }    } **else** {  app.getView.render('checkout/shipping/selectshippingmethodjson');  }  } |

### Retail POS

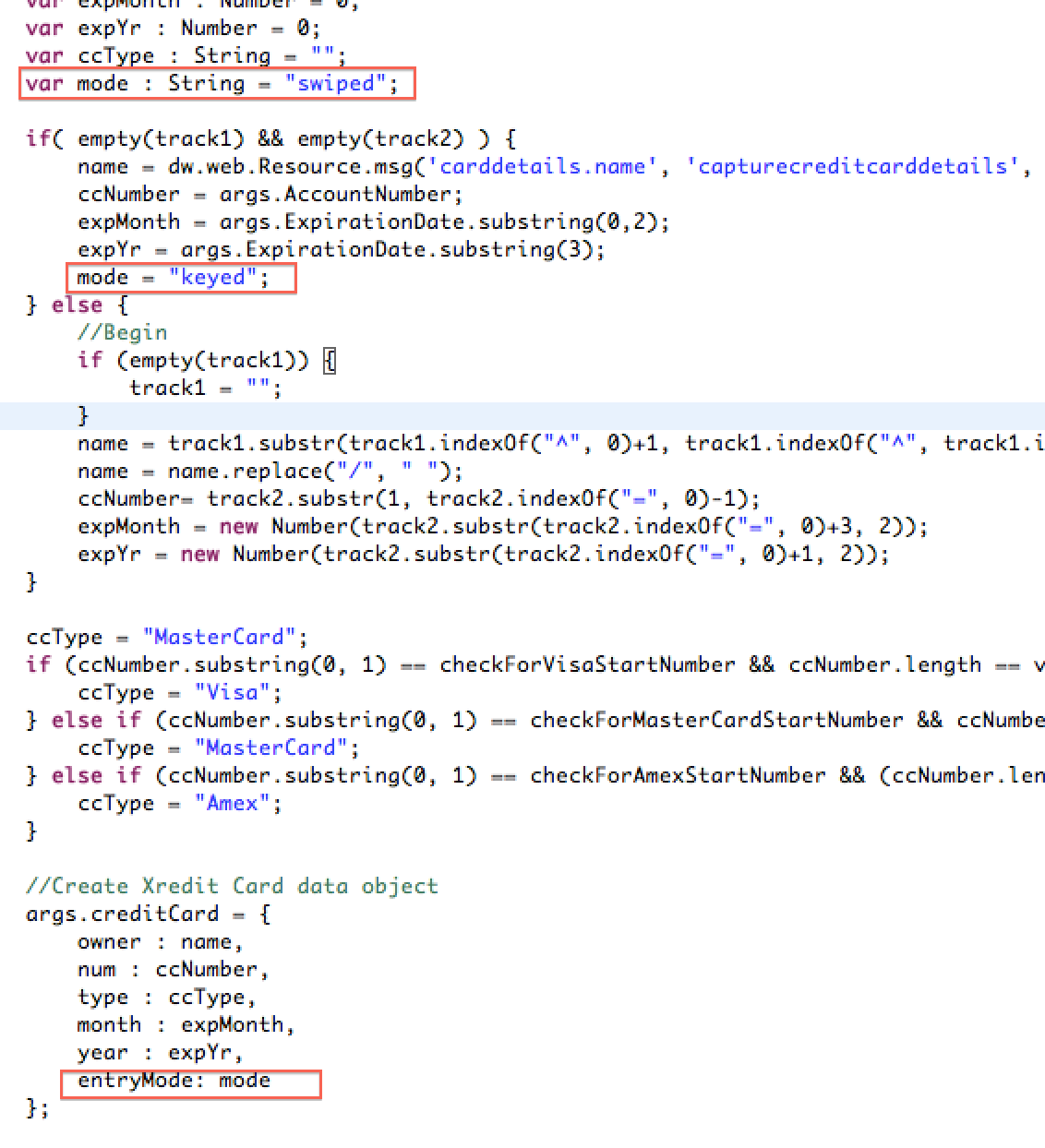
This integration requires only one sub-pipeline to be integrated to your project. The pipeline screenshot is shown below which needs to be called in your project as required:



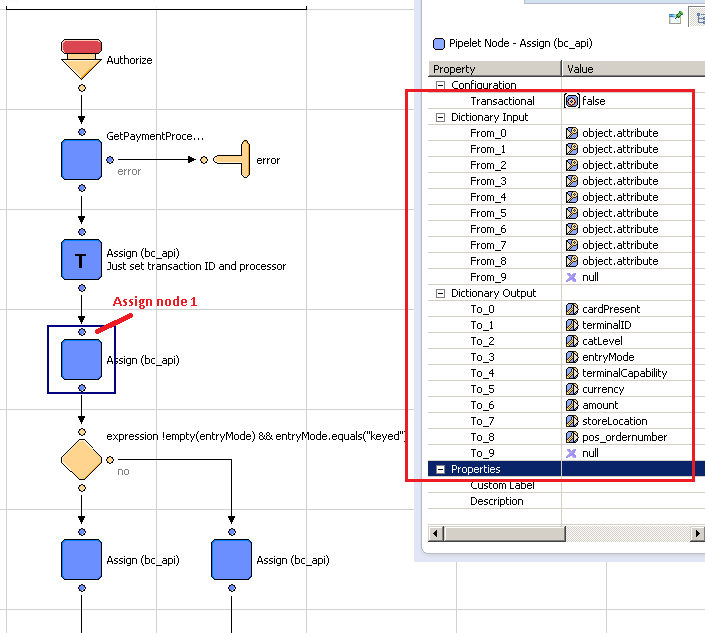
This required call node should be integrated at EACreditCard-Authorize pipeline of DSS app as show below. The track data, expiration date or account number should not be encrypted and may need to be decrypted prior to calling Cybersource\_POS-AuthorizePOS depending on the payment terminal used.



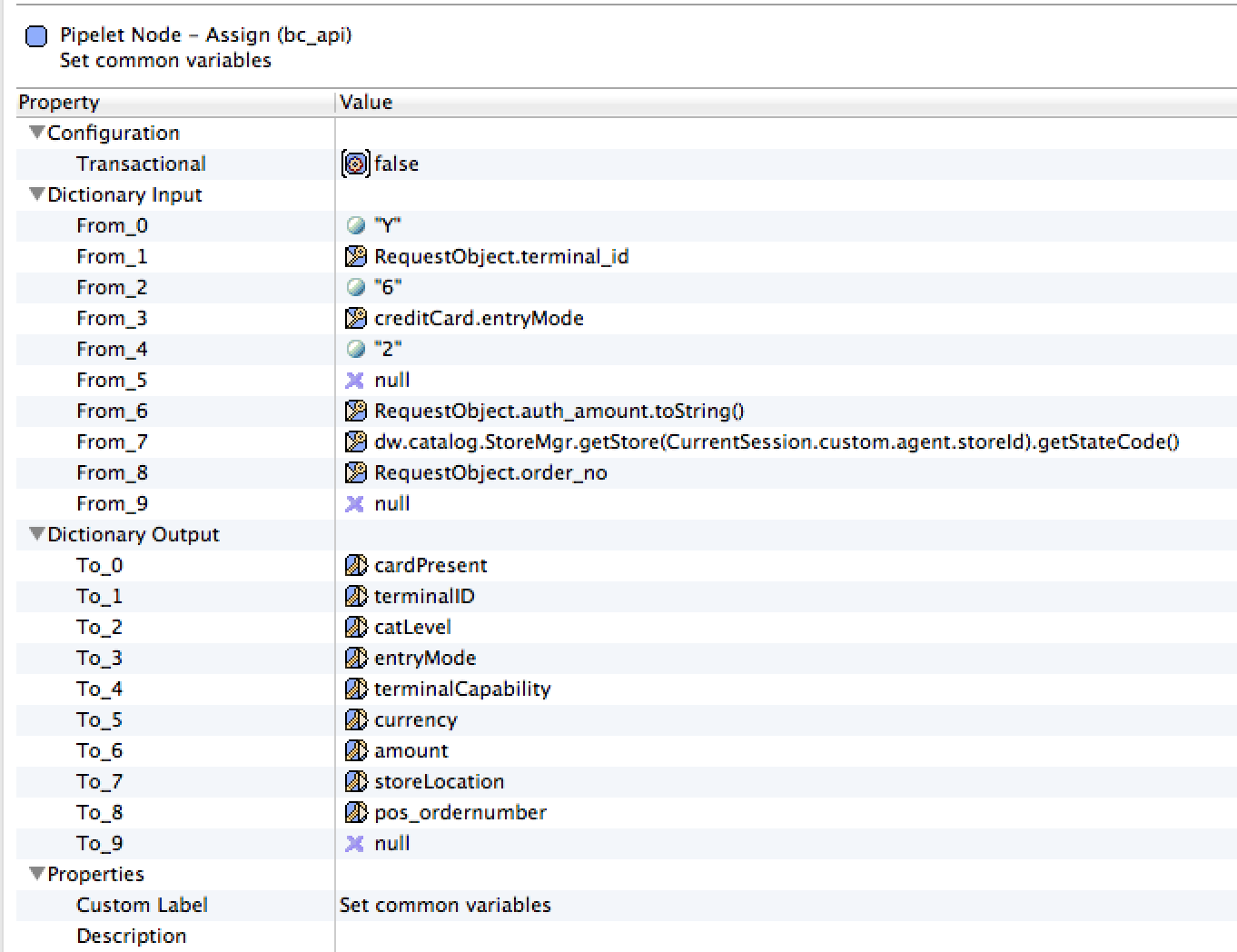
Three assign nodes must be used for required node Cybersource\_POS-AuthorizePOS to work correctly as shown above. The assign node must set variables based on POS terminal entry mode. Below are the use and description for assignment of variables. Assuming that object.attribute variable used in Assign node will be replaced by actual object/variable to get the required values:

POS terminal entry mode can be set in int\_ocapi\_ext/cartridge/scripts/actions/CaptureCreditCardDetails.ds as shown below.

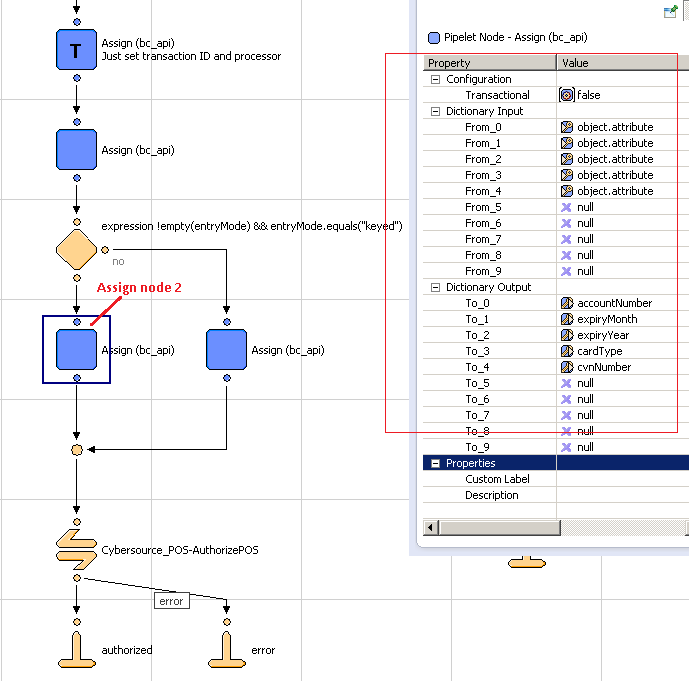
1. Use of Assign node labeled as “Assign node 1” above. This will set the common variables for the transaction irrespective of entry mode used. Below is the screenshot of variables used in this assign node:



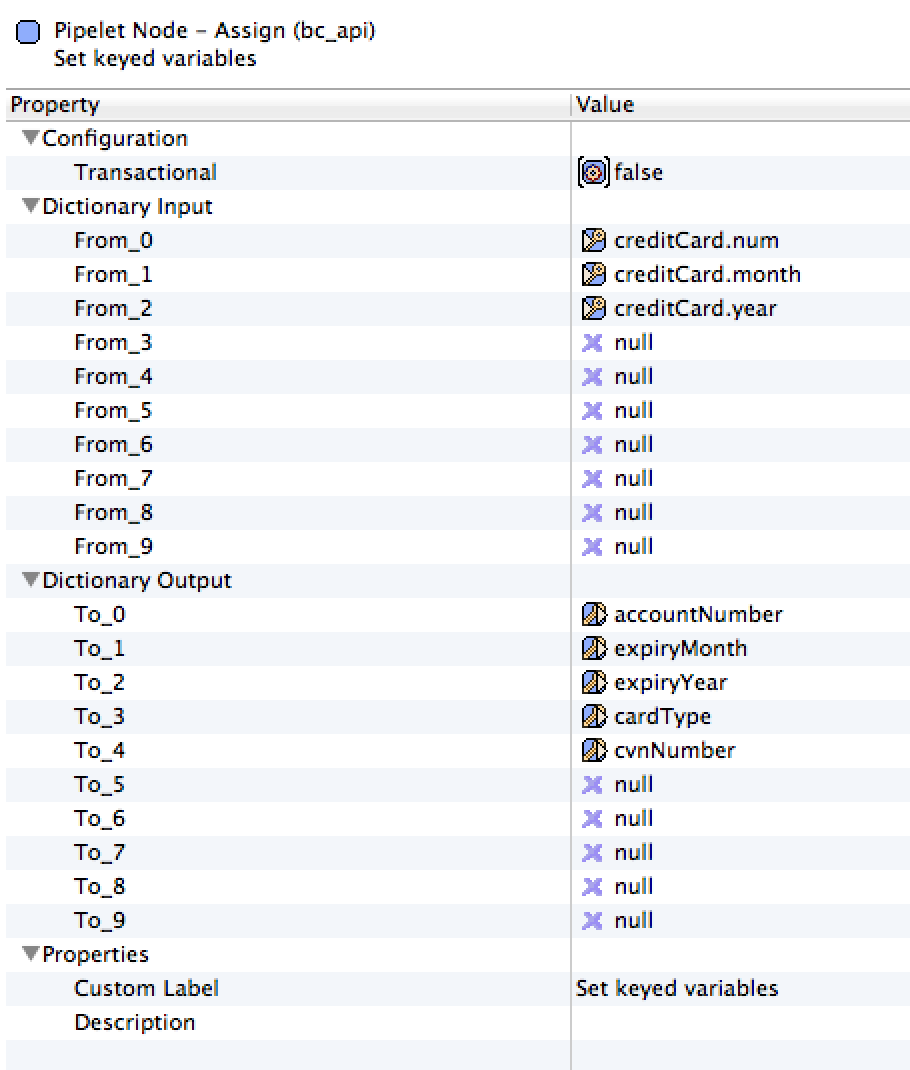
Example input variables from DSS:



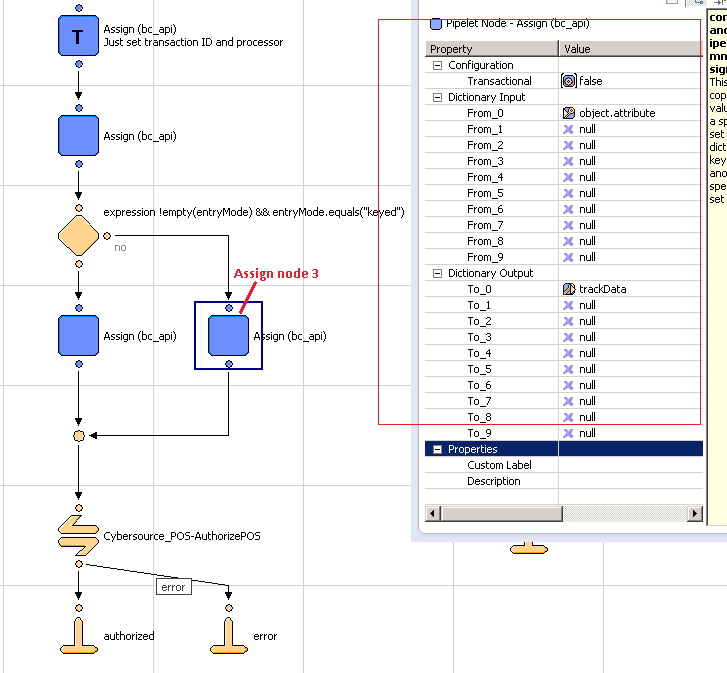
2. Use of Assign node labeled as “Assign node 2” if keyed entry mode is used on the POS terminal device. See below screenshot for the list of variables to set:

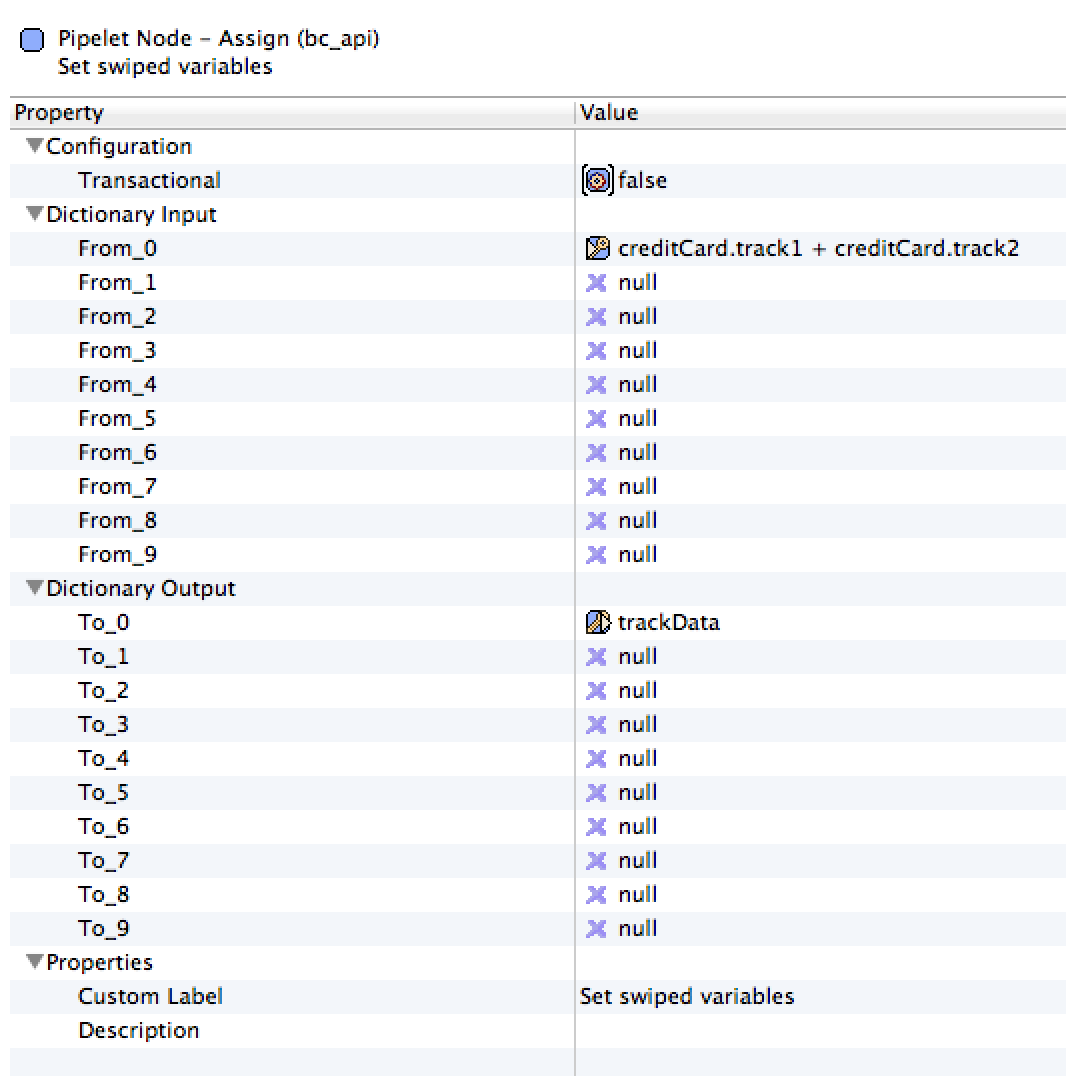


Example input variables from DSS:

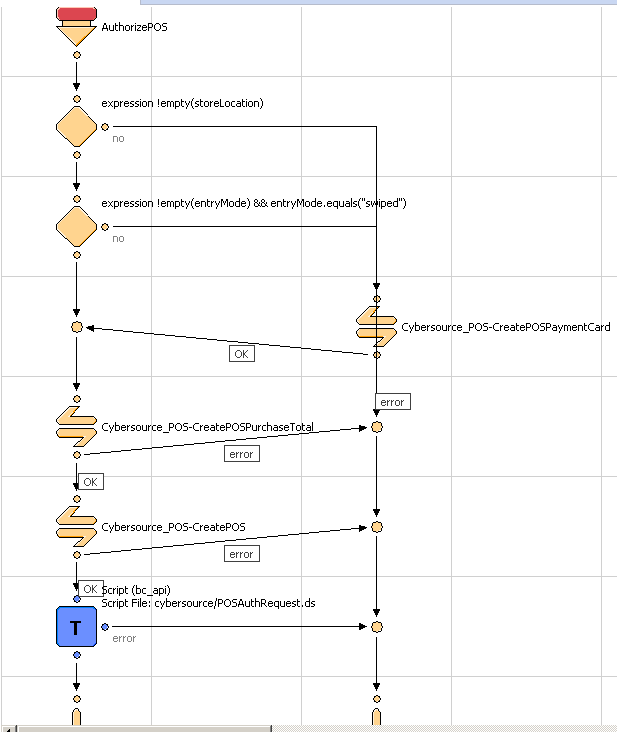


1. Use of Assign node labeled as “Assign node 3” if swiped entry mode is used on the POS terminal device. See below screenshot for the variable to set:



Example of input variables from DSS:

Below is the snapshot of required pipeline.



Below is the list of variables with description. One or two variables become mandatory depending upon other variables and few are optional:

|  |  |  |  |
| --- | --- | --- | --- |
| S. No. | Variable name | Description | Note |
| 1 | cardPresent | Indicates whether the card is present at the time of retail POS transaction. Possible values:  N – card not present  Y – card is present | Required. |
| 2 | catLevel | Type of cardholder activated terminal. Possible values:  1 – Automated dispensing machine  2 – Self-service terminal  3 – Limited amount terminal  4 – In-flight commerce (IFC) terminal  5 – Radio frequency device  6 – Mobile acceptance terminal | Optional. This variable becomes required if terminalID variable is set to a value. |
| 3 | entryMode | Method of entering credit card information into the POS terminal. Possible values:  keyed – Manually keyed into POS terminal.  swiped – Read from credit card magnetic stripe. | Required. |
| 4 | terminalCapability | POS terminal’s capability. Possible values:  1 – Terminal has a magnetic stripe reader only.  2 – Terminal has a magnetic stripe reader and manual entry capability.  3 – Terminal has manual entry capability only. | Required. |
| 5 | terminalID | Identifier for the terminal at your retail location. You can define this value yourself, but consult with the processor for requirements. Terminal ID(s) are configurable in a custom object named ‘POS\_TerminalMapping’ (Refer custom object definition XML to be impoorted). Here terminal device’s serial number will be mapped to a Terminal ID. This variable should be assigned device’s serial number. Code will pick configured Terminal ID if found and passed to CyberSource API in request. | Optional. |
| 6 | trackData | Card’s track 1 and 2 data. Some processors require track 1 data, some processors require track 2 data, and some processors require both track 1 data and track 2 data. To make sure that you provide the required information regardless of the processor that you use now or may use in the future, CyberSource recommends that you send both track 1 and track 2 data in your retail POS requests.  The sentinels are required. The start sentinel (%) indicates the initial data position on the track. The end sentinel (?) follows the final character of data recorded on the track. Details of track 1 and track 2 data for the example %B4111111111111111^SMITH/JOHN^1612101976110000868000000?;4111111111111111=16121019761186800000?  Track 1 – the track 1 data precedes the semicolon (;)  Track 2 – the track 2 data follows the semicolon (;) | Required if entryMode=swiped. |
| 7 | currency | Currency used for order. For possible values refer [ISO Standard Currency Codes](http://apps.cybersource.com/library/documentation/sbc/quickref/currencies.pdf) | If this variable is not set with any currency code then default currency code is retrieved configured for web store in Business Manager. |
| 8 | amount | Grand total for the order. |  |
| 9 | accountNumber | Customer’s credit card number. | This variable becomes mandatory if entryMode=keyed. |
| 10 | cardType | Type of card to authorize. Possible values:  001 – Visa  002 – MasterCard  003 – American Express  004 – Discover  005 – Diners Club  006 – Carte Blanche  007 – JCB | CyberSource strongly recommends that you send the card type even when it is optional for your processor and card type. Omitting the card type can cause the transaction to be processed with the wrong card type. |
| 11 | cvnNumber | This number is never transferred during card swipes. | Optional. |
| 12 | expiryMonth | Two-digit month in which credit card expires. Format: MM. Possible values: 01 through 12. Leading 0 is required. | Required if entryMode=keyed. |
| 13 | expiryYear | Four-digit year in which credit card expires. Format: YYYY. | Required if entryMode=keyed. |
| 14 | storeLocation | Store’s physical location. This is use to configure merchant’s ID and security key in a custom object to call CyberSource API for the transaction. This is dependent upon merchant how they wanted to link store(s) to Merchant ID (MID). For e.g. if merchant has 3 separate CyberSource merchant ID and want to use one MID for store(s) in Massachusetts, 2nd MID for store(s) in New York City, etc. then assign this variable as MA or Massachusetts or any string representing the location AND configure the same value as POS Location field for POS\_MerchantIDs custom object in Business Manager after import. | Location can be set as State code or Zip code or city etc. For e.g. MA (Massachusetts) or  01803 (Burlington, MA) or  Burlington |
| 15 | pos\_ordernumber | Order number for the transaction needs to be set to this variable | Required |

In order to obtain messages in DSS based on failures from Cybersource changes will need to be made to int\_cybersource/cartridge/scripts/cybersource/POSAuthRequest.ds in order to output a Status. Add a @output Status : dw.system.Status to the beginning comment of the file and then set the following Status values:



Then add these lines to the int\_ocapi\_ext/cartridge/templates/resources/eastatus.properties file

CREDITCARD\_INVALID\_USERDENIED=400|InvalidCreditCardExpception|User Denied|User has been denied.

CREDITCARD\_INVALID\_AUTH=400|InvalidCreditCardException|Credit Card Declined|Authorization Denied.

CREDITCARD\_INVALID\_UNABLEPROCESS=400|InvalidCreditCardException|Unable to Process|Call Customer Service.

CREDITCARD\_UNCONFIRMED=400|InvalidCreditCardException|Unconfirmed|Unconfirmed

In the Cybersource\_POS cartridge set the Status to the CreditCardStatus

### ApplePay REST Interface Integration ways with Device/APP

The Interface prepared as part of the document is for testing purpose, during real-time checkout journey of ApplePay there can be multiple ways to utilize interface AS whole or its components. This section depicts anticipated three ways to utilize the interface in real-time, though these ways are not tested (not in scope). Also below steps are assumed to be developed in app/device before utilization of interface components.

1. Device or App code have code written for checkout journey where user opted for ApplePay
2. ApplePay to provide response either Payload or NetworkToken related data
3. The above response must be available in script file defined in hook (say: hook script) where OCAPI hook function to be developed

#### Interface AS A Service

1. Using “Interface AS A Service” has limitation that merchant site MUST disable “Limit Storefront Order” setting
2. Register interface in service initialization script file say “SoapServiceInit.ds”
3. Define above service end point as merchant site URL for pipeline “Cybersource\_ApplePay-Authorize” in BM service configurations
4. Define user/password to be picked from site preferences “cybApplePayInterfaceUser”, “cybApplePayInterfacePassword” in service initialization script file say “SoapServiceInit.ds”
5. The Hook script file having OCAPI hook defined invoke service endpoint by passing required JSON input. (The JSON Input format defined in appropriate REST Interface section above in the document.)
6. Interpret the response received and display thank you page on success and order failure page on failure

#### Interface Direct Functions [when basket or order available]

1. This integration way is recommended when hook script has order or basket available along with other service required inputs. Also merchant site enabled “Limit Storefront Order” setting
2. The Hook script file having OCAPI hook defined call below functions directly and before calling also validate inputs are valid.
3. The function “ApplePayAPIAuthRequest” is called when Payload is available

ApplePayFacade.ApplePayAPIAuthRequest(lineItemCtnr : dw.order.LineItemCtnr, orderNo : String, IPAddress : String, encryptedPaymentData)

|  |  |
| --- | --- |
| Parameter | Type |
| lineItemCtnr | dw.order.LineItemCtnr |
| orderNo | String |
| IPAddress | String |
| encryptedPaymentData | String |

1. The function “ApplePayInAppAuthRequest” is called when network token is available ApplePayFacade.ApplePayInAppAuthRequest(lineItemCtnr : dw.order.LineItemCtnr, orderNo : String, IPAddress : String, cryptogram, networkToken, tokenExpirationMonth, tokenExpirationYear, cardType)

|  |  |
| --- | --- |
| Parameter | Type |
| lineItemCtnr | dw.order.LineItemCtnr |
| orderNo | String |
| IPAddress | String |
| Cryptogram | String |
| networkToken | String |
| tokenExpirationMonth | String |
| tokenExpirationYear | String |
| cardType | String |

1. This function called to update the payment instrument with the service response

PaymentInstrumentUtils.UpdatePaymentTransactionCardAuthorize(paymentInstrument, ServiceResponseObject: Object)

|  |  |
| --- | --- |
| Parameter | Type |
| paymentInstrument | dw.order.PaymentInstrument |
| ServiceResponseObject | Object |

1. Interpret the response received and display thank you page on success and order failure page on failure

#### Interface Functions [when required service request objects available]

1. This integration way is recommended when hook script has order or basket available in for of JSON instead of object along with other service required inputs. Also merchant site enabled “Limit Storefront Order” setting
2. Hook script to prepare CyberSource service related objects like billto, shipto, purchaseTotal etc.
3. The Hook script file having OCAPI hook defined call below functions and before calling also validate inputs are valid.
4. The function “ApplePayAPIObjectAuthRequest” is called when Payload is available

ApplePayFacade.ApplePayAPIObjectAuthRequest(billTo, shipTo, purchaseObject, items, orderNo : String, IPAddress : String, encryptedPaymentData)

|  |  |
| --- | --- |
| Parameter | Type |
| billTo | Cybersource\_BillTo\_Object |
| shipTo | Cybersource\_ShipTo\_Object |
| purchaseObject | Cybersource\_PurchaseTotals\_Object |
| Items | Cybersource\_Item\_Object |
| orderNo | String |
| IPAddress | String |
| encryptedPaymentData | String |

1. The function “ApplePayInAppObjectAuthRequest” is called when Network Token is available

ApplePayFacade.ApplePayInAppObjectAuthRequest(billTo, shipTo, purchaseObject, items, orderNo : String, IPAddress : String, cryptogram, networkToken, tokenExpirationMonth, tokenExpirationYear, cardType)

|  |  |
| --- | --- |
| Parameter | Type |
| billTo | Cybersource\_BillTo\_Object |
| shipTo | Cybersource\_ShipTo\_Object |
| purchaseObject | Cybersource\_PurchaseTotals\_Object |
| Items | Cybersource\_Item\_Object |
| orderNo | String |
| IPAddress | String |
| Cryptogram | String |
| networkToken | String |
| tokenExpirationMonth | String |
| tokenExpirationYear | String |
| cardType | String |

1. This function called to update the payment instrument with the service response

PaymentInstrumentUtils.UpdatePaymentTransactionCardAuthorize(paymentInstrument, ServiceResponseObject: Object)

|  |  |
| --- | --- |
| Parameter | Type |
| paymentInstrument | dw.order.PaymentInstrument |
| ServiceResponseObject | Object |

1. Interpret the response received and display thank you page on success and order failure page on failure

### Visa Checkout

#### Genric Changes:

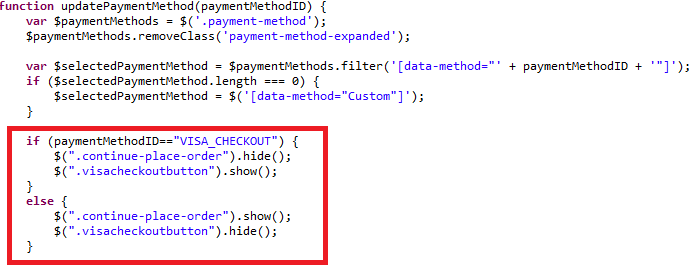
##### billing.js

1. Show Hide Visa checkout button :

Add a condition in function updatePaymentmethod for visa checkout button display on selected payment methods as ‘VISA\_CHECKOUT’ otherwise hide.

Sample code :

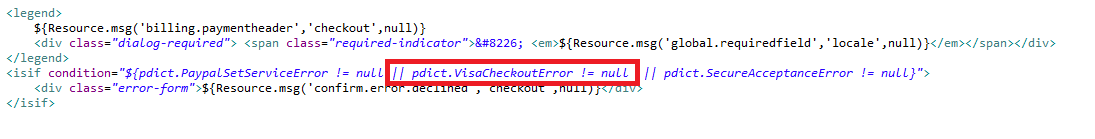
|  |
| --- |
| **if** (paymentMethodID=="VISA\_CHECKOUT") {  $(".continue-place-order").hide();  $(".visacheckoutbutton").show();  }  **else** {  $(".continue-place-order").show();  $(".visacheckoutbutton").hide();  } |



##### paymentmethods.isml

1. Add error condition for Visa Checkout error handling ex:

<isif condition="${pdict.PaypalSetServiceError != null || pdict.VisaCheckoutError != null || pdict.SecureAcceptanceError != null}">



##### billing.isml

* Add class on “continue to place order” button that would be used in billing.js to hide or show button based on payment method selection

|  |
| --- |
| <button class="button-fancy-large secureacceptance continue-place-order" type="button" name="${pdict.CurrentForms.billing.save.htmlName}" value="${Resource.msg('global.continueplaceorder','locale',null)}"><span>${Resource.msg('global.continueplaceorder','locale',null)}</span></button> |

* Include Visa Checkout Button

Add following div section after the form ends

<div class=*"visacheckoutbutton hide"* style="text-align: *center*;">

<isinclude url=*"${URLUtils.url('VisaCheckout-Button')}"*/>

</div>

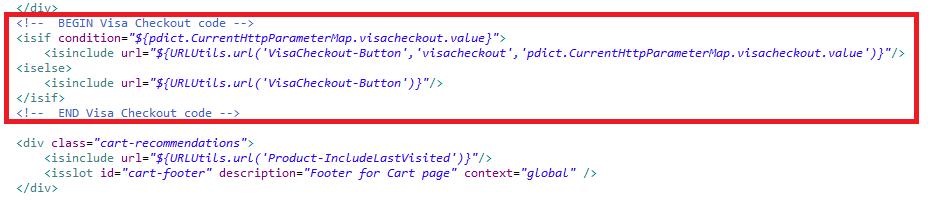


##### cart.isml

1. Include Visa checkout button:

Add following lines before cart-recommendations div

|  |
| --- |
| <!-- BEGIN Visa Checkout code -->  <isif condition="${pdict.CurrentHttpParameterMap.visacheckout.value}">  <isinclude url="${URLUtils.url('VisaCheckout-Button','visacheckout','pdict.CurrentHttpParameterMap.visacheckout.value')}"/>  <iselse>  <isinclude url="${URLUtils.url('VisaCheckout-Button')}"/>  </isif>  <!-- END Visa Checkout code --> |



##### minicart.isml

1. Add following line in div having id <div class="mini-cart-totals"> before checkout button

|  |
| --- |
| <!-- BEGIN Visa Checkout code -->  <isif condition="${empty(pdict.CurrentHttpParameterMap.visacheckout.value) || !pdict.CurrentHttpParameterMap.visacheckout.value}">  <isinclude url="${URLUtils.url('VisaCheckout-Button','buttonsource','minicart')}"/>  <iselse>  <isinclude url="${URLUtils.url('VisaCheckout-Button')}"/>  </isif>  <!-- END Visa Checkout code --> |

##### footer\_UI.isml

1. Include the template visacheckout/launch.isml at the end of the file.

<iscomment>Visa Checkout launch</iscomment>

<isinclude template="visacheckout/launch.isml" />

##### header.isml

1. In the header section replace mini-cart section with below snippet

<iscomment>INCLUDE: Mini-cart, do not cache</iscomment>

|  |
| --- |
| <div id=*"mini-cart"*>  <isif condition=*"${!empty(pdict.CurrentHttpParameterMap.visacheckout.value) && pdict.CurrentHttpParameterMap.visacheckout.value}"*>  <isinclude url=*"${URLUtils.url('Cart-MiniCart','visacheckout',pdict.CurrentHttpParameterMap.visacheckout.value)}"*/>  <iselse>  <isinclude url=*"${URLUtils.url('Cart-MiniCart')}"*/></isif>  </div> |

##### htmlhead.isml

1. Add following line to prevent visa checkout clickjacking in the end

<iscomment>Visa Checkout clickjacking prevention</iscomment>

<isinclude template="visacheckout/clickjackingPrevent.isml" />

##### checkout.properties

Add following values in checkout.properties file for visa checkout changes.

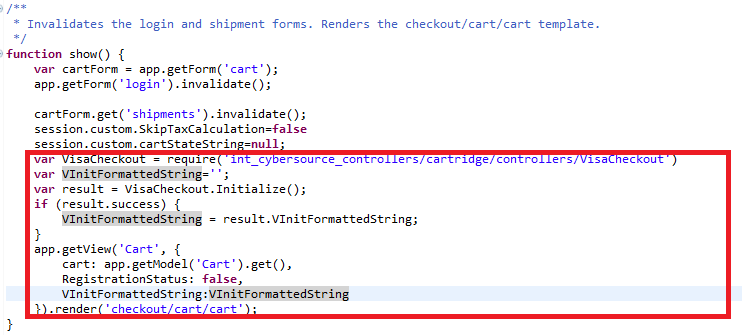
|  |
| --- |
| VisaCheckoutError = Your shopping cart can currently not be ordered. If you have any questions, please contact Customer Service.  visaCheckout.shippingUpdate.prepareShipments = PrepareShipments failed |

#### Controller - Cart.js

##### Update the Show() method

1. Add following line of code to show() function where during view render pass additional field “VInitFormattedString” as shown in the snippet:

|  |
| --- |
| var VisaCheckout = require('int\_cybersource\_controllers/cartridge/controllers/VisaCheckout')  var VInitFormattedString='';  var result = VisaCheckout.Initialize();  if (result.success) {  VInitFormattedString = result.VInitFormattedString;  }  app.getView('Cart', {  cart: app.getModel('Cart').get(),  RegistrationStatus: false,  VInitFormattedString:VInitFormattedString  }).render('checkout/cart/cart'); |



#### Controller - COBilling.js

##### Update the returnToForm() method

1. Add following code of line

var VisaCheckout = require('int\_cybersource\_controllers/cartridge/controllers/VisaCheckout')

var VInitFormattedString='';

var result = VisaCheckout.Initialize(false);//no delivery address in lightbox

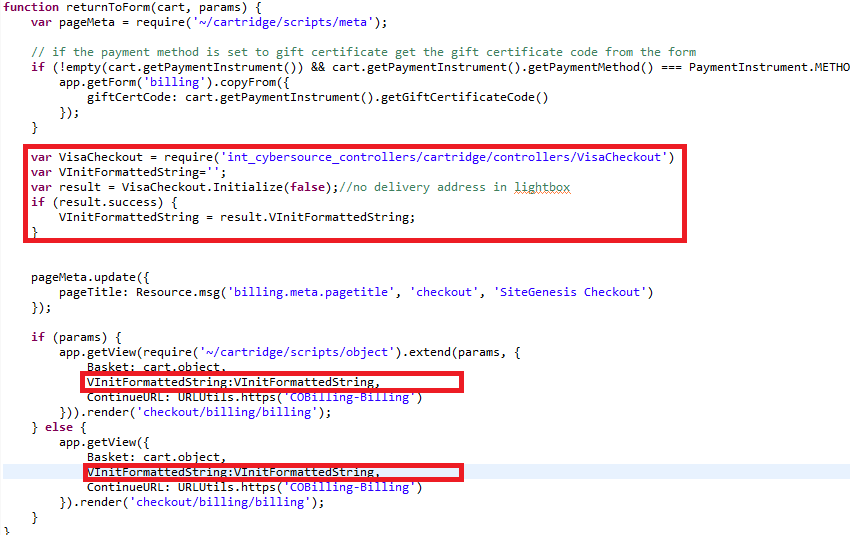
if (result.success) {

VInitFormattedString = result.VInitFormattedString;

}

1. Add VInitFormattedString:VInitFormattedString, in the view rendering

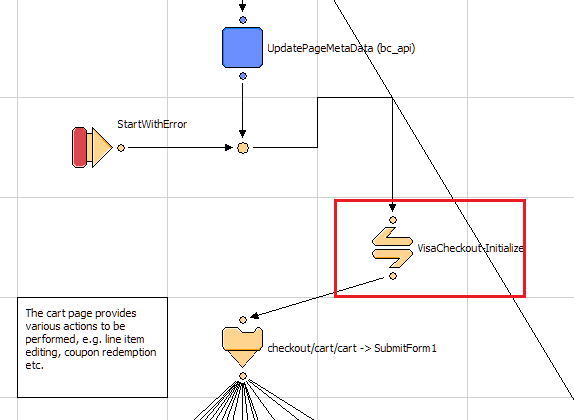
VInitFormattedString:VInitFormattedString,



#### Pipeline – Cart.xml

##### Update the Show sub pipeline

1. Add call node VisaCheckout-Initialize just before interaction continue node of cart.isml.



#### Pipeline – COBilling.xml

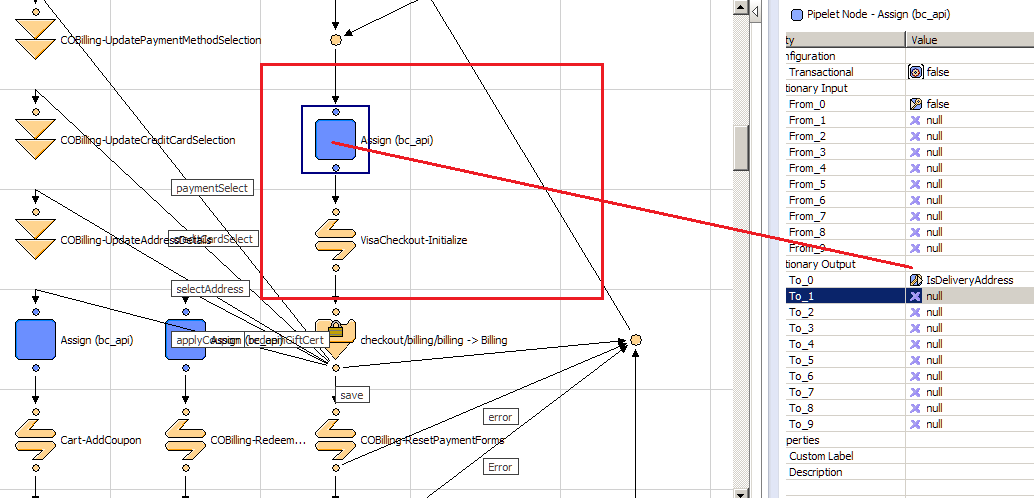
##### Update the Start Node

1. Add an assign node above interaction continue node checkout/billing/billing having property

Input value = false ; output value = IsDeliveryAddress

1. Add the call node VisaCheckout-Initialize in the Start node of COBilling.xml pipeline.

This call node is placed just above the interaction continuous node having template as Template checkout/billing/billing



### Secure Acceptance

#### Generic Section

##### JS file – billing.js [compiled to app.js]

##### Update “export.init “function

* Add below code snippet after $('#creditCardList').on('change', function () {

|  |
| --- |
| // Secure Acceptance Redirect or iframe payment method : on selection change of saved credit card  $('.creditCardList').on('change', **function** () {  **var** cardUUID = $(**this**).val();  **if** (!cardUUID) {**return**;}    **var** selectedPaymentMethod = $selectPaymentMethod.find(':checked').val();  populateCreditCardForm(cardUUID,selectedPaymentMethod);  // remove server side error  $('.required.error').removeClass('error');  $('.error-message').remove();  }); |

###### Update “populateCreditCardForm” function

|  |
| --- |
| **function** populateCreditCardForm(cardID,selectedPaymentMethod) {  // load card details  **var** url = util.appendParamToURL(Urls.billingSelectCC, 'creditCardUUID', cardID);  ajax.getJson({  url: url,  callback: **function** (data) {  **if** (!data) {  window.alert(Resources.CC\_LOAD\_ERROR);  return false;  }  **switch** (selectedPaymentMethod) {  **case** "SA\_REDIRECT":  $('.payment-method-expanded .saCCToken .field-wrapper').val(data.selectedCardID); $("#dwfrm\_billing\_paymentMethods\_creditCard\_selectedCardID").val(data.selectedCardID);  **break**;  **case** "SA\_IFRAME":  $('.payment-method-expanded .saIframeCCToken .field-wrapper').val(data.selectedCardID); $("#dwfrm\_billing\_paymentMethods\_creditCard\_selectedCardID").val(data.selectedCardID);  **break**;  **case** "CREDIT\_CARD":  setCCFields(data);  **break**;  default:  setCCFields(data);  }  }  });  } |

##### Template – Cart.isml

* Add below error condition just after cart-banner

|  |
| --- |
| <isslot id="cart-banner" description="Banner for Cart page" context="global" />  <isif condition="${pdict.CurrentHttpParameterMap.SecureAcceptanceError != null && !empty(pdict.CurrentHttpParameterMap.SecureAcceptanceError.stringValue)}">  <div class="error-form">${Resource.msg('sa.cart.payment.error.declined','cybersource',null)}</div>  </isif> |

##### Template – Summary.isml

* Replace the PlaceOrderError section with below code

|  |
| --- |
| <isif condition=*"${pdict.CurrentHttpParameterMap.SecureAcceptanceError != null && !empty(pdict.CurrentHttpParameterMap.SecureAcceptanceError.stringValue)}"*>  <div class=*"error-form"*>${Resource.msg('confirm.error.technical','checkout',null)}</div>  <iselseif condition=*"${pdict.PlaceOrderError != null}"*>  <div class=*"error-form"*>${Resource.msg(pdict.PlaceOrderError.code,'checkout',null)}</div>  </isif> |

##### Template – Billing.isml

* Add below error condition just after checkout progress indicator

|  |
| --- |
| <isif condition="${!pdict.CurrentForms.multishipping.entered.value}">  <ischeckoutprogressindicator step="2" multishipping="false" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'false' : 'true'}"/>  <iselse/>  <ischeckoutprogressindicator step="3" multishipping="true" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'false' : 'true'}"/>  </isif>    <isif condition=*"${pdict.CurrentHttpParameterMap.SecureAcceptanceError != null && !empty(pdict.CurrentHttpParameterMap.SecureAcceptanceError.stringValue)}"*>  <div class=*"error-form"*>${Resource.msg('sa.billing.payment.error.declined','cybersource',null)}</div>  </isif> |

##### Template – paymentmethod.isml

* Add below code snippet to handle secure acceptance error

|  |
| --- |
| <isif condition="${pdict.PaypalSetServiceError != null || pdict.VisaCheckoutError != null || pdict.SecureAcceptanceError != null}">  <div class=*"error-form"*>${Resource.msg('confirm.error.declined','checkout',null)}</div>  </isif> |

##### Form – creditcard.xml

* Set the default value of formid="saveCard" to false

|  |
| --- |
| <field formid="saveCard" label="creditcard.savecard" type="boolean" mandatory="false" default-value="false" /> |

* Add more year options as below:

|  |
| --- |
| <option optionid="2022" label="year.2022" value="2022"/>  <option optionid="2023" label="year.2023" value="2023"/>  <option optionid="2024" label="year.2024" value="2024"/>  <option optionid="2025" label="year.2025" value="2025"/>  <option optionid="2026" label="year.2026" value="2026"/>  <option optionid="2027" label="year.2027" value="2027"/>  <option optionid="2028" label="year.2028" value="2028"/>  <option optionid="2029" label="year.2029" value="2029"/>  <option optionid="2030" label="year.2030" value="2030"/>  <option optionid="2031" label="year.2031" value="2031"/>  <option optionid="2032" label="year.2032" value="2032"/>  <option optionid="2033" label="year.2033" value="2033"/>  <option optionid="2034" label="year.2034" value="2034"/>  <option optionid="2035" label="year.2035" value="2035"/>  <option optionid="2036" label="year.2036" value="2036"/>  <option optionid="2037" label="year.2037" value="2037"/> |

##### Resources – form.properties

* Add year values above year year.2022=2022

|  |
| --- |
| year.2037=2037  year.2036=2036  year.2035=2035  year.2034=2034  year.2033=2033  year.2032=2032  year.2031=2031  year.2030=2030  year.2029=2029  year.2028=2028  year.2027=2027  year.2026=2026  year.2025=2025  year.2024=2024  year.2023=2023  year.2022=2022 |

##### Resources – checkout.properties

* Add below line at the at the end of file

|  |
| --- |
| checkout.invalid.credit.card.info = Enter correct credit card detail  checkout.getsignature.service.problem = Problem in service |

#### Secure Acceptance Redirect Section

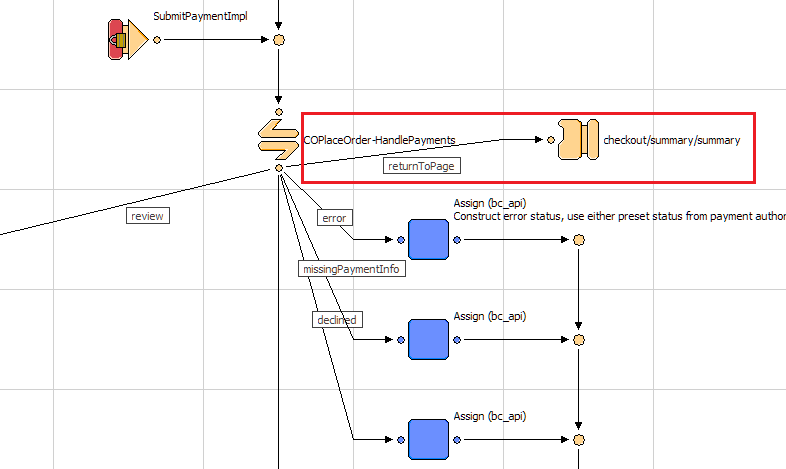
* All secure acceptance redirect implementation changes will be inside int\_CyberSource and int\_CyberSource\_controller cartridge

#### Secure Acceptance Iframe Section

##### Pipeline - COPlaceOrder.xml

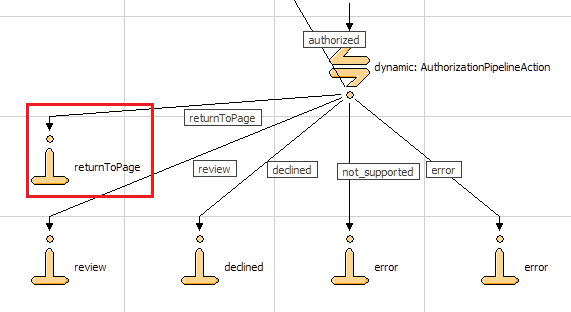
###### Update “Start” node

* **Add a new** transition”**returnToPage”** in COPlaceOrder-HandlePaymentsinside COPlaceOrder-Start Node and call the interaction node checkout/summary/summary to render Iframe from Summary page



###### Update “HandlePayments” node

* **Add a new** transition”**returnToPage”** in COPlaceOrder-HandlePayments



##### Controller - COPlaceOrder.js

###### Update “Start” function

* Add below code for new transition returnToPage(if not found in [generic section](#_Update_Replace_))

|  |
| --- |
| **var** handlePaymentsResult = handlePayments(order);  **if** (handlePaymentsResult.error) {  session.custom.SkipTaxCalculation=**false**;  **return** Transaction.wrap(**function** () {  OrderMgr.failOrder(order);  **return** {  error: **true**,  PlaceOrderError: **new** Status(Status.ERROR, 'confirm.error.declined')  };  });  }**else** **if**(handlePaymentsResult.returnToPage){  app.getView({Order : handlePaymentsResult.order}).render('checkout/summary/summary');  **return** {};  } **else** **if** (handlePaymentsResult.missingPaymentInfo) { |

###### Update “HandlePayments” function

* **Add a new** transition”**returnToPage”** in COPlaceOrder-HandlePayments

|  |
| --- |
| authorizationResult = PaymentProcessor.authorize(order, paymentInstrument);  **if** (authorizationResult.not\_supported || authorizationResult.error || authorizationResult.failed) {  **return** {  error: **true**  };  } **else** **if**(authorizationResult.returnToPage){  **return** {returnToPage :**true**, order : order  };  } |

##### Pipeline - COSummary.xml

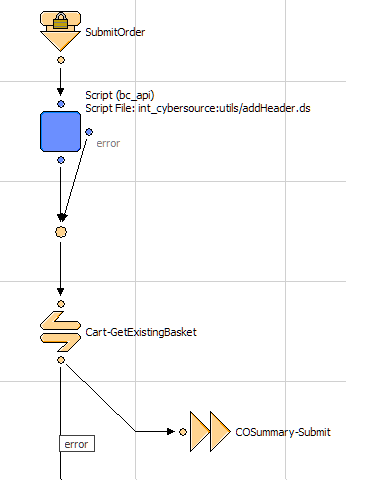
###### Create new “SubmitOrder” node

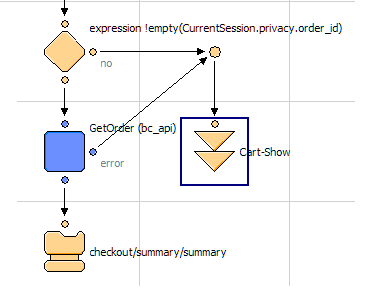
1. Create an new Public and secured node as “SubmitOrder”
2. Add a script file int\_cybersource:utils/addHeader.ds with no input parameters
3. Add a call node Cart-GetExistingBasket
4. Jump to COSummary-Submit if cart is fetched
5. If ERROR, add a decision node to check if order\_id is not empty:

!empty(CurrentSession.privacy.order\_id)

1. If not empty, call GetOrder pipelet with the input as CurrentSession.privacy.order\_id and set Order in output
2. If error fetching Order, jump to Cart-Show

Please find below image for the same:





##### Controller - COSummary.js

###### Create new “SubmitOrder” function

Add a new function as below and add the export of the function at the end of file

|  |
| --- |
| **function** submitOrder() {  **var** cart = Cart.get();  **if** (cart) {  submit();  **return**;  } **else** **if** (!empty(session.privacy.order\_id)) {  response.addHttpHeader("X-FRAME-OPTIONS","SAMEORIGIN");  **var** Order = app.getModel('Order');  app.getView({  Order : Order.get(session.privacy.order\_id).object  }).render('checkout/summary/summary');  **return**;  } **else** {  app.getController('Cart').Show();  **return** {};  }  } |

|  |
| --- |
| exports.SubmitOrder = guard.ensure(['https'], submitOrder); |

##### Template changes

###### Update “summary.isml”

Add below line on top of file

|  |
| --- |
| <isset name="summarypage" value="${true}" scope="page"/>  <isset name="paymentMethod" value="${null}" scope="page"/> |

* Update this code <isreportcheckout checkoutstep="${5}" checkoutname="${'OrderSummary'}"/> with below code snippet

|  |
| --- |
| <isif condition=*"${!empty(pdict.Basket)}"*>  <isset name=*"LineCntr"* value=*"${pdict.Basket}"* scope=*"page"*/>  <iselseif condition=*"${!empty(pdict.Order)}"*>  <isset name=*"LineCntr"* value=*"${pdict.Order}"* scope=*"page"*/>  </isif>  <isset name=*"summaryaction"* value=*"${URLUtils.https('COSummary-Submit')}"* scope=*"page"* />  <script src=*"${URLUtils.staticURL('/lib/jquery/jquery-1.11.1.min.js')}"* type=*"text/javascript"*></script>  <isif condition=*"${!empty(LineCntr.getPaymentInstruments())}"*>  <isloop items=*"${LineCntr.getPaymentInstruments()}"* var=*"paymentInstr"* status=*"loopstate"*>  <isset name=*"paymentMethod"* value=*"${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).ID}"* scope=*"page"*/>  <isif condition=*"${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).ID=='SA\_IFRAME'}"*>  <isset name=*"summaryaction"* value=*"${URLUtils.https('COSummary-SubmitOrder')}"* scope=*"page"* />  <isbreak/>  </isif>  </isloop>  </isif>  <isif condition=*"${!empty(LineCntr)}"*>  <isreportcheckout checkoutstep=*"${5}"* checkoutname=*"${'OrderSummary'}"*/>  <isif condition=*"${!pdict.CurrentForms.multishipping.entered.value}"*>  <ischeckoutprogressindicator step=*"3"* multishipping=*"false"* rendershipping=*"${LineCntr.productLineItems.size() == 0 ? 'false' : 'true'}"*/>  <iselse/>  <ischeckoutprogressindicator step=*"4"* multishipping=*"true"* rendershipping=*"${LineCntr.productLineItems.size() == 0 ? 'false' : 'true'}"*/>  </isif>  <isif condition=*"${pdict.PlaceOrderError != null}"*>  <div class=*"error-form"*>${Resource.msg(pdict.PlaceOrderError.code,'checkout',null)}</div>  </isif>  </isif> |

* Update the line <isloop items="${pdict.Basket.shipments}" var="shipment" status="shipmentloopstate"> with below code

|  |
| --- |
| <isloop items="${LineCntr.shipments}" var="shipment" status="shipmentloopstate">  <isif condition="${shipment.productLineItems.size() > 0 || shipment.giftCertificateLineItems.size() > 0}">  <isset name="shipmentCount" value="${shipmentCount+1}" scope="page"/>  <isif condition="${LineCntr.shipments.size() > 1}">  <tr class="cart-row">  <td colspan="5">  <div class="shipment-label">${Resource.msgf('multishippingshipments.shipment','checkout',null, shipmentCount)}</div>  </td>  </tr>  </isif> |

* Update pdict.Basket with LineCntr for below line

|  |
| --- |
| <iscomment>RENDER COUPON/ORDER DISCOUNTS</iscomment>  <isloop items="${LineCntr.couponLineItems}" var="couponLineItem" status="cliloopstate"> |

* Update pdict.Basket with LineCntr for below line

|  |
| --- |
| <isloop items="${LineCntr.priceAdjustments}" var="priceAdjustment" status="cliloopstate"> |

* Update pdict.Basket with LineCntr for below line

|  |
| --- |
| <isordertotals p\_lineitemctnr="${LineCntr}" p\_showshipmentinfo="${false}" p\_shipmenteditable="${false}" p\_totallabel="${Resource.msg('summary.ordertotal','checkout',null)}"/> |

* Add a condition just above form action and update the form action as below

|  |
| --- |
| <isif condition="${!empty(pdict.Basket)}">  <form action="${summaryaction}" method="post" class="submit-order">  <fieldset>  <div class="form-row">  <a class="back-to-cart" href="${URLUtils.url('Cart-Show')}">  <isprint value="${Resource.msg('summary.editcart','checkout',null)}" encoding="off" /></a>  <button class="button-fancy-large" type="submit" name="submit" value="${Resource.msg('global.submitorder','locale',null)}">  ${Resource.msg('global.submitorder','locale',null)}  </button>  </div>  </fieldset>  </form>  </isif> |

* Add below condition at the end of file and just above </isdecorate>

|  |
| --- |
| <isif condition=*"${paymentMethod != null && paymentMethod=='SA\_IFRAME'}"*>  <isinclude template="custom/secureAcceptanceIframeSummmary"/>  </isif>  </isdecorate> |

###### Update “miniBillingInfo.isml”

Replace the line

<isset name=*"billingAddress"* value=*"${pdict.Basket.billingAddress}"* scope=*"page"*/>

<isset name=*"paymentInstruments"* value=*"${pdict.Basket.paymentInstruments}"* scope=*"page"*/> with the code below

|  |
| --- |
| <isif condition="${!empty(pdict.Basket)}"><isset name="lineCtnr" value="${pdict.Basket}" scope="page"/>  <isset name="billingAddress" value="${lineCtnr.billingAddress}" scope="page"/>  <isset name="paymentInstruments" value="${lineCtnr.paymentInstruments}" scope="page"/>  <iselseif condition="${!empty(pdict.Order)}">  <isset name="lineCtnr" value="${pdict.Order}" scope="page"/>  <isset name="billingAddress" value="${pdict.Order.billingAddress}" scope="page"/>  <isset name="paymentInstruments" value="${pdict.Order.paymentInstruments}" scope="page"/>  </isif>  <isif condition="${!empty(billingAddress)}"> |

* Replace <a tag in billingAddress if condition with the line below

|  |
| --- |
| <div class="mini-billing-address order-component-block">  <h3 class="section-header">  <isif condition="${( empty(pdict.CurrentHttpParameterMap.paypalOrigin.value) || pdict.CurrentHttpParameterMap.paypalOrigin.value.equals('cart')) && ( empty(pdict.paypalOrigin) || !(pdict.paypalOrigin.equals('cart')))}">  <isif condition="${!empty(pdict.Basket)}"><a href="${URLUtils.https('COBilling-Start')}" class="section-header-note">${Resource.msg('global.edit','locale',null)}</a></isif>  </isif>  ${Resource.msg('minibillinginfo.billingaddress','checkout',null)}  </h3>  <div class="details">  <isminicheckout\_address p\_address="${billingAddress}"/>  </div>  </div> |

* Replace <a tag in paymentInstruments if condition with the line below

|  |
| --- |
| <isloop items=*"${paymentInstruments}"* var=*"paymentInstr"* status=*"loopstate"*>          <div class=*"mini-payment-instrument  order-component-block <isif condition="*${loopstate.first}"> first <iselseif condition=*"${loopstate.last}"*> last</isif>">  <h3 class=*"section-header"*>                <isif condition=*"${( empty(pdict.CurrentHttpParameterMap.paypalOrigin.value) || pdict.CurrentHttpParameterMap.paypalOrigin.value.equals('cart')) && ( empty(pdict.paypalOrigin) || !(pdict.paypalOrigin.equals('cart')))}"*>                                    <isif condition=*"${!empty(pdict.Basket)}"*><a href=*"${URLUtils.https('COBilling-Start')}"*  class=*"section-header-note"*>${Resource.msg('global.edit','locale',null)}</a></isif>                             </isif>                 <isif condition=*"${loopstate.first}"*><span>${Resource.msg('minibillinginfo.paymentmethod','checkout',null)}</span></isif>              </h3> |

###### Update “miniSumary.isml”

* Add below code snippet just above this line <isif condition="${!empty(pdict.checkoutstep)}">

|  |
| --- |
| <isif condition="${!empty(pdict.Basket)}">  <isset name="lineCtnr" value="${pdict.Basket}" scope="page"/>  <iselseif condition="${!empty(pdict.Order)}">  <isset name="lineCtnr" value="${pdict.Order}" scope="page"/>  </isif>  <isif condition="${!empty(pdict.checkoutstep)}"> |

* Replace the line with below line <isif condition="${checkoutstep <= 5}">

|  |
| --- |
| <isif condition="${checkoutstep <= 6}"> |

* Replace pdict.Basket with lineCtnr at below places

|  |
| --- |
| <isif condition="${lineCtnr.productLineItems.size() == 0 && lineCtnr.giftCertificateLineItems.size() == 1}">  <isset name="editUrl" value="${URLUtils.url('GiftCert-Edit','GiftCertificateLineItemID', lineCtnr.giftCertificateLineItems[0].UUID)}" scope="page"/>  </isif> |

* Replace the line with below ${Resource.msg('summary.title','checkout',null)} <a class="section-header-note" href="${editUrl}">${Resource.msg('global.edit','locale',null)}</a>

|  |
| --- |
| ${Resource.msg('summary.title','checkout',null)} <isif condition="${!empty(pdict.Basket)}"><a class="section-header-note" href="${editUrl}">${Resource.msg('global.edit','locale',null)}</a></isif> |

* Update the DIV “checkout-mini-cart” with below code

|  |
| --- |
| <div class="checkout-mini-cart">  <isif condition="${checkoutstep != 5 && checkoutstep != 6}">  <isminilineitems p\_lineitemctnr="${lineCtnr}"/>  </isif>  </div> |

* Update the DIV “checkout-order-totals” with below code

|  |
| --- |
| <div class=" checkout-order-totals">  <isif condition="${checkoutstep == 6}">  <isordertotals p\_lineitemctnr="${lineCtnr}" p\_showshipmentinfo="${true}" p\_shipmenteditable="${false}" p\_totallabel="${Resource.msg('global.ordertotal','locale',null)}"/>  <iselseif condition="${checkoutstep > 3}">  <isordertotals p\_lineitemctnr="${lineCtnr}" p\_showshipmentinfo="${true}" p\_shipmenteditable="${true}" p\_totallabel="${Resource.msg('global.ordertotal','locale',null)}"/>  <iselse/>  <isordertotals p\_lineitemctnr="${lineCtnr}" p\_showshipmentinfo="${false}" p\_shipmenteditable="${false}" p\_totallabel="${Resource.msg('global.estimatedtotal','locale',null)}"/>  </isif>  </div> |

###### Update “minshipments.isml”

* Replace this line <isset name="Shipments" value="${pdict.Basket.shipments}" scope="page"/> with below code snippet

|  |
| --- |
| <isif condition="${!empty(pdict.Basket)}">  <isset name="lineCtnr" value="${pdict.Basket}" scope="page"/>  <isset name="Shipments" value="${lineCtnr.shipments}" scope="page"/>  <iselseif condition="${!empty(pdict.Order)}">  <isset name="lineCtnr" value="${pdict.Order}" scope="page"/>  <isset name="Shipments" value="${pdict.Order.shipments}" scope="page"/>  </isif> |

* Replace pdict.Basket with lineCtnr at below places

|  |
| --- |
| <isif condition="${shipment.productLineItems.length <= 0 || shipment.custom.shipmentType == null && shipment.UUID==lineCtnr.defaultShipment.UUID && !empty(shipment.shippingAddress) && empty(shipment.shippingAddress.address1)}"> |
| <isif condition="${Shipments.size() > 1 && lineCtnr.productLineItems.size() > 0}"><div class="name">${Resource.msgf('multishippingshipments.shipment','checkout',null, shipmentCount)}</div></isif> |

* Replace the line with below <a href="${editUrl}" class="section-header-note">${Resource.msg('global.edit','locale',null)}</a> twice in a file

|  |
| --- |
| <iselseif condition="${shipment.custom.shipmentType == 'instore'}"/>  <isset name="editUrl" value="${URLUtils.https('Cart-Show')}" scope="page"/>  <isif condition="${!empty(pdict.Basket)}"><a href="${editUrl}" class="section-header-note">${Resource.msg('global.edit','locale',null)}</a></isif>  ${Resource.msg('cart.store.instorepickup','checkout',null)}  <iselseif condition="${shipment.shippingAddress != null && lineCtnr.productLineItems.size() > 0}"/>  <isif condition="${!empty(pdict.Basket)}"><a href="${editUrl}" class="section-header-note">${Resource.msg('global.edit','locale',null)}</a></isif>  ${Resource.msg('minishipments.shippingaddress','checkout',null)}  </isif> |

* Replace pdict.Basket with lineCtnr at below line

|  |
| --- |
| <iselseif condition="${shipment.shippingAddress != null && lineCtnr.productLineItems.size() > 0}"> |

###### Update “ReportCheckout.isml”

* Add a condition after this <isset name="checkoutname" value="${pdict.checkoutname}" scope="page"/> with below code snippet

|  |
| --- |
| <isset name="LineCntr" value="${pdict.Basket}" scope="page"/>  <isif condition="${!empty(pdict.Basket)}">  <isset name="LineCntr" value="${pdict.Basket}" scope="page"/>  <iselseif condition="${!empty(pdict.Order)}">  <isset name="LineCntr" value="${pdict.Order}" scope="page"/>  </isif> |

* Replace pdict.Basket with LineCntr twice in file along with null check

|  |
| --- |
| 'BasketID', null != LineCntr ? LineCntr.UUID:null, |

##### Core - scss changes

###### Update “\_checkout.scss”

* Add below code snippet at the end of file

|  |
| --- |
| .SecureAcceptance\_IFRAME iframe{  height**:**600px !important**;**  **}**  @media screen and **(** max-width**:**1024px **){**  .SecureAcceptance\_IFRAME iframe**{**  height**:**650px !important**;**  **}**  **}**  @media screen and **(** max-width**:**767px **){**  .SecureAcceptance\_IFRAME iframe**{**  height**:**670px !important**;**  **}**  **}** |

#### Secure Acceptance Silent Post Section

##### Template - billing.isml

Add a a div for secure acceptance silent post after the end of </form> tag

|  |
| --- |
| </form>  <div id=*"secureAcceptancePost"*>  </div> |

Add a “secureacceptance” class inside button and specify type as”button”as below

|  |
| --- |
| <div class=*"form-row form-row-button"*>  <button class=*"button-fancy-large secureacceptance continue-place-order"* type=*"button"* name=*"${pdict.CurrentForms.billing.save.htmlName}"* value=*"${Resource.msg('global.continueplaceorder','locale',null)}"*><span>${Resource.msg('global.continueplaceorder','locale',null)}</span></button>  </div> |

##### Core – footer\_UI.isml

Include script and copy file jquery.payment.js at respective location (app\_storefront\_core/cartridge/static/default/lib/jquery) inside site cartridge from int\_cybersource cartridge

|  |
| --- |
| <script src="${URLUtils.staticURL('/lib/jquery/jquery.validate.min.js')}" type="text/javascript"></script>  <script src="${URLUtils.staticURL('/lib/jquery/jquery.payment.js')}" type="text/javascript"></script> |

##### Core – Resource.ds

* Add two new Resource in ResourceHelper.getResources

|  |
| --- |
| QUICK\_VIEW\_POPUP : Resource.msg('product.quickview.popup', 'product', null),  INVALID\_SERVICE : Resource.msg('checkout.getsignature.service.problem', 'checkout', null),  INVALID\_CREDITCARD : Resource.msg('checkout.invalid.credit.card.info', 'checkout', null) |

* Add below line under ResourceHelper.getUrls

|  |
| --- |
| ,  silentpost : URLUtils.https('SECURE\_ACCEPTANCE-GetRequestDataForSilentPost').toString() |

##### checkout.properties

Add following values in checkout.properties file for visa checkout changes.

|  |
| --- |
| silentpost.invalidRequestData.error = missing request parameters  silentpost.testCreateToken.error = System Exception occured contact administrator |

##### Core - billing.js

###### Create new “secureacceptance” on Click function

Create a new secure acceptance silent post function to handle credit card information using Ajax call above this function $couponCode.on('keydown', function (e) {

|  |
| --- |
| $('.secureacceptance').on('click', **function** (e) {    **var** $selectPaymentMethod = $('.payment-method-options');  **var** selectedPaymentMethod = $selectPaymentMethod.find(':checked').val();  **if** ('SA\_SILENTPOST' == selectedPaymentMethod) {  **var** $checkoutForm = $('.checkout-billing');  **var** ccnumber = $($checkoutForm).find('input[name$="\_creditCard\_number"]').val();  **var** cvn = $($checkoutForm).find('input[name$="\_creditCard\_cvn"]').val();  **var** month = $('.payment-method-expanded .month select').val();  **var** expyear = $('.payment-method-expanded .year select').val();  **var** dwcctype = $('.payment-method-expanded .cctype select').val();  var savecc = $($checkoutForm).find('input[name$="\_creditCard\_saveCard"]').is(':checked');  **var** customerEmail = $("#dwfrm\_billing\_billingAddress\_email\_emailAddress").val();  **var** cardmap= {'Visa': '001','Amex': '003','MasterCard': '002','Discover': '004','Maestro':'042'};  **if**(month.length == 1) {  month = "0"+month;  }  **var** cctype = cardmap[dwcctype];    **var** firstname = encodeRequestFieldValue($($checkoutForm).find('input[name$="\_addressFields\_firstName"]').val());  **var** lastname = encodeRequestFieldValue($($checkoutForm).find('input[name$="\_addressFields\_lastName"]').val());  **var** address1 = encodeRequestFieldValue($($checkoutForm).find('input[name$="\_addressFields\_address1"]').val());  **var** address2 = encodeRequestFieldValue($($checkoutForm).find('input[name$="\_addressFields\_address2"]').val());  **var** city = encodeRequestFieldValue($($checkoutForm).find('input[name$="\_addressFields\_city"]').val());  **var** zipcode = encodeRequestFieldValue($($checkoutForm).find('input[name$="\_addressFields\_postal"]').val());  **var** country = encodeRequestFieldValue($($checkoutForm).find('select[name$="\_addressFields\_country"]').val());  **var** state = encodeRequestFieldValue($($checkoutForm).find('select[name$="\_addressFields\_states\_state"]').val());  **var** phoneno = encodeRequestFieldValue($($checkoutForm).find('input[name$="\_addressFields\_phone"]').val());  **var** cctoken = encodeRequestFieldValue($('[data-method="CREDIT\_CARD"]').find('[name$="creditCard\_selectedCardID"]').val());    **var** validCardType = dwcctype.toLowerCase();  **var** validCardNumber = $.payment.validateCardNumber(ccnumber);  **var** validCardCvv= $.payment.validateCardCVC(cvn,validCardType);  **var** validCardExp = $.payment.validateCardExpiry(month, expyear);    **if**(cctoken) {  validCardNumber = **true**;  }    $($checkoutForm).find('input[name$="\_creditCard\_number"]').val("");  $($checkoutForm).find('input[name$="\_creditCard\_cvn"]').val("");  $($checkoutForm).find('input[name$="\_creditCard\_expiration\_month"]').val("");  $($checkoutForm).find('input[name$="\_creditCard\_expiration\_year"]').val("");  $($checkoutForm).find('input[name$="\_creditCard\_type"]').val("");    **if**(validCardCvv && validCardExp && validCardNumber) {  **var** data = {  custemail : customerEmail,  savecc : savecc,  firstname : firstname,  lastname : lastname,  address1 : address1,  address2 : address2,  city : city,  zipcode : zipcode,  country : country,  state : state,  phone : phoneno,  cctoken : cctoken,  format : 'ajax'  };  $.ajax({  url: Urls.silentpost,  type: "POST",  data: data,  success: **function**(xhr,data) {  **if**(xhr) {  **if**(xhr.error == **true**) { $("#saspCardError").html(xhr.errorMsg);  $("#saspCardError").addClass('error');  }  **else** { $("#secureAcceptancePost").html(xhr);  $("#card\_expiry\_date").val(month +'-'+expyear); $("#card\_type").val(cctype);  $("#card\_cvn").val(cvn); **if**(cctoken == **null** || cctoken == '') {  $('#silentPostFetchToken').append('<input type="hidden" id="card\_number" name="card\_number" />'); $("#card\_number").val(ccnumber);  } $("#silentPostFetchToken").submit();  }  }  **else** {  $("#saspCardError").html(Resources.INVALID\_SERVICE);  $("#saspCardError").addClass('error');  }  },  error: **function** () { $("#saspCardError").html(Resources.INVALID\_SERVICE).addClass('error');  }  });  }  **else**{  $("#saspCardError").html(Resources.INVALID\_CREDITCARD);  $("#saspCardError").addClass('error');    **return** **false**;  }  }  **else**{  $('.secureacceptance').prop("type", "submit").submit();  **return** **true**;  }  }); |

###### Create new “encodeRequestFieldValue” function

Create a new function to encode input field value below setCCFields :

|  |
| --- |
| /\*\*  \* **@function**  \* **@description** function to convert html tag to lt or gt;  \* **@param** {fieldValue} value of the field  \*/  **function** encodeRequestFieldValue(fieldValue) {    **return** fieldValue.replace(/</g, "&lt;").replace(/>/g, "&gt;")  } |

###### Update “updatePaymentMethod “function

* Update the function:

|  |
| --- |
| **function** updatePaymentMethod(paymentMethodID) {  **var** $paymentMethods = $('.payment-method');  $paymentMethods.removeClass('payment-method-expanded');  **var** dataMethod = paymentMethodID;  **if** (paymentMethodID=='SA\_SILENTPOST') {  dataMethod = 'CREDIT\_CARD';  }  **var** $selectedPaymentMethod = $paymentMethods.filter('[data-method="' + dataMethod + '"]');  **if** ($selectedPaymentMethod.length === 0) {  $selectedPaymentMethod = $('[data-method="Custom"]');  }  **if** (paymentMethodID=="CREDIT\_CARD" || paymentMethodID=="SA\_SILENTPOST") {  $(".spsavecard").show();  } **else** **if** ((paymentMethodID=="SA\_REDIRECT" || paymentMethodID=="SA\_IFRAME") && SitePreferences.TOKENIZATION\_ENABLED) {  $(".spsavecard").show();  }  else {  $(".spsavecard").hide();  } |

# Device Fingerprint

The device fingerprint enables CyberSource to detect fraud/spam more efficient.  
The device fingerprint can be used as an addition of the Credit Card Payment, it is not an independent service.

## How does it work?

During/before checkout three (invisible) ‘beacons’ at the checkout page (a JavaScript, an image and a flash object) would collect and transmit several client-specific parameters to CyberSource partner.

Those beacons contain the session Id.

With the Credit Card Payment, this session Id is transmitted again and CyberSource is able to combine the data for advanced fraud detection.

## Setup:

(Prerequisites: CyberSource cartridge is already installed).

1. Enable the device fingerprint at the Site Preferences of CyberSource and set the Organization ID (provided by CyberSource). The Merchant ID should be set already, anyway.
2. Include following snippet i.e. at the billing.isml page (Recommended: at bottom of page to have no visual impacts)

<isifcondition=*”${dw.system.Site.getCurrent().getCustomPreferenceValue(‘CsDeviceFingerprintEnabled’)}”*>

<isincludeurl=*”${URLUtils.url(‘Cybersource-IncludeDigitalFingerprint’)}”*/>

</isif>

Do a checkout with Credit Card payment. After this checkout, at the CyberSource Business Manager you will see (at the Transaction Manager):

*Device Fingerprint: submitted*

### Hints for the CsDeviceFingerprintRedirectionType:

To get improved deviceFingerprint results, Cybersource recommends redirecting the included code (loading a image, a flash and a javascript) pointing to the CsJetmetrixLocation, to a local domain.

There are three possible settings for this redirection: ‘none’, static’ and dynamic.

No redirection, the beacons will be loaded direct from the CsJetmetrixLocation (i.e. https://h.online-metrix.net)

*Static* The beacons are included with aemandware pipeline call. The pipelinecall will redirect to the CsJetmetrixLocation.

Dynamic If set to dynamic, you have to specify a mapping rule at SiteUrls->Static Mappings.

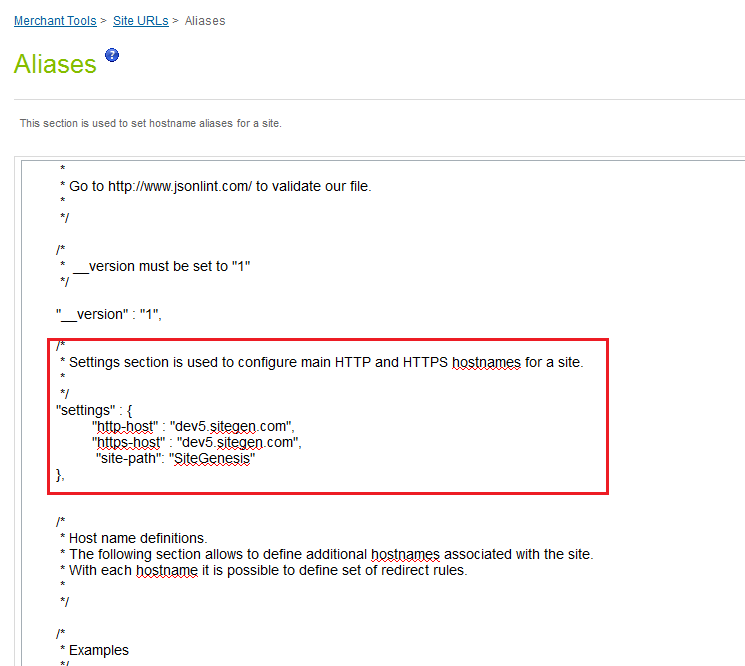
All URLs matching the pattern will be redirected by the Demandware Server.



Example for a matching mapping rule for the device fingerprint redirection

Make an Alias entry in Business manager to execute Device finger print with “ Dynamic” redirection Type

Go to Site > Site URLs > Aliases and add an Alias for your domain like below:



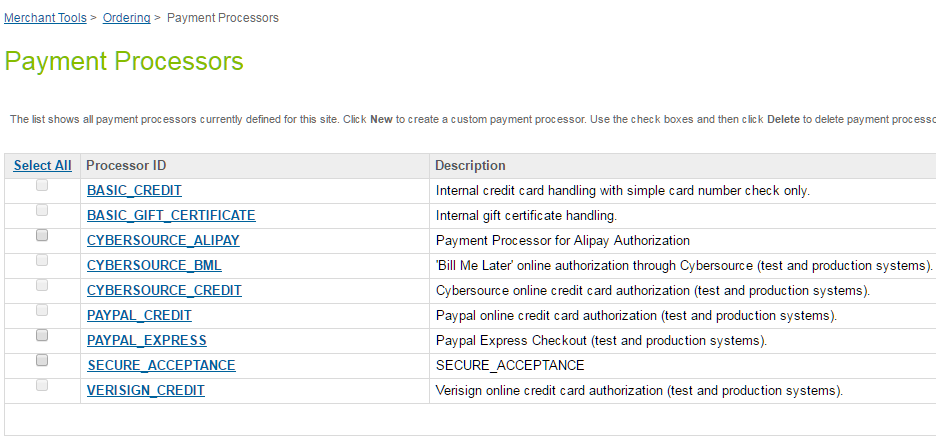
## Site Configuration

### Configure Payment Processor

#### Steps to Create payment processor

Go to Site -> Ordering -> Payment Processors, add a new payment processor with ID and description as given in below table

|  |  |
| --- | --- |
| Processor ID | Description |
| [CYBERSOURCE\_ALIPAY](https://cybersource02.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/ViewPaymentMethod_52-Show?PaymentMethodUUID=2e465f83282638559e5b5d8909&ChannelID=0529909f885a5606e75e831d7c) | [CYBERSOURCE\_ALIPAY](https://cybersource02.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/ViewPaymentMethod_52-Show?PaymentMethodUUID=2e465f83282638559e5b5d8909&ChannelID=0529909f885a5606e75e831d7c) (test and production systems). |
| [CYBERSOURCE\_CREDIT](https://cybersource02.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/ViewPaymentMethod_52-Show?PaymentMethodUUID=d957151d6c54ebbe0d99533b91&ChannelID=0529909f885a5606e75e831d7c) | Cybersource online credit card authorization and visa checkout (test and production systems). |
| [PAYPAL\_EXPRESS](https://cybersource02.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-Site/default/ViewPaymentMethod_52-Show?PaymentMethodUUID=790b5cbacfad50e9ea68f2a970&ChannelID=0529909f885a5606e75e831d7c) | Paypal Express Checkout and from billing (test and production systems). |



### Import Meta Data

Import following site configuration meta-data through Business Manager:

To import the following site configuration Go to Administration -> Site Development -> Import & Export -> upload the below mentioned files and import the configuration.

* /int\_cybersource/configuration/ Cybersource-metadata.xml – sets all the required meta configurations of system defined and custom defined

### Import Payment Methods

To import the following site payment methods Go to Site > Ordering > Import & Export-> upload the below mentioned file and import the configuration in to Payment Methods.

* /int\_cybersource/configuration/Cybersource\_ Payment\_Method.xml
* Mechant can enable/disable any of the payment method listed below:

|  |  |
| --- | --- |
| Payment Method ID | Payment Method Name |
| ALIPAY | Alipay |
| CREDIT\_CARD | Credit Card |
| DW\_APPLE\_PAY | Apple Pay |
| PayPal | Pay Pal |
| SA\_IFRAME | Secure Acceptance Iframe |
| SA\_REDIRECT | Secure Acceptance Redirect |
| SA\_SILENTPOST | Silent Post |
| VISA\_CHECKOUT | Visa Checkout |

### Configure Services

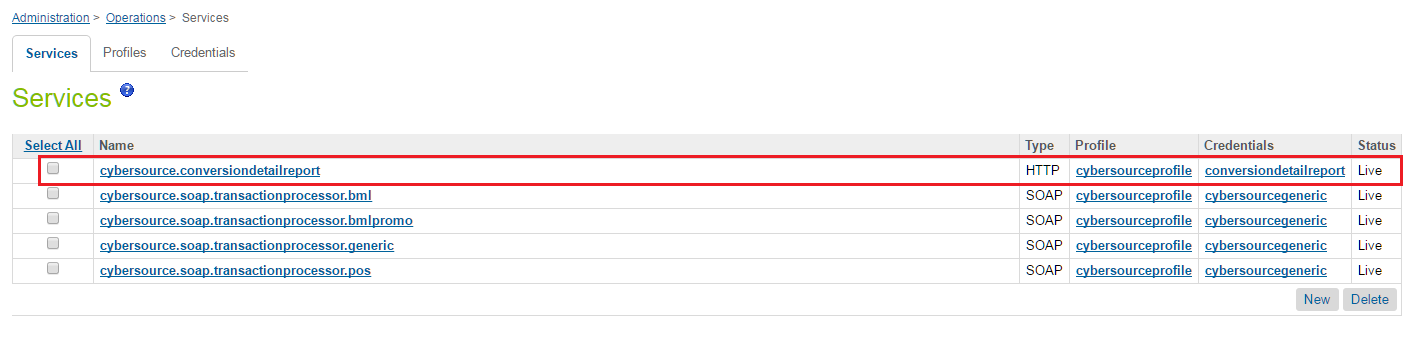
To import the following Service configuration Go to Administration > Operations > Import & Export-> upload the below mentioned file and import the configuration under services

* /int\_cybersource/configuration/CyberSource\_Services.xml – add new Service for cybersource integration

After import above file ensure to update credentials as per cybersource merchant account appropriately in BM.

The following Business Manager Screenshot depicts the import / Export functionality:





* The below Cybersource Services created with single profile and credential
  1. Cybersource.soap.transactionprocessor.generic
  2. Cybersource.soap.transactionprocessor.pos
  3. Cybersource.conversiondetailreport

The profile name is cybersourceprofile, the merchant can create new profile if they require separate profile settings for each service stated above.

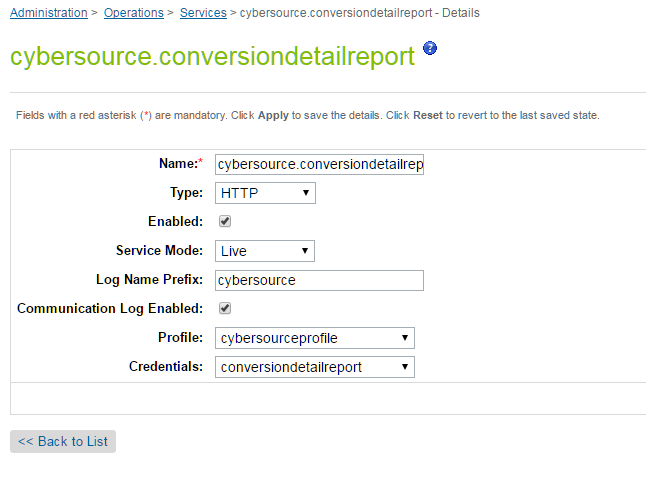
Similarly, merchant can create or update existing credential settings for each service stated above.

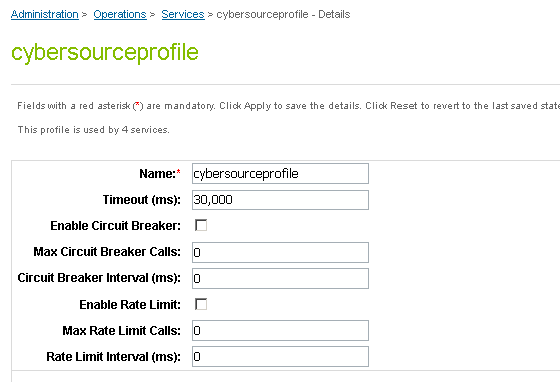
There is Cyber Source detailed report service created in DemandWare with below separate Credentials as follows:

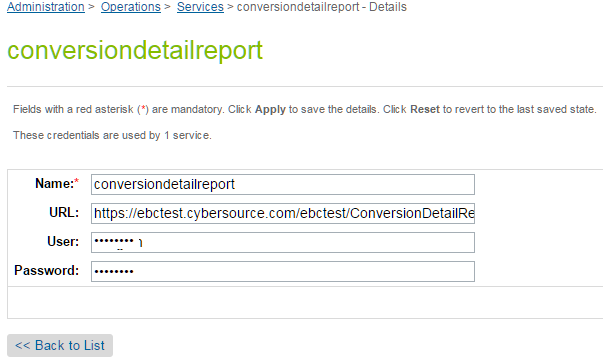
1. URL: Specify below report location along with the requested parameter ,the parameter values are replaced at runtime by the JOB code
   * Test environment URL is "https://ebctest.cybersource.com/ebctest/ConversionDetailReportRequest.do?merchantID={merchantID}&userName={userName}&password={password}&startDate={startDate}&startTime={startTime}&endDate={endDate}&endTime={endTime}"
   * Production environment URL is "https://ebc.cybersource.com/ebc/ConversionDetailReportRequest.do?merchantID={merchantID}&userName={userName}&password={password}&startDate={startDate}&startTime={startTime}&endDate={endDate}&endTime={endTime}"
2. User: Merchant specific username [Represents user having report downloader role in cybersource console]
3. Password: Merchant specific password

* Modify the merchant name, timeout details in profile. Also merchant can configure different profiles for different cybersource services depending on need of the project.

Refer below:







### Configure Site Preferences

#### CyberSource site preference

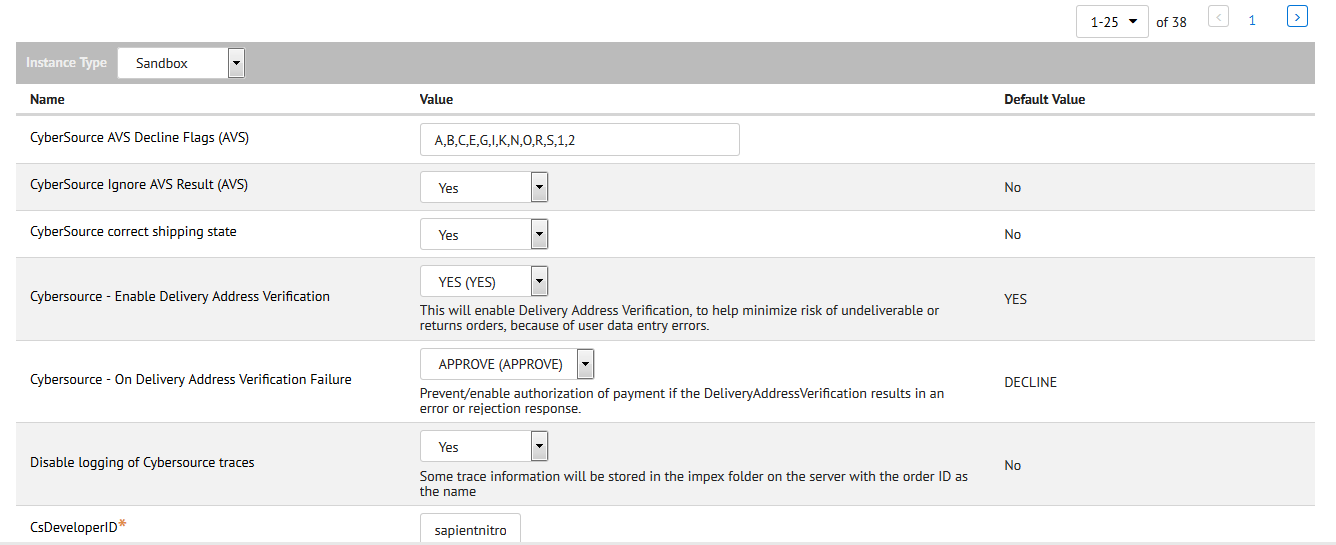
##### Site Preferences Attribute

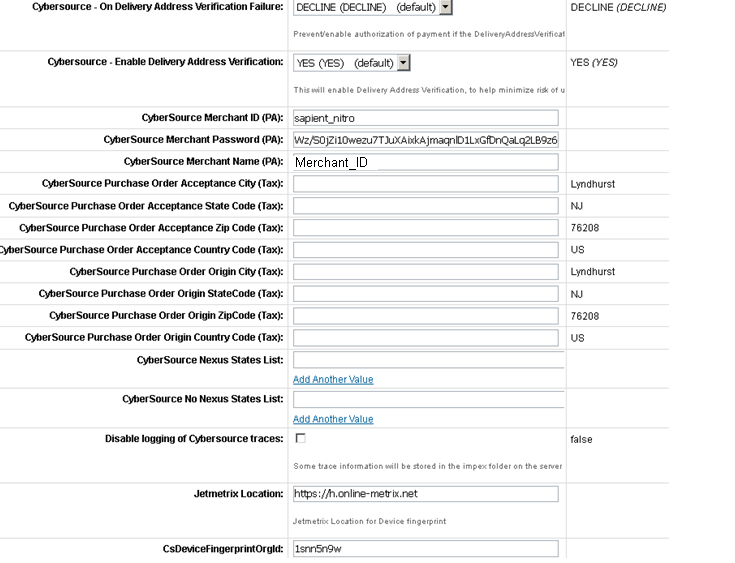
|  |  |
| --- | --- |
| Site Preferences | Description |
| CyberSource Merchant Id(CsMerchantId) | CyberSource Merchant ID |
| CyberSource Merchant Key(CsSecurityKey) | CyberSource Security Key |
| CyberSourceEndpoint(CsEndpoint) | CyberSource Web service End points: Test https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessorProdhttps://ics2wsa.ic3.com/commerce/1.x/transactionProcessor |
| CyberSourceShipFromCity(CsShipFromCity) | Ship to data if fixed for the site |
| CyberSourceShipFromStateCode(CsShipFromStateC)ode | Ship to data if fixed for the site |
| CyberSourceShipFromZipCode(CsShipFromZipCode) | Ship to data if fixed for the site |
| CyberSourceShipFrom Country Code(CsShipFromCountryCode) | Ship to data if fixed for the site |
| CyberSource Ignore AVS Result(CsAvsIgnoreResult) | AVS ignore results |
| CyberSource AVS Decline Flags(CsAvsDeclineFlags) |  |
| Cybersource – On Delivery Address Verification Failure(CsDavOnAddressVerificationFailure) |  |
| Cybersource – Enable Delivery Address Verification(CsDavEnable) | This will enable Delivery Address Verification, to help minimize risk of undeliverable or returns orders, because of user data entry errors. |
| CyberSource Merchant ID(CsPaMerchantId) | Payer Auth merchant ID |
| CyberSource Merchant Password(CsPaMerchantPassword) | Payer Auth Merchant Key |
| CyberSource Merchant Name(CsPaMerchantName) | Name |
| CyberSource Purchase Order Acceptance City(CsPoaCity) | CyberSource purchase order acceptance data – used by Tax |
| CyberSource Purchase Order Acceptance State Code(CsPoaStateCode) | CyberSource purchase order acceptance data – used by Tax |
| CyberSource Purchase Order Acceptance Zip Code(CsPoaZipCode) | CyberSource purchase order acceptance data – used by Tax |
| CyberSource Purchase Order Acceptance Country Code(CsPoaCountryCode) | CyberSource purchase order acceptance data – used by Tax |
| CyberSource Purchase Order Origin City(CsPooCity) | CyberSource purchase order origin data – used by Tax |
| CyberSource Purchase Order Origin StateCode(CsPooStateCode) | CyberSource purchase order origin data – used by Tax |
| CyberSource Purchase Order Origin ZipCode(CsPooZipCode) | CyberSource purchase order origin data – used by Tax |
| CyberSource Purchase Order Origin Country Code(CsPooCountryCode) | CyberSource purchase order origin data – used by Tax |
| CyberSource Nexus States List(CsNexus) | CyberSource nexus state list |
| CyberSource No Nexus States List(CsNoNexus) | CyberSource no nexus state list |
| Disable logging of Cybersource traces(CsDebugCybersource) | To enable/disable debugging |
| Cybersource Device Fingeprintenabled(CsDeviceFingerprintEnabled) | To enable / disable the device fingerprint for advanced fraud detection |
| JetmetrixLocation(CsJetmetrixLocation) | Location of device fingerprint service |
| CsDeviceFingerprintOrgId(CsDeviceFingerprintOrgId) | Id of DeviceFingerprintOrgId |
| Device Fingerprint Redirection(CsDeviceFingerprintRedirectionType) | None,static or dynamic for type of redirection. |
| Cybersource – Enable Tokenization(CsTokenizationEnable) | To enable/disable tokenization call in CC Authorization |
| CyberSource Save Proof.xml(CsPaEnableProofXML) | To enable/disable saving of proof.xml in order object |
| Alipay Payment Type(apPaymentType) | Alipay Payment Type for Domestic as well as International Payment |
| Test Reconciliation ID for Alipay(apTestReconciliationID) | Test Reconciliation ID for Alipay to test initiate and check status services. |
| Decision Manager Enable for Card (csCardDecisionManagerEnable) | Setting to enable/disable decision manager for Credit Card authorization |
| CyberSource correct shipping state (CsCorrectShipState) | Default false, whether expect cybersource to correct the shipping state |
| CyberSource Default Product Tax Code (Tax) (CsDefaultProductTaxCode) | Default 5310 Represent default producttaxcodeif not specified |
| CyberSource Save ParesStatus (PA) (CsPaSaveParesStatus) | Default False Save ParesStatus received as response from Pa Authenticate request and send it as param in ccAuth request call. This field should be enabled after verifying cybersource merchant account settings. |
| Master Card Auth Indicator (csMasterCardAuthIndicator) | Default NonePreauthorization: 0 passed in request Final authorization: 1 passed in request Undefined authorization:omit authIndicator field from the request message |
| CsDeveloperID | Merchant developer Id , mandatory for Cybersource configuration (max limit- String 8 char) |

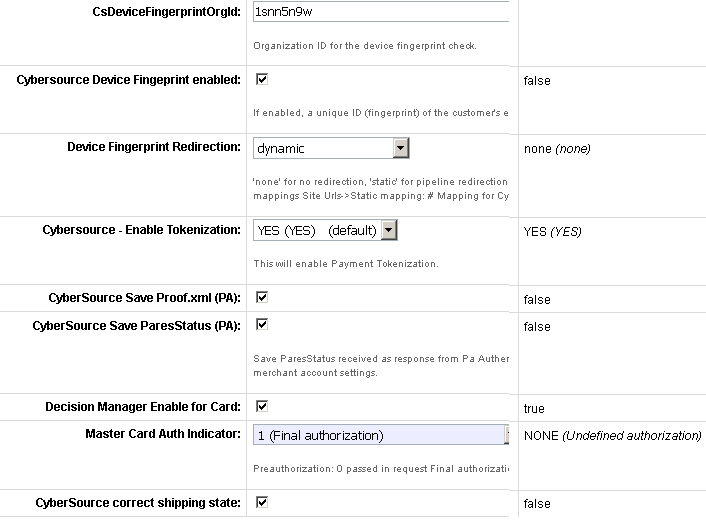
##### Site preference data

Update CyberSource site preference through Business Manager >StoreFront Site> Site Preferences > Custom Preferences.

The screen shot below depicts the site preferences configuration:

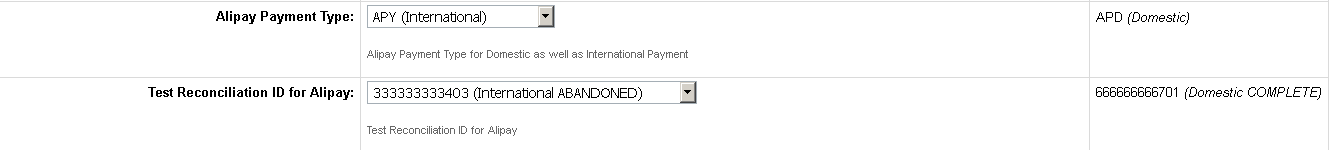






#### Alipay Site Preference

Verify Alipay Site Preferences in already existing custom preferences group “Cybersource”.



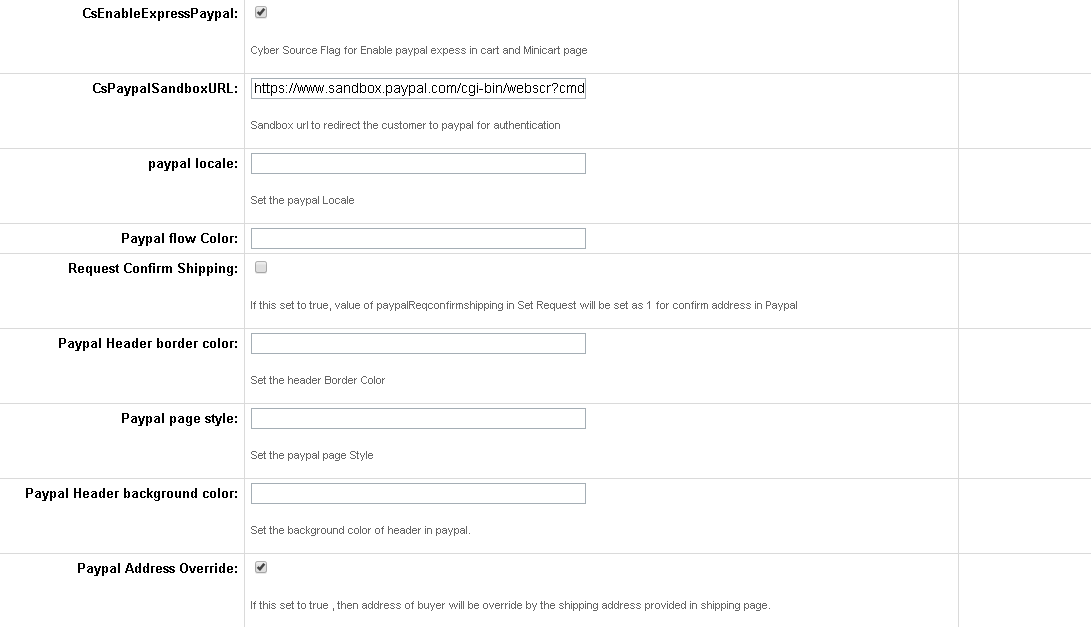
#### PayPal and PayPal Express Checkout Site Preference

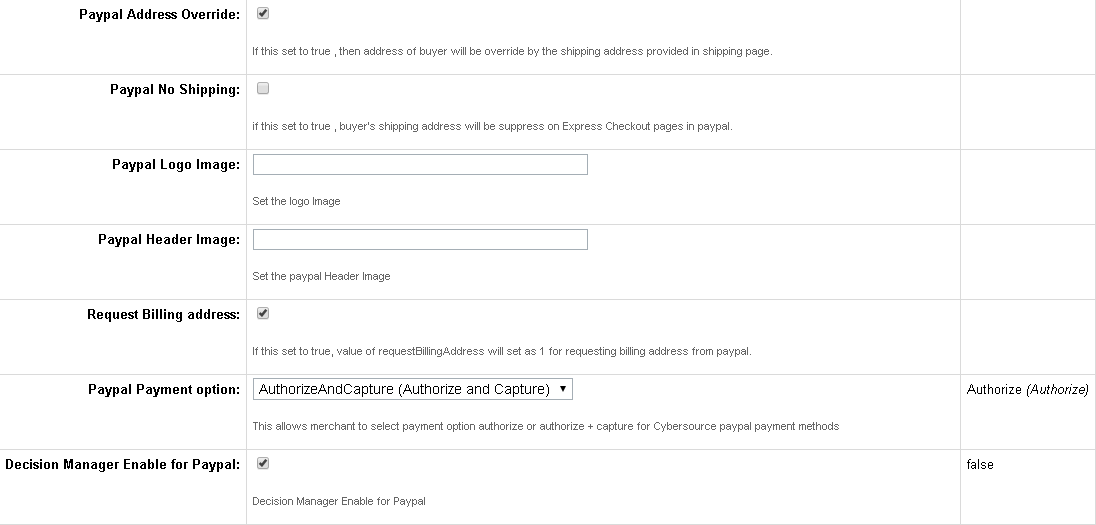
##### Site Preference Attribute

|  |  |  |
| --- | --- | --- |
| **Site Preference** | **Description** | **Default Value** |
| CsEnableExpressPaypal | Cyber Source Flag for Enable paypal expess in cart and Minicart page | NA |
| CsPaypalSandboxURL | Sandbox url to redirect the customer to paypal for authentication | NA |
| CsPaypalLc | Set the paypal Locale | NA |
| CsPaypalPayflowcolor | Set Paypal Flow Color | NA |
| CsPaypalReqconfirmshipping | If this set to true, value of paypalReqconfirmshipping in Set Request will be set as 1 for confirm address in Paypal | NA |
| CsPaypalHdrbordercolor | Set the header Border Color | NA |
| CsPaypalPagestyle | Set the paypal page Style | NA |
| CsPaypalHdrbackcolor | Set the background color of header in paypal. | NA |
| CssPaypalAddressOverride | If this set to true, then address of buyer will be override by the shipping address provided in shipping page. | NA |
| CsPaypalNoshipping | if this set to true , buyer's shipping address will be suppress on Express Checkout pages in paypal. | NA |
| CsPaypalLogoimg | Set the logo Image | NA |
| CsPaypalHdrimg | Set the paypal Header Image | NA |
| CsRequestBillingAddress | If this set to true, value of requestBillingAddress will set as 1 for requesting billing address from paypal. | NA |
| CsPaypalPaymentOption | This allows merchant to select payment option authorize or authorize + capture for Cybersource paypal payment methods | Authorize |
| isDecisionManagerEnable | Decision Manager Enable for Paypal | False |

##### Site Preference Data

Verify Cybersource\_paypal as newly added Custom Preferences Group

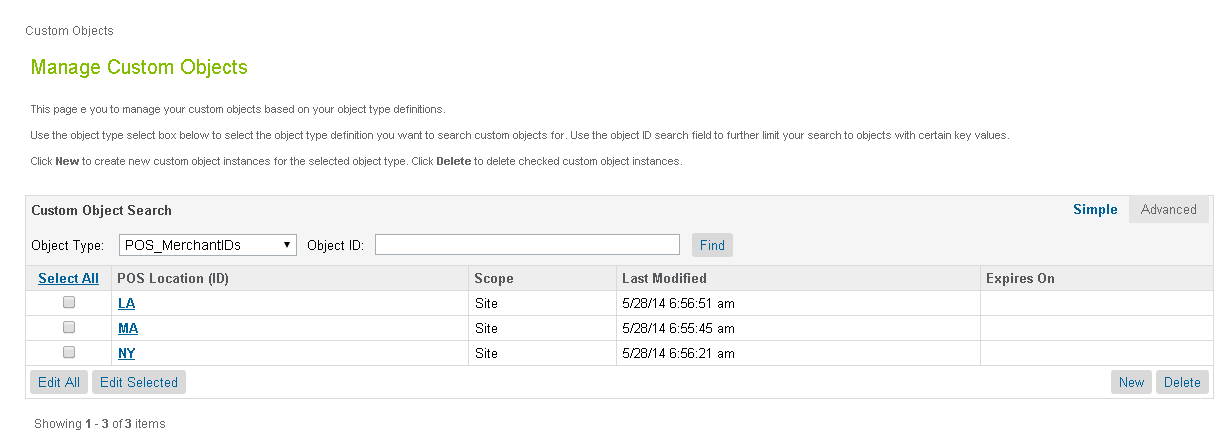


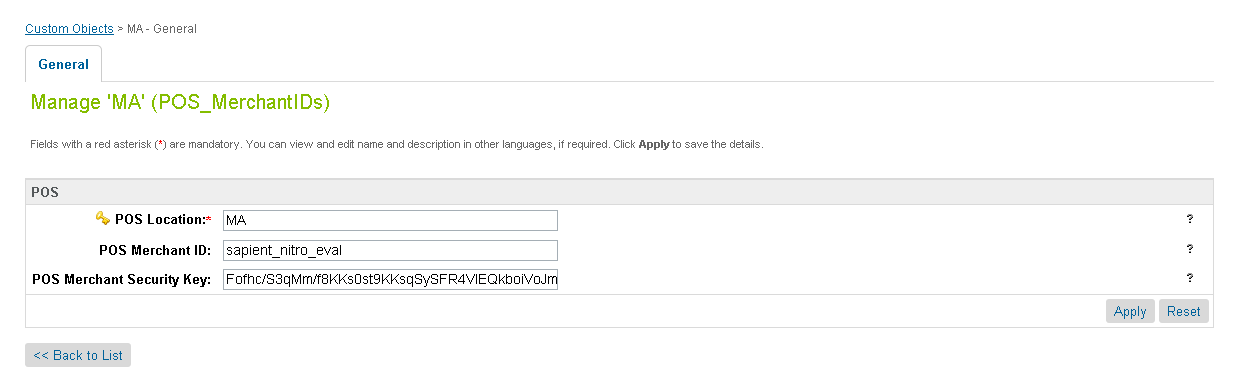


#### Configure Custom Objects for Retail POS

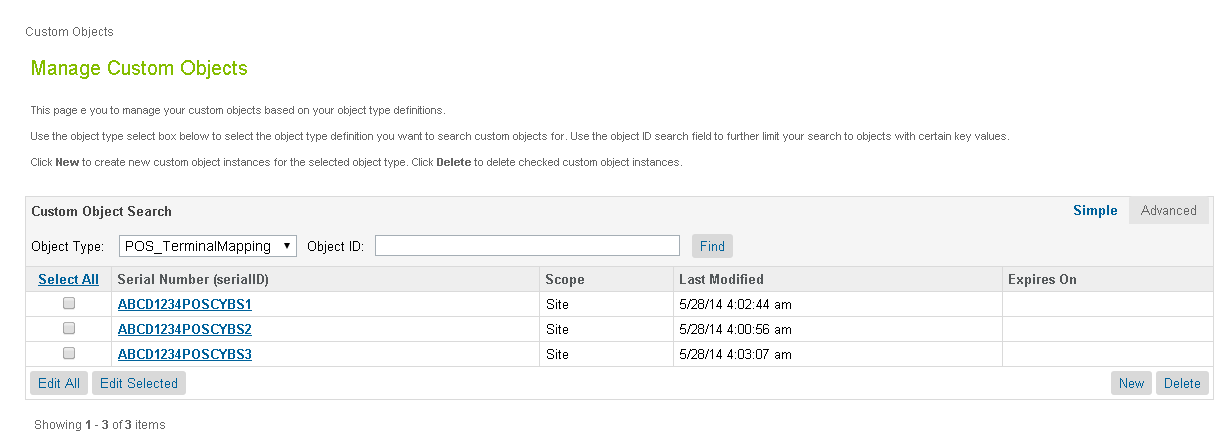
Two custom objects have been added for POS transactions. Ensure to populate these custom objects with merchant specific data. Below are screenshots of sample custom object entry for both custom objects:

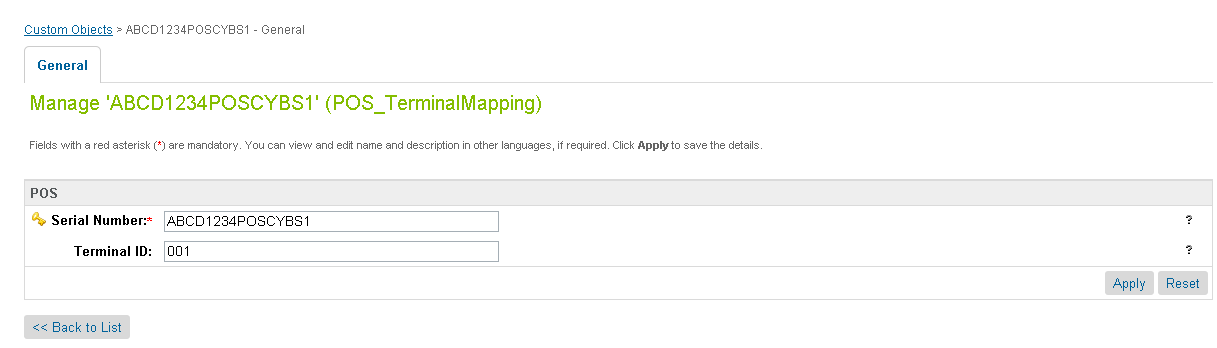
* 1. POS\_MerchantIDs





* 1. POS\_TerminalMapping





#### Visa Checkout site preference

##### Site Preferences Attribute

Below is the list of attributes added in CyberSource site preference:

* Use the sdk.js JavaScript library to control the operation of Visa Checkout [Field name: **cybVisaSdkJsLibrary**]

|  |  |
| --- | --- |
| Environment | Possible values |
| Sandbox | <https://sandbox-assets.secure.checkout.visa.com/checkout-widget/resources/js/integration/v1/sdk.js> |
| LIVE | <https://assets.secure.checkout.visa.com/checkout-widget/resources/js/integration/v1/sdk.js> |

* Use the v-button img class to render a Visa Checkout button that a consumer clicks to initiate a payment [Field name: **cybVisaButtonImgUrl**]

|  |  |
| --- | --- |
| Environment | Possible values |
| Sandbox | <https://sandbox.secure.checkout.visa.com/wallet-services-web/xo/button.png> |
| LIVE | <https://secure.checkout.visa.com/wallet-services-web/xo/button.png> |

* Use below configuration fields for VISA checkout setup and must be different for sandbox and production based on merchant accounts

|  |  |
| --- | --- |
| Field | Description |
| cybVisaExternalProfileId | Use profile's name, created externally by a merchant whom Visa Checkout uses to populate settings, such as accepted card brands and shipping regions. The properties set in this profile override properties in the merchant's current profile. (Alphanumeric; maximum 50 characters) |
| cybVisaAPIKey | The Visa Checkout account API key specified in cybersource business center (Alphanumeric; maximum 100 characters) |
| cybVisaSecretKey | The secret key specified VISA Checkout account profile |

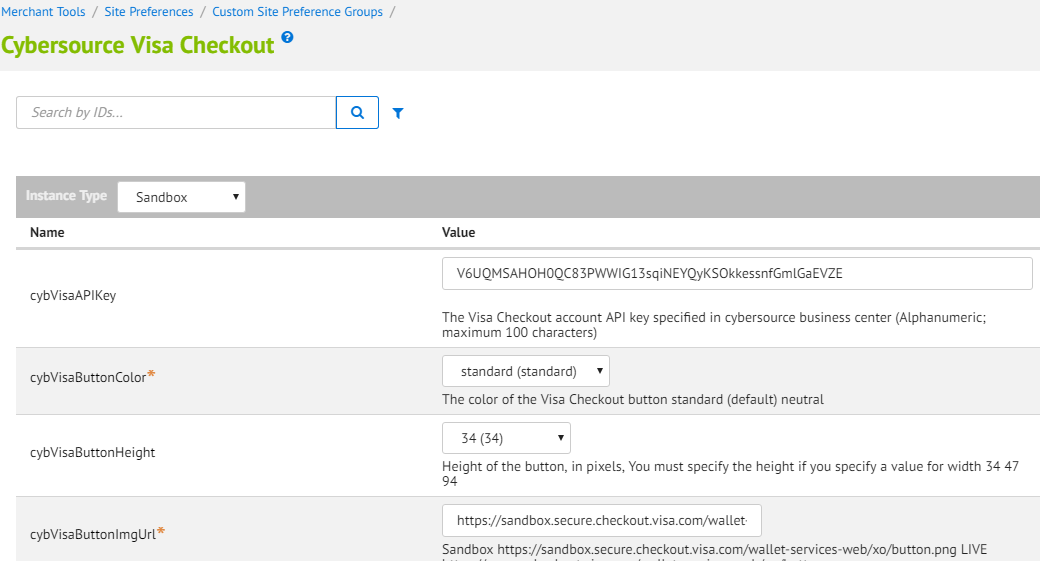
* Use below configuration fields for VISA checkout features and can be kept same for sandbox and production

|  |  |  |
| --- | --- | --- |
| Field | Description | Possible Values |
| cybVisaButtonSize | You can either specify size to display a standard size button, or you can specify height and width to specify a custom size. If you do not specify size or both height and width, the button size is 213 pixels. If you specify height or width, the value of size is ignored | 154 - small  213 - medium (default)  425 - High resolution or large |
| cybVisaButtonHeight | Height of the button, in pixels, You must specify the height if you specify a value for width | 34  47  94 |
| cybVisaButtonWidth | Width of the button, in pixels, You must specify the width if you specify a value for height | -less than 477 if height is 34; default value is 154  -greater than 212 and less than 659 if height is 47; default value is 213  -greater than 424 and less than 1317 if height is 94; default value is 425 |
| cybVisaButtonColor | The color of the Visa Checkout button | standard (default)  neutral |
| cybVisaCardBrands | Override value for brands associated with card art to be displayed. If a brand matching the consumer's preferred card is specified, the card art is displayed on the button; otherwise, a generic button is displayed | Comma Separated list is accepted  VISA  MASTERCARD  AMEX  DISCOVER |
| cybVisaThreeDSActive | Whether Verified by Visa (VbV) is active for this transaction. If Verified by Visa is configured, you can use threeDSActive to deactivate it for the transaction; otherwise, VbV will be active if it has been configured | false (default)  true |
| cybVisaThreeDSSuppressChallenge | Whether a Verified by Visa (VbV) consumer authentication prompt is suppressed for this transaction. If true, VbV authentication is performed only when it is possible to do so without the consumer prompt. | true - Do not display a consumer prompt.  false - Allow a consumer prompt |
| cybVisaTellMeMoreLinkActive | Indicate whether Tell Me More Link to be displayed with VISA button | true (default)  false |

##### Site preference data

Update CyberSource site preference through Business Manager >StoreFront Site> Site Preferences > Custom Preferences.

The screen shot below depicts the site preferences configuration:

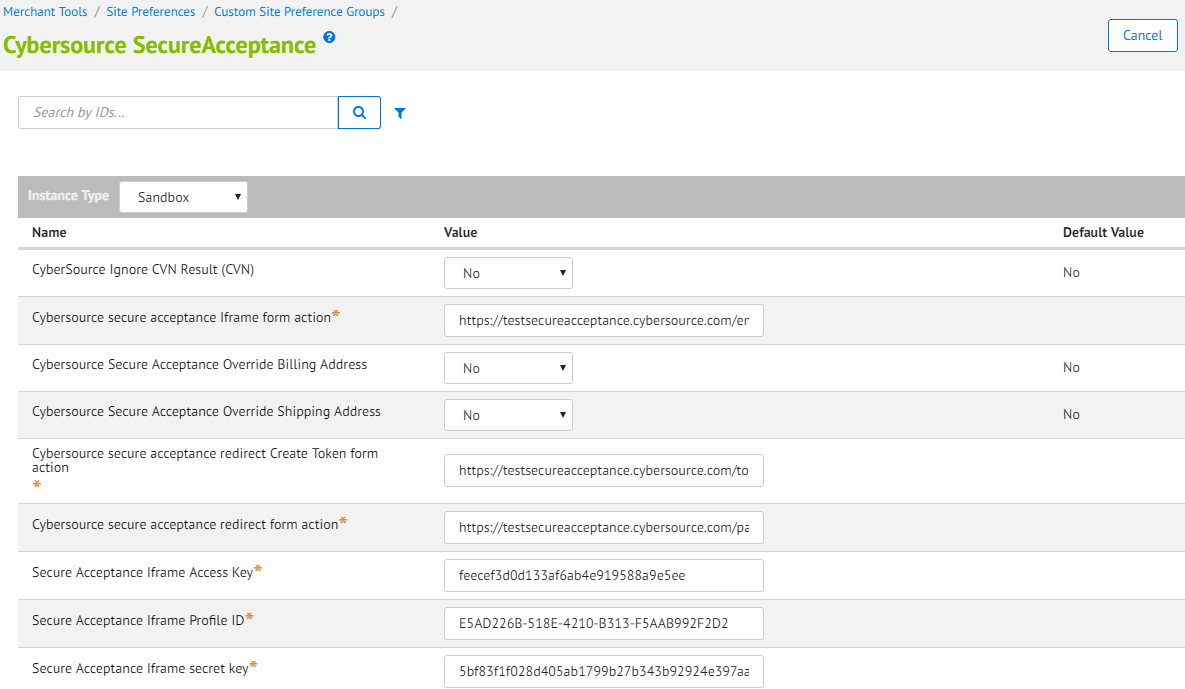


#### Secure Acceptance site preference

##### Site Preferences Attribute

|  |  |  |
| --- | --- | --- |
| Attribute ID | Data Type | Description |
| cybmerchantpostfailure | Boolean | Flag to check for all secure acceptance failed order |
| savecard | Boolean | Set Custom attribute if Customer choose to save card |
| CsSAOverrideBillingAddress | Boolean | Cybersource Secure Acceptance Override Billing Address |
| CsSAOverrideShippingAddress | Boolean | Cybersource Secure Acceptance Override Shipping Address |
| CsCvnDeclineFlags | Boolean | CyberSource Ignore CVN Result (CVN)  [should be in sync with CYB profile cvn flag] |
| SA\_Redirect\_AccessKey | String | Secure Acceptance Redirect Access Key |
| SA\_Redirect\_ProfileID | String | Secure Aceeptance Redirect Profile ID |
| SA\_Redirect\_SecretKey | String | Secure Acceptance Redirect Secret Key |
| SA\_Redirect\_CreateToken\_AccessKey | String | Secure Acceptance Redirect Create Token Access Key |
| SA\_Redirect\_CreateToken\_ProfileID | String | Secure Aceeptance Redirect Create Token Profile ID |
| SA\_Redirect\_CreateToken\_SecretKey | String | Secure Acceptance Redirect Create Token Secret Key |
| SA\_Iframe\_AccessKey | String | Secure Acceptance Iframe Access Key |
| SA\_Iframe\_ProfileID | String | Secure Acceptance Iframe Profile ID |
| SA\_Iframe\_SecretKey | String | Secure Acceptance Iframe secret key |
| SA\_Silent\_AccessKey | String | Secure Acceptance Silent Post Access Key |
| SA\_Silent\_ProfileID | String | Secure Aceeptance Silent Post Profile ID |
| SA\_Silent\_SecretKey | String | Secure Acceptance Silent Post Secret Key |
| CsSARedirectFormAction | String | Cybersource secure acceptance redirect form action |
| CsSARedirectCreateTokenFormAction | String | Cybersource secure acceptance redirect Create Token form action |
| CsSAIframetFormAction | String | Cybersource secure acceptance Iframe form action |
| Secure\_Acceptance\_Token\_Create\_Endpoint | String | Secure Acceptance Token Create Endpoint |
| Secure\_Acceptance\_Token\_Update\_Endpoint | String | Secure Acceptance Token Update Endpoint |

##### Site Preferences Data



#### Enable Payer Authentication for cards

Update credit card preference through Business Manager >StoreFront Site> Ordering> Payment Methods> Credit Card/Debit Cards > Choose cardand then modify Enable Payer Authentication checkbox

The screen shot below depicts the site preferences configuration:



#### Update shipping method preference

Update shipping method preference through Business Manager >StoreFront Site> Ordering> Shipping Methods > Name > CyberSource Shipping ID

The screen shot below depicts the site preferences configuration:

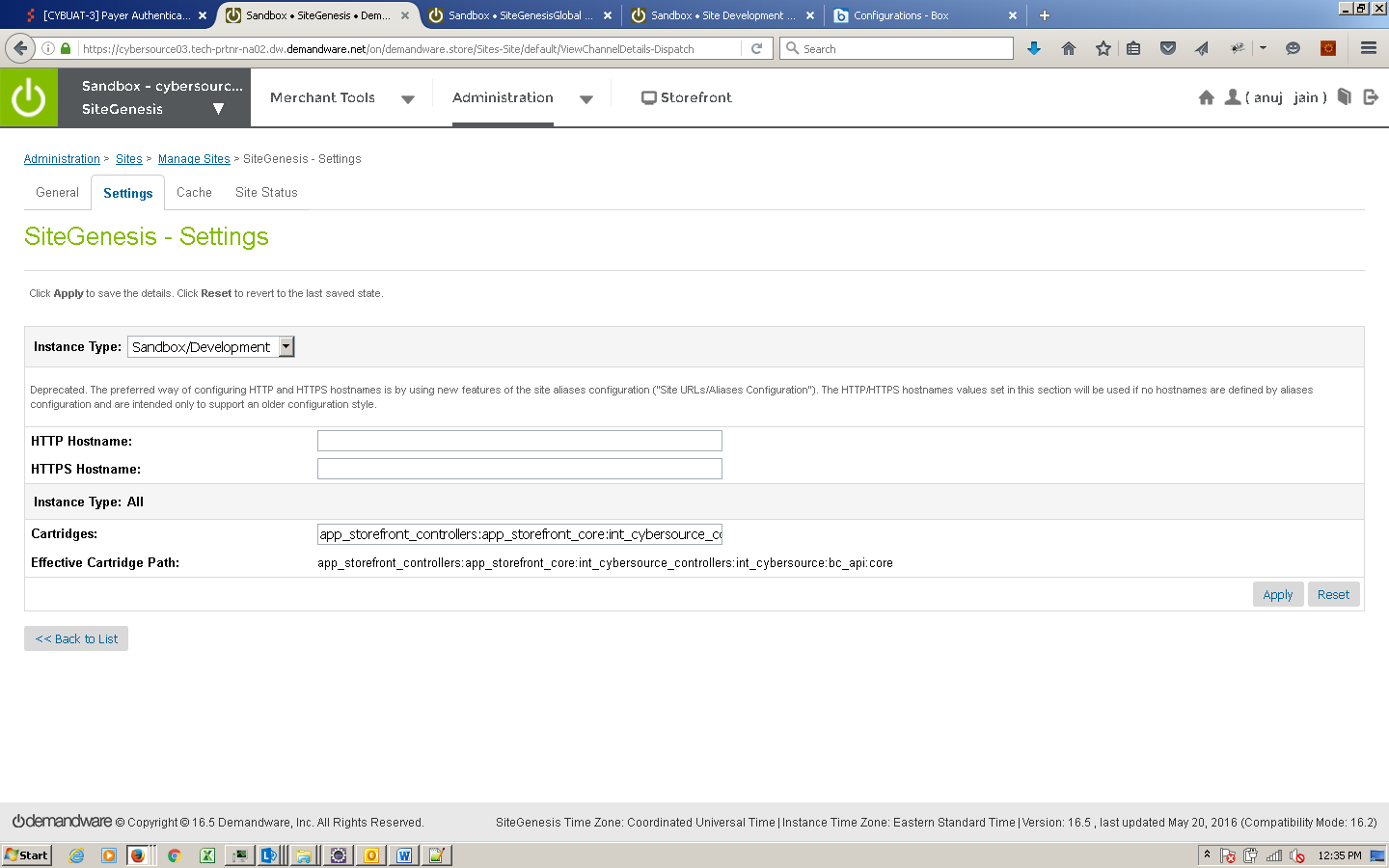


#### Applying CyberSource Cartridge to the Site

Go to the “Administration” in the left hand list to expand the menu and select Sites > Manage Sites link. This will open a list of the active sites on the Demandware platform in your account. Click on the site for which you wish to add the CyberSource cartridge. This will open the General Settings page for that site.

Add int\_cybersource cartridges to the cartridge path for Pipeline integration

Add int\_cybersource\_controllers and int\_cybersource cartridges to the cartridge path for controller integration as depicted in the following screen:



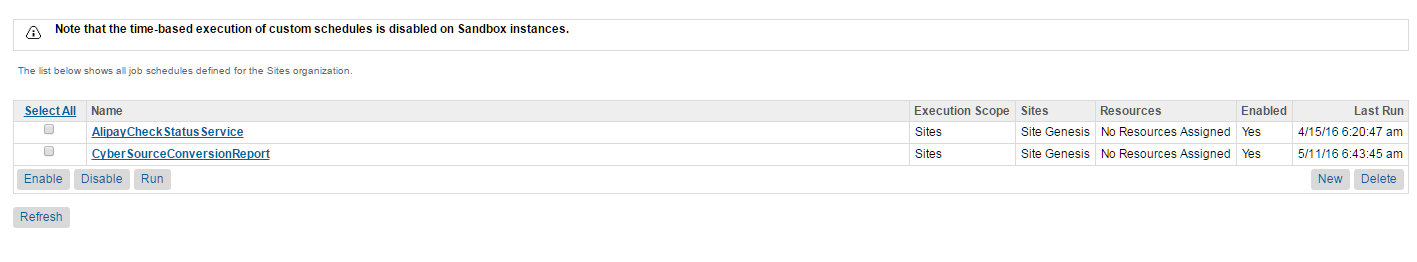
### Batch Jobs

Below are the list of batch jobs need to be configured:

#### Configure Batch Job for AliPay

To import the following Job Schedule configuration Go Adminsistration > Operations > Import & Export-> upload the below mentioned file and import the configuration.

* /int\_cybersource/configuration/Cybersource\_Alipay\_CheckStatusService\_BatchJob.xml – add new batch job for Alipay check status service
* Verify the newly added batch jobs for Alipay Check Status Service
* Go to Administration - > Operations -> Job Schedules



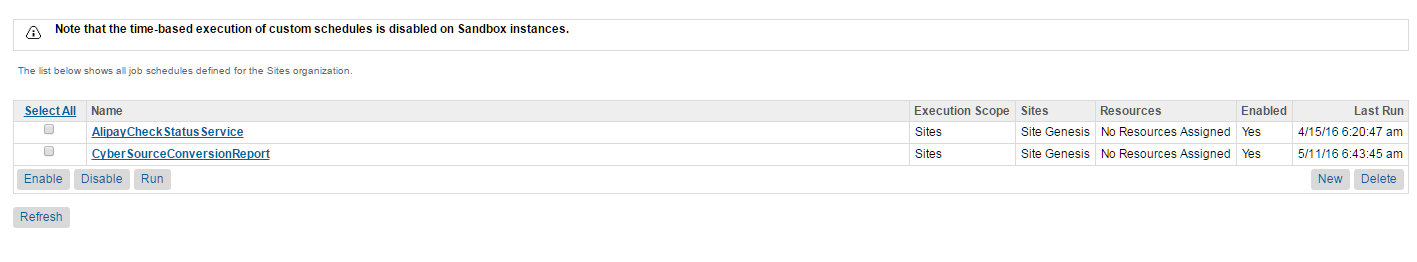
#### Configure Batch Job for Conversion Detail Report

To import the following Job Schedule configuration Go Adminsistration > Operations >  Import & Export-> upload the below mentioned file and import the configuration.

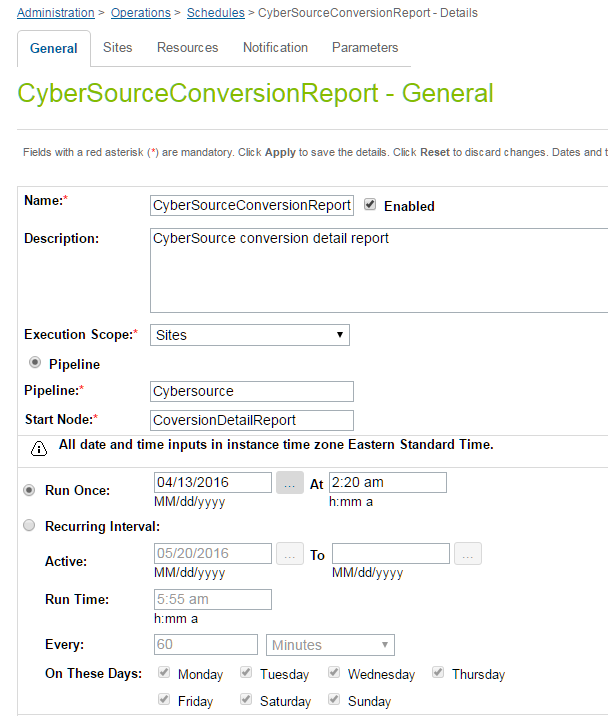
* /int\_cybersource/configuration/CyberSource\_Conversion\_Report\_BatchJob.xml – add new batch job to update order status in BM for CyberSource “Accepted” & “Rejected” orders.

Verify the newly added batch jobs for Conversion detail report service.

Go to Administration - > Operations -> Job Schedules



The batch job created for cybersource conversion detail report specified below, it updates the status of order in demandware which are in CREATED state and mark them as “CANCELLED” for rejected order or “NEW” for accepted order. The accepted orders are marked for “READY FOR EXPORT” as well.

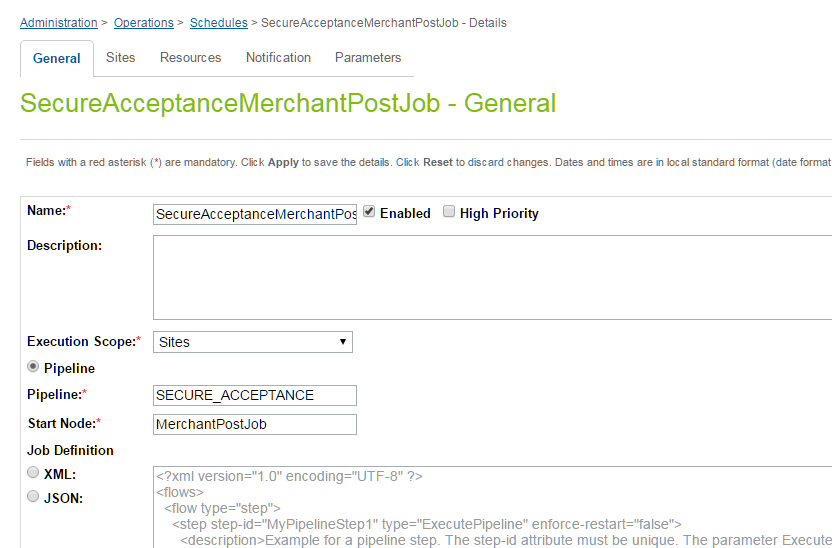


#### Secure Acceptance Merchant Post Batch Job

To import the following Service configuration Go to Administration > Operations > Import & Export-> upload the below mentioned file and import the configuration under services

* /int\_cybersource/configuration/CyberSource\_SecureAcceptanceMerchantPost\_BatchJob.xml – add new Service for secure Acceptance Order update via merchant post notifications

After import above file ensure to update credentials as per cybersource merchant account appropriately in BM.



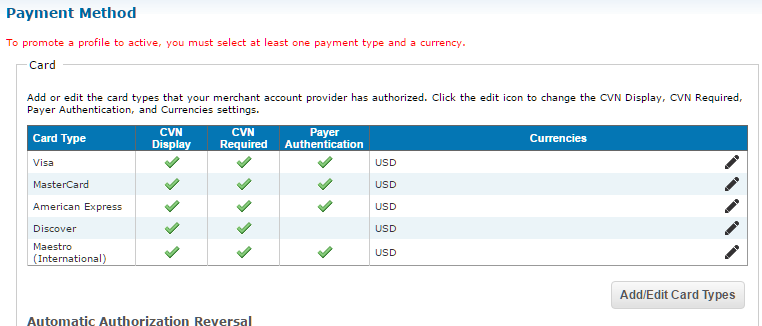
### Secure Acceptance Profile Configuration into CyberSource Business Manager

Secure Acceptance profile settings are configured on CyberSource, along with other settings below are key setting which must be configured in cybersource profiles in order to complete the checkout process successfully.

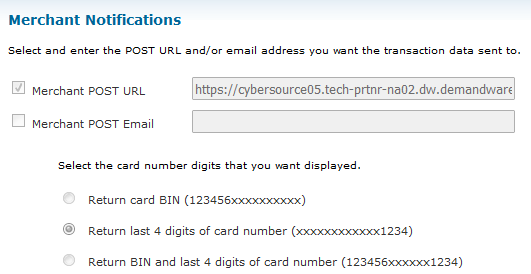
|  |  |  |  |
| --- | --- | --- | --- |
| Profile name | Notification Section  [Merchant post URL] | Custom Response pages | |
| Transaction response page | Cancel Response Page |
| SA Redirect | [Merchant specific URL]/SECURE\_ACCEPTANCE-MerchantPost | [Merchant specific URL]/COPlaceOrder-Submit?provider=saredirect | [Merchant specific URL]/COPlaceOrder-Submit?provider=saredirect |
| SA Iframe | [Merchant specific URL]/SECURE\_ACCEPTANCE-MerchantPost | [Merchant specific URL]/COPlaceOrder-Submit?provider=saiframe | [Merchant specific URL]/COPlaceOrder-Submit?provider=saiframe |
| SA SilentPost | N/A | [Merchant specific URL]/SECURE\_ACCEPTANCE-SilentPostResponse | N/A |

[Ex: <https://cybersource02.tech-prtnr-na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis2-Site/en_US/COPlaceOrder-Submit?provider=saiframe> ]

Only five types of Card are supported in Demanware, so the cards configured in cybersource payment settings should be in sync with Demandware supported cards



Merchant Notifications POST URL card number digits supported option 2 as shown.



## Testing

Use CybersourceUnitTest pipeline to test all the services as follows:

### Authorize Credit Card

Use and modify the CybersourceUnitTest-TestCCAuth pipeline and associated scripts and sub-pipelines. The end node of the unit test pipeline is a template which displays all relevant request/response information in an easy to digest manner. User can change static credit card and address data to observe various responses.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestCCAuth function** with https guard before executing the test case

Example: exports.TestCCAuth=guard.ensure(['https'], TestCCAuth);

### Tax Service

Use and modify the CybersourceUnitTest-TestTax pipeline and associated scripts and sub-pipelines. The script nodes for creating CreateCybersourceShipTo and CreateCybersourceBillTo objects have bindings to produce valid results, but otherwise can be manually modified to test against any domestic or international address.

The end node of the unit test pipeline for taxes is a template which displays all relevant request/response information in an easy to digest manner, to aid the debugging the various response codes and corrected address response.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestTax function** with https guard before executing the test case

Example: exports.TestTax=guard.ensure(['https'], TestTax);

### Address Verification Service (AVS)

Use and modify the CybersourceUnitTest-TestCCAuth pipeline and associated scripts and sub-pipelines. By running simplified payment authorizations with different site preferences set, you can see how the AVS process works and how that result affects the overall payment authorization process.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestCCAuth function** with https guard before executing the test case

Example: exports.TestCCAuth=guard.ensure(['https'], TestCCAuth);

### Delivery Address Verification Service (DAV)

To test stand-alone DAV service, use and/or modify CybersourceUnitTest-TestDAVCheck pipeline and associated scripts and sub-pipelines. Like other test pipelines, test data can be customize to simulate various situations that need to be handled.

Note: Mark the start node as “PUBLIC” before executing the test case

The end node of the unit test pipeline for the stand-alone DAV Service is a template which displays all relevant request/response information in an easy to digest manner, to aid the debugging the various response codes and corrected address response.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestDAVCheck function** with https guard before executing the test case

Example: exports.TestDAVCheck=guard.ensure(['https'], TestDAVCheck);

### Payment Tokenization

Use the Cybersource\_Subscription-Start pipeline to start Subscription creation test suite. By entering test data you can use the various Payment Tokenization related services like Create Subscription, View Subscription, Update Subscription, Delete Subscription, Use Subscription for One Time Payment.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **Cybersource\_Subscription.js functions like Start, CreateSubscription, ViewSubscription, UpdateSubscription, DeleteSubscription, OnDemandPayment** with https guard before executing the test case

Example: exports.CreateSubscription=guard.ensure(['https'], CreateSubscription);

### Full Authorization reversal

Use the Cybersource\_Services-Start pipeline to start Authorization reversal test suite. By entering test data merchant can use the Cybersource Full Authorization Reversal service.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **Cybersource\_Services.js functions like Start, Reversal** with https guard before executing the test case

Example: exports.Reversal=guard.ensure(['https'], Reversal);

### Device Fingerprint

Call the pipelineCybersourceUnitTest-TestFingerprint to test the device Fingerprint Service. A CreditCard Authorization is done and a device fingerprint will be additionally submitted.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestFingerprint function** with https guard before executing the test case

Example: exports.TestFingerprint=guard.ensure(['https'], TestFingerprint);

### Payer Authentication

Call the pipeline CybersourceUnitTest-TestPA to test the Payer Authentication Service.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestPA function** with https guard before executing the test case

Example: exports.TestPA=guard.ensure(['https'], TestPA);

### Retail POS Authorization Request

Call the pipeline CybersourceUnitTesting-StartPOS to test the retail POS Service. This renders a template with a form containing various request fields to enter/select values. The service response is shown after the submit button is clicked. The field’s label turns to red colored font if the field was mandatory.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js StartPOS function** with https guard before executing the test case

Example: exports.StartPOS=guard.ensure(['https'], StartPOS);

### Alipay Initiate Request

Call the pipeline CybersourceUnitTesting-TestAlipayInitiateService to test Alipay Initiate request. Use and modify the mentioned pipeline and associated scripts to test initiate request. The end node of the unit test pipeline is a template which displays all relevant request/response information in an easy to digest manner. User can change static purchase object data and payment type to observe various responses.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestAlipayInitiateService function** with https guard before executing the test case

Example: exports.TestAlipayInitiateService=guard.ensure(['https'], TestAlipayInitiateService);

https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CybersourceUnitTesting-TestAlipayInitiatesService

### Alipay Check Status Request

Call the pipeline CybersourceUnitTesting-TestAlipayCheckStatusService to test Alipay Check Status request. Use and modify the mentioned pipeline and associated scripts to test initiate request. The end node of the unit test pipeline is a template which displays all relevant request/response information in an easy to digest manner. User can change value of requestID either in pipeline assign node or can pass the value in the URL along with pipeline URL to observe various responses. Pass requestId generated by Alipay Initiate service in the URL as mentioned below to test this service.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestAlipayCheckStatusService function** with https guard before executing the test case

Example: exports.TestAlipayCheckStatusService=guard.ensure(['https'], TestAlipayCheckStatusService);

https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CybersourceUnitTesting-TestAlipayCheckStatusService?requestId=”Initiate Request Id”

### Credit Card Capture Request

Call the pipeline CybersourceUnitTesting-TestCaptureCard to test Card Capture request. Use and modify the mentioned pipeline and associated scripts to test capture request. The end node of the unit test pipeline is a template which displays all relevant request/response information in an easy to digest manner. User can change value of orderNo either in pipeline GetOrder pipelet node or can pass the value in the URL as “oid” parameter along with the pipeline URL to observe various responses.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestCaptureCard function** with https guard before executing the test case

Example: exports.TestCaptureCard=guard.ensure(['https'], TestCaptureCard);

https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CybersourceUnitTesting-TestCaptureCard?oid=”Order Number”

### Paypal Capture Request

Call the pipeline CybersourceUnitTesting-TestPaypalCaptureService to test Paypal Capture request. Use and modify the mentioned pipeline and associated scripts to test initiate request. The end node of the unit test pipeline is a template which displays all relevant request/response information in an easy to digest manner. User can change value of orderNo either in pipeline GetOrder pipelet node or can pass the value in the URL along with the pipeline URL to observe various responses. Pass orderNo for order with Demandware Order Status as New and Order Payment Status as Not Paid in the URL as mentioned below to test this service.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **CybersourceUnitTest.js TestPaypalCaptureService function** with https guard before executing the test case

Example: exports.TestPaypalCaptureService=guard.ensure(['https'], TestPaypalCaptureService);

https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CybersourceUnitTesting-TestPaypalCaptureService?orderNo=”Order Number”

### Secure Acceptance Redirect Create Token Test Service Request

Before TESTING please complete the profile setup for service to work refer section **Secure Acceptance profile setup** for more details

Call the pipeline SECURE\_ACCEPTANCE-TestSATokenCreate to test the secure acceptance redirect create token Service. This renders a secure acceptance hosted page at cybersource having details of card options to choose to enter/select values. The service response is shown after the pay button is clicked.  The field’s label turns to red colored font if the field was mandatory. The response arrived to pipeline SECURE\_ACCEPTANCE-TestSATokenCreateResponse which displays the service result fields.

Note: Mark the start node as “PUBLIC” before executing the test case

#### For Controllers

Modify exports of **SECURE\_ACCEPTANCE.js TestSATokenCreate and TestSATokenCreateResponse function** with https guard before executing the test case

Example: exports.TestSATokenCreateResponse =guard.ensure(['https'], TestSATokenCreateResponse);

Example: exports.TestSATokenCreate=guard.ensure(['https'], TestSATokenCreate);

### Apple Pay

#### Rest Interface Testing

The Interface can be tested via any REST client like SOAPUI etc. Below are the steps to test the REST service

1. Install the REST client on machine or browser
2. Hit the secure End Point URL as POST request having merchant site URL for “Cybersource\_ApplePay-Authorize” [example: https://<merchant sandbox>/on/demandware.store/Sites-<merchant site>-Site/default/Cybersource\_ApplePay-Authorize]
3. Add key-value pairs in header for credentials

|  |  |
| --- | --- |
| HEADER KEY | HEADER VALUE |
| dw\_applepay\_user | User is configured by merchant in demandware platform under site preferences |
| dw\_applepay\_password | Password is configured by merchant in demandware platform under site preferences. Further the password to be base64 encode before passing to REST interface |
| Content-Type | application/json |

1. Pass below JSON when Payload test data available

|  |  |
| --- | --- |
| JSON KEY | JSON VALUE |
| orderID | The order ID of ApplePay order object created during checkout journey of ApplePay |
| encryptedPaymentBlob | Encrypted ApplePay blob data returned by ApplePay for PSP to place the order. This contains billing/shipping/card details in encrypted form. |

1. Pass below JSON when Network Token test data available

|  |  |
| --- | --- |
| JSON KEY | JSON VALUE |
| orderID | The order ID of ApplePay order object created during checkout journey of ApplePay |
| networkToken | Network Token returned by ApplePay for PSP authorization (Max length 20 character) |
| cardType | Card Type returned by ApplePay for PSP authorization. Supported types visa/mastercard/amex |
| tokenExpirationDate | Network Token Expiration Date returned by ApplePay for PSP authorization. Format YYMMDD |
| cryptogram | Cryptogram encoded form (max length 40 character) |

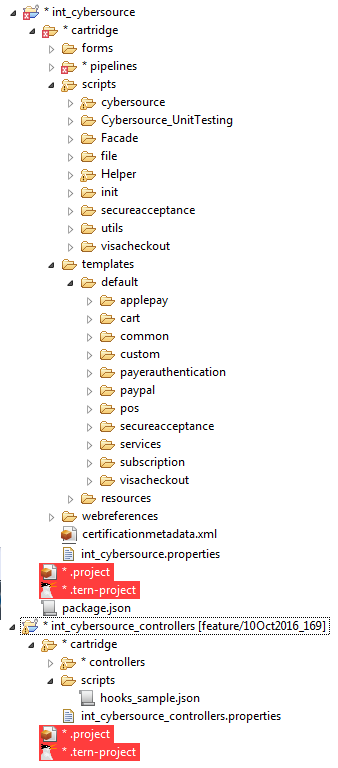
1. Test the Success response JSON

|  |  |
| --- | --- |
| JSON KEY | JSON VALUE |
| TRANSACTION\_RESULT | Below json key-value pairs |
| DECISION | Possible values ACCEPT | REVIEW | REJECT | ERROR | CANCEL |
| REASON\_CODE | ReasonCode |
| REQUEST\_ID | RequestID |
| REQUEST\_TOKEN | RequestToken |
| AUTHORIZATION\_AMOUNT | AuthorizationAmount |
| AUTHORIZATION\_CODE | AuthorizationCode |
| AUTHORIZATION\_REASON\_CODE | AuthorizationReasonCode |
| DAV\_REASON\_CODE | DAVReasonCode |
| RAW\_SERVICE\_RESPONSE | Entire service response in form of JSON |

1. Test the Validation/Failure response JSON

|  |  |
| --- | --- |
| JSON KEY | JSON VALUE |
| ERROR\_CODE | Validation failure error code of interface |
| ERROR\_MSG | Validation failure message of interface |

## Cartridges Structure and Reference



# Typical Project Plan

## Roles, Responsibilities

Typically most of the integration works is done by the backend developer. We expect that the person doing this integration is familiar with the web service, xml processing and has hands on experience with the Demandware platform.

## Typical Efforts and Timelines

The level of effort is mostly detected by the services merchant may choose from the CyberSource cartridge. The

|  |  |  |
| --- | --- | --- |
| **CyberSource Service** | **Level of Effort (LOE)** | **Dependencies** |
| Initial Cartridge Setup | **0.5**– Person Day  List of tasks involved:   * Add CyberSource Cartridges to the project * Import Configuration files as specified in configuration section | * Cartridge is available |
| Authorize Credit Card | **0.5**– Person Day  List of tasks involved:   * Integrate CyberSource-AuthorizeCreditCard pipeline with COPlaceOrder. | * Merchant ID and Key is established for the client. * Site Preferences for authorization configured with above ID and Key. |
| Device Fingerprint (as addition to Authorize Credit Card) | **0.5** Person Day | * Enable Device Fingerprint, set Organization ID * Add include at billing page. |
| Address Verification Service (AVS)\* | **0.5**– Person Day | * Initial Cartridge Setup |
| Delivery Address Verification (DAV)\* | **0.5**– Person Day | * Initial Cartridge Setup |
| Decision Manager | **0.5**– Person Day | * Access to Decision Manager. * Business rules are defined. * Order status notification URL pointing to Cybersource-NewDecision pipeline is defined. |
| Payment Tokenization\* | **0.5**– Person Day  +  Depends on customization needs | * Initial Cartridge Setup |
| Payer Authentication | **1.5**– Person Day | * Initial Cartridge setup * Update CoPlaceOrder-HandlePayments pipeline * Handle error scenarios in merchant specific ways |
| Alipay Integration on Payment Page | **1.0**– Person Day | * Initial Cartridge setup * Update CoPlaceOrder-HandlePayments pipeline * Handle error scenarios in merchant specific ways |
| Paypal Express Checkout on Cart page and Mini Cart Page | **1.0**– Person Day | * Initial Cartridge setup * Update CoPlaceOrder-HandlePayments pipeline * Handle error scenarios in merchant specific ways |
| Paypal Integration on Payment Page | **1.0**– Person Day | * Initial Cartridge setup * Update CoPlaceOrder-HandlePayments pipeline * Handle error scenarios in merchant specific ways |
| Visa Checkout | **0.5**– Person Day  List of tasks involved:  Integrate VISACHECKOUT pipeline and merchant site specific button injection on minicart, cart and billing page. | * Merchant ID and Key is established for the client. * Visa checkout account setup requried * Site Preferences for authorization configured with above ID and Key. |
| Apple Pay | **2**– Person Day  List of tasks involved:  Choose and decide the integration mechanism with applepay interface. | * Site Preferences for heder authentication exposed. |
| Secure Acceptance (Redirect/Iframe/silent post) | **0.5**– Person Day (1 out of 3 methods)  List of tasks involved:  Integrate SECURE\_ACCEPTANCE pipeline | * Cartridge setup * Configure profile and URL in Cybersource * Site preference configuration in Demandware Business manager * config * . |

\*Note that because customized user interface elements are completely dependent on merchant specification, the time required to interact with the customer to correct address information or confirm standardized address format corrections, is not included; only the time required to integrate with the web services is included, with minimal testing and simplified validation handling, ie. Automatically make correction to a customer address, as per validation response.

## Pre-Production Steps

In order to avoid misuse of unit testing pipelines on production instances it is advised to make following **pipelines node as private and controller function export guard to be removed** before pushing code to production instances.

CybersourceUnitTesting-TestCCAuth

CybersourceUnitTesting- TestAlipayInitiateService

CybersourceUnitTesting- TestAlipayCheckStatusService

CybersourceUnitTesting- TestPaypalCaptureService

CybersourceUnitTesting-TestTax

CybersourceUnitTesting-TestDAVCheck

CybersourceUnitTesting-TestPA

CybersourceUnitTesting-TestFingerprint

Cybersource\_Subscription-Start

Cybersource\_Subscription-CreateSubscription

Cybersource\_Subscription-ViewSubscription

Cybersource\_Subscription-UpdateSubscription

Cybersource\_Subscription-DeleteSubscription

Cybersource\_Subscription-OnDemandPayment

Cybersource\_Services-Start

Cybersource\_Services-Reversal

CybersourceUnitTesting-StartPOS

SECUREACCEPTANCE- TestSATokenCreate

# Known Issues

1. Incase of setting Ignore AVS Result custom preference to true, there can be a known issue as described below:

If the AVS response code received as N, the cartridge ignores the ccAuthReply reason code and processes the transaction under “review” status. This can lead to an ambiguous situation when the Credit Card was rejected, but due to the AVS code as “N”, the cartridge continued with order processing and successful order placement.

1. Testing of Alipay is possible only with Test data provided by CyberSource such as Reconciliation ID that is getting passed to Alipay Initiate Service to get the response back. We don’t have Alipay simulator and access to Alipay live environment.

# CyberSource document links

1. <http://www.cybersource.com/support_center/implementation/testing_info/simple_order_api/General_testing_info/soapi_general_test.html>
2. <http://www.cybersource.com/support_center/support_documentation/quick_references/view.php?page_id=422>
3. <http://apps.cybersource.com/library/documentation/dev_guides/CC_Svcs_SO_API/Credit_Cards_SO_API.pdf> - Page 163 - Appendix C.
4. <http://apps.cybersource.com/library/documentation/dev_guides/Getting_Started/Getting_Started_Advanced.pdf>
5. <http://www.cybersource.com/support_center/support_documentation/quick_references/>
6. <http://apps.cybersource.com/library/documentation/dev_guides/Payer_Authentication_IG/20090928_Payauth_IG.pdf>
7. <http://apps.cybersource.com/library/documentation/dev_guides/Payer_Authentication_IG/html/>
8. <http://apps.cybersource.com/library/documentation/dev_guides/Verification_Svcs_IG/20091012_Verification_IG.pdf>
9. <http://www.cybersource.com/support_center/support_documentation/services_documentation/tax.php>
10. <http://apps.cybersource.com/library/documentation/dev_guides/Tax_IG/Tax_Guide.pdf>
11. <http://apps.cybersource.com/library/documentation/dev_guides/Retail_SO_API/Retail_SO_API.pdf>
12. <http://apps.cybersource.com/library/documentation/dev_guides/PayPal_Express_SO/PayPal_Express_SO_API.pdf>
13. <http://apps.cybersource.com/library/documentation/dev_guides/AliPayDom/AliPay_Dom_SO_API.pdf>
14. <http://apps.cybersource.com/library/documentation/dev_guides/AliPayInt/AliPay_Int_SO_API.pdf>
15. http://apps.cybersource.com/library/documentation/dev\_guides/apple\_payments/SO\_API/Apple\_Pay\_SO\_API.pdf
16. http://apps.cybersource.com/library/documentation/dev\_guides/Secure\_Acceptance\_WM/Secure\_Acceptance\_WM.pdf
17. http://apps.cybersource.com/library/documentation/dev\_guides/Secure\_Acceptance\_SOP/Secure\_Acceptance\_SOP.pdf
18. <http://apps.cybersource.com/library/documentation/dev_guides/VCO_SO_API/Visa_Checkout_SO_API.p>df

# Release History

|  |  |  |
| --- | --- | --- |
| Version | Date | Changes |
| 1.0.0.1 | 02/02/2010 | Initial release |
| 1.0.0.2 | 02/08/2010 | Device Fingerprint Feature added |
| 1.0.0.3 | 03/01/2012 | Updated Tax pipeline to remove unnecessary / redundant tax requests to reduce tax service charges. |
| 1.0.0.4 | 12/18/2012 | Updated Tax pipeline to remove redundant tax requests by using SkipTaxCalculation parameter |
| 1.1.0 | 01/16/2013 | Incorporated review comments from Demandware team |
| 1.1.0 | 02/06/2013 | Incorporated New changes as per new Site Genesis code |
| 2.0.0 | 09/23/2013 | V.me support changes added. Removed deprecated method setGrossPrice for taxation |
| 2.1.0 | 10/04/2013 | V.me Clickjacking changes added |
| 2.1.1 | 11/04/2013 | Removed unsued code from pipeline |
| 2.1.2 | 04/25/2014 | RSA key removed from the cartridge. Bug fixed related to promotional discount. |
| 2.1.3 | 05/29/2014 | Retail Point of Sale (POS) API added |
| 14.2.1 | 08/04/2014 | Document version updated |
| 15.0 | 03/25/2015 | Alipay, Paypal Express and Paypal implementation |
| 15.1.0 | 04/15/2015 | Changes done for Taxation service call and other Changes related to Credit Card and BML. V.me support changes and V.me Clickjacking changes removed. |
| 16.1.0 | 05/30/2016 | Changes done for Controller AsWrapper to call pipeline flows, defects fixes and change request |
| 103.1.3 | 10/23/2016 | Removed:   * BML   Added :   * Visa Checkout * Secure Acceptance Web/Mobile [Redirect/Iframe] * Secure Acceptance Silent Order Post * Apple Pay REST Interface * Added Full controller Migration Except PayPal Express |