CyberSourceCartridge

Version 16.1.0



05/20/2016

Contents

[Summary 5](#_Toc452564015)

[Component Overview 7](#_Toc452564016)

[Functional Overview 7](#_Toc452564017)

[Credit Card Authorization Service 7](#_Toc452564018)

[Taxes 8](#_Toc452564019)

[Address Verification Service (AVS) 9](#_Toc452564020)

[Delivery Address Verification Service (DAV) 9](#_Toc452564021)

[Bill Me Later (BML) 9](#_Toc452564022)

[Decision Manager 10](#_Toc452564023)

[Conversion Detail Report 10](#_Toc452564024)

[Payment Tokenization 11](#_Toc452564025)

[Payer Authentication 11](#_Toc452564026)

[Full Authorization Reversal 12](#_Toc452564027)

[Retail Point-of-Sale (POS) 12](#_Toc452564028)

[Alipay Authorization 13](#_Toc452564029)

[Alipay Batch Job 14](#_Toc452564030)

[PayPal Express Authorization [From Cart Page and Mini Cart] 14](#_Toc452564031)

[PayPal Authorization [From Billing Page] 15](#_Toc452564032)

[Use Cases Scenarios 16](#_Toc452564033)

[Credit Card Authorization 16](#_Toc452564034)

[Taxes 18](#_Toc452564035)

[Address Validation Service (AVS) 18](#_Toc452564036)

[Delivery Address Verification Service (DAV) 19](#_Toc452564037)

[BML 19](#_Toc452564038)

[Conversion Detail Report 19](#_Toc452564039)

[Payment Tokenization 20](#_Toc452564040)

[Payer Authorization 20](#_Toc452564041)

[Retail Point-of-Sale (POS) 20](#_Toc452564042)

[Alipay Authorization 21](#_Toc452564043)

[PayPal Express Checkout and Authorization 22](#_Toc452564044)

[Credit Card Order Status Mapping with Demandware Order 26](#_Toc452564045)

[Alipay PayPal Order Status Mapping with Demandware Order 27](#_Toc452564046)

[Demandware – Cybersource Service Response Field Mapping for Alipay and Paypal Services 27](#_Toc452564047)

[Limitations, Constraints 31](#_Toc452564048)

[Compatibility 33](#_Toc452564049)

[Implementation Guide 33](#_Toc452564050)

[Custom Code 33](#_Toc452564051)

[Generic Section 33](#_Toc452564052)

[Credit Card Auth 56](#_Toc452564053)

[Bill Me Later 68](#_Toc452564054)

[Tax Service 72](#_Toc452564055)

[Address Verification Service 79](#_Toc452564056)

[Delivery Address Validation Service 79](#_Toc452564057)

[Full Authorization Reversal 80](#_Toc452564058)

[Payer Authentication Service 80](#_Toc452564059)

[Payment Tokenization Service 81](#_Toc452564060)

[Alipay Authorization 94](#_Toc452564061)

[Alipay Batch Job 105](#_Toc452564062)

[PayPal Express Checkout [From Cart Page and Mini Cart] 106](#_Toc452564063)

[PayPal Checkout [From Billing Page] 119](#_Toc452564064)

[Remove Duplicate Payment Methods while Checkout 134](#_Toc452564065)

[Retail POS 136](#_Toc452564066)

[Site Configuration 150](#_Toc452564067)

[Configure Payment Processor for Alipay 150](#_Toc452564068)

[Import Meta Data 150](#_Toc452564069)

[Import Payment Methods and Cards 151](#_Toc452564070)

[Configure Services 151](#_Toc452564071)

[Configure Site Preferences 155](#_Toc452564072)

[Configure Site Preferences for Alipay 159](#_Toc452564073)

[Configure Batch Job for AliPay 159](#_Toc452564074)

[Configure Batch Job for Conversion Detail Report 160](#_Toc452564075)

[Configure Site Preferences for PayPal and PayPal Express Checkout 161](#_Toc452564076)

[Applying CyberSource Cartridge to the Site 162](#_Toc452564077)

[Configure Custom Objects for Retail POS 163](#_Toc452564078)

[Testing 165](#_Toc452564079)

[Authorize Credit Card 165](#_Toc452564080)

[Tax Service 165](#_Toc452564081)

[Address Verification Service (AVS) 166](#_Toc452564082)

[Delivery Address Verification Service (DAV) 166](#_Toc452564083)

[Payment Tokenization 167](#_Toc452564084)

[Full Authorization reversal 167](#_Toc452564085)

[Device Fingerprint 167](#_Toc452564086)

[Payer Authentication 168](#_Toc452564087)

[Retail POS Authorization Request 168](#_Toc452564088)

[Alipay Initiate Request 168](#_Toc452564089)

[Alipay Check Status Request 169](#_Toc452564090)

[Credit Card Capture Request 169](#_Toc452564091)

[Paypal Capture Request 170](#_Toc452564092)

[Cartridges Structure and Reference 170](#_Toc452564093)

[Pipelines & Controllers 170](#_Toc452564094)

[Scripts 171](#_Toc452564095)

[Templates 173](#_Toc452564096)

[Configuration Files 173](#_Toc452564097)

[Retail POS Cartridge Components 174](#_Toc452564098)

[Typical Project Plan 174](#_Toc452564099)

[Roles, Responsibilities 174](#_Toc452564100)

[Typical Efforts and Timelines 174](#_Toc452564101)

[Pre-Production Steps 176](#_Toc452564102)

[CyberSource Site Preferences 178](#_Toc452564103)

[Site preference and description 178](#_Toc452564104)

[Cybersource\_paypal Site Preferences 180](#_Toc452564105)

[Site preference and description 180](#_Toc452564106)

[Device Fingerprint 182](#_Toc452564107)

[How does it work? 182](#_Toc452564108)

[Setup: 182](#_Toc452564109)

[Hints for the CsDeviceFingerprintRedirectionType: 183](#_Toc452564110)

[Modified Scripts and pipelines for the device fingerprint 185](#_Toc452564111)

[Known Issues 186](#_Toc452564112)

[CyberSource document links 187](#_Toc452564113)

[Release History 188](#_Toc452564114)

# 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 – The CyberSource Credit Card Authorization service provides merchant with a mechanism to get authorization for the order amount. The authorization service validates the card based and authorize card for the order amount. For additional spam/fraud detection by Cybersource, a ‘device fingerprint’ could be submitted additionally, if configured.

CyberSource Address Verification (AVS) – The CyberSource AVS service provides merchants with a mechanism to reduce merchant banking fees, by verifying billing information before authorizing payment for customer purchases. Although the AVS service is automatically called during the authorization process, the behavior resulting from specific AVS responses and its interaction with the payment process is customizable through storefront configuration.

Tax Service – The CyberSource tax calculation service provides merchants with a complete tax calculation service according to and pursuant to domestic and international tax regulations, including but not limited to, district, city, county and state levels of governing tax authority.

Delivery Address Verification (DAV) Service – The CyberSource DAV service provides merchants with an optional mechanism to prevent, limit or correct faulty shipping information, related to improperly entered or formatted information from the customer. This service helps reduce the potential additional costs resulting from undeliverable or returned merchandise.

Bill Me Later Service – The Bill Me Later service from Bill Me Later, Inc., and available via your single connection to CyberSource, allows your customers to make purchases using an instant line of credit.

Decision Manager – CyberSource Decision Manager Service provides set of tools that merchant to evaluate rules and chose tool and return a decision of “Accept”, “Reject”, or “Review”. Merchant can also setup process to ignore certain rules when necessary.

Payment Tokenization – CyberSource Payment Tokenization Service provides set of tools to store customer and payment related sensitive data on secured cybersource hosted servers.

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.

Alipay Authorization – The CyberSource Alipay authorization service provide merchant with a mechanism to get authorization for order amount. The Initiate service of CyberSource Alipay initiates and authorizes the ordered amount and check status service returns the payment status of raised request from Alipay and return user to merchant site. There is a slight difference of currency associated with the amount in the request for Alipay Domestic and International simulators. CNY would be the currency associated for Domestic Alipay requests and Site specific currency would be associated with International Alipay requests.

Note: Please refer to the section Alipay PayPal Order Status Mapping with Demandware Order on page 37.

Alipay Batch Job – The CyberSource Alipay Batch Job provide merchant an additional functionality to change the Demandware order and payment status after getting response from check status service call. This batch job processes all the order placed with Alipay as payment methods and orders with status New, Open and Created. For each order it passes Request Id to Alipay Check Status service and accordingly updates the Demandware Order and Payment Status.

Note: Please refer to the section Alipay PayPal Order Status Mapping with Demandware Order on page 37.

PayPal Express Authorization [From Cart Page and Mini Cart] – The CyberSource PayPal Express authorization service provide merchant with a mechanism to authorize and capture order amount by using billing and shipping address of user’s PayPal account. User directly routed to PayPal site on choosing express checkout and return back to merchant site after validating its credentials. On placing the order, authorization and capture service of CyberSource PayPal authorize the amount and change the status of Demandware order to Paid. On configuring system to authorization service only, amount would be authorized but order payment status would be Not Paid.

Note: Please refer to the section Alipay PayPal Order Status Mapping with Demandware Order on page 37.

PayPal Authorization [From Billing Page] – The CyberSource PayPal authorization service provide merchant with a mechanism to authorize and capture order amount using normal checkout flow with PayPal as payment option. On placing an order user will be redirected to PayPal account to authenticate its credentials and after authentication user will be redirected back to merchant site to complete the checkout flow. Order payment status would be same mentioned above for PayPal express authorization.

Note: Please refer to the section Alipay PayPal Order Status Mapping with Demandware Order on page 37.

# 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

### Bill Me Later (BML)

A customer selects Bill Me Later during the checkout process at your site, similar to selecting the option to pay via Visa or MasterCard.

To request a Bill Me Later authorization, Demandware pipeline/controller sends a request for a credit card authorization but instead of including a credit card number in the request, send the customer’s Bill Me Later account number set in the custom preferences.

To bill the customer, send a request for a credit card capture. No additional capture request fields are required for a Bill Me Later capture, unless you are processing multiple captures.

The Bill Me Later authorization service pipeline/controller allows storefront application to request for credit authorization for the total order amount through the Bill Me Later.

The Demandware Cybersource–AuthorizeBML 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.

Bill Me Later authorization sequence process:

* A customer using Bill Me Later, are presented with terms and conditions and asked for date of birth and last 4 digits of social security number.
* Once terms are agreed to, checkout continues as usual.
* A real-time credit decision is made and the consumer is notified within 3-5 seconds.
* On subsequent purchases, the customer simply chooses Bill Me later, a real-time credit decision is made and the customer is notified within 3-5 seconds.
* In rare circumstances, additional verification steps are applied.
* After completion of the first purchase a welcome email is sent by Bill Me Later. This email contains a secure link for login to the self-service website, as well as a user ID for the customer.
* Your customer typically receives a bill (sent by Bill Me Later, Inc.) in their mailbox within fifteen days of the settlement posting to the customer’s account. The due date for this bill is 25 days from the date the bill is rendered.
* You submit the transaction for settlement upon shipment of goods or rendering the service—just as you do with a credit card purchase. Funding is received within the same timeframe as that of a purchase made with a credit card.

### 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.

## Use Cases Scenarios

### Credit Card 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. |

### BML

List of use cases and appropriate action taken listed below:

|  |  |
| --- | --- |
| **Use case scenarios** | **Result** |
| BML Authorization failed with response.decision = ERROR | Client application to display appropriate user friendly message to the end user. |
| BML Authorization failed with response.decision = ACCEPT | Pipeline sets the Authorization code to BMLPaymentInstrument.paymentTransaction.transactionID and ends with Authorized status  Order object is populated with cybersource transaction ID |
| BML Authorization failed with script error, or exception | Pipeline ends with error status, client code to display appropriate error message to the end user. |

### 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.

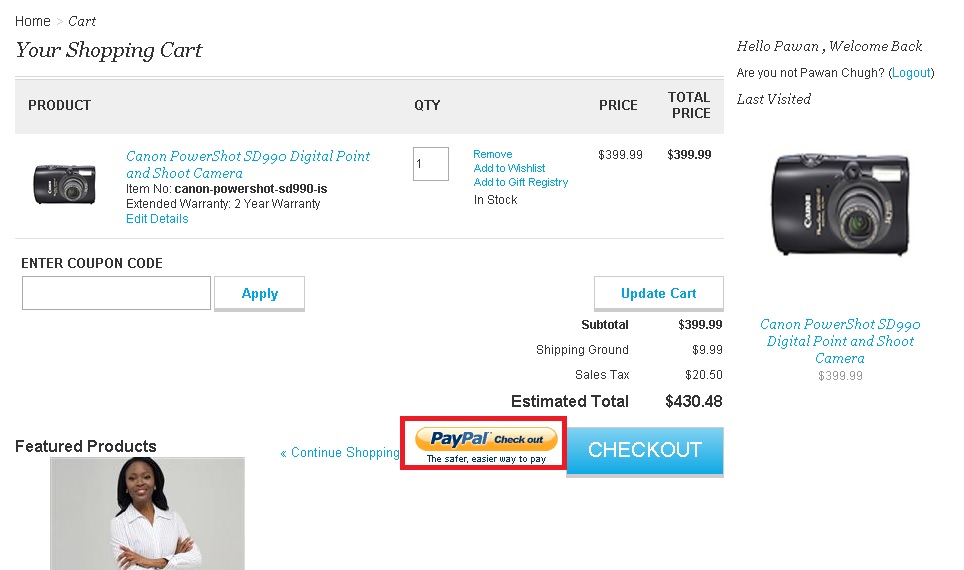
**Step 3:** Order Setup request and reply— accept payer id and order details to generate order setup response required to authorize the request.

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

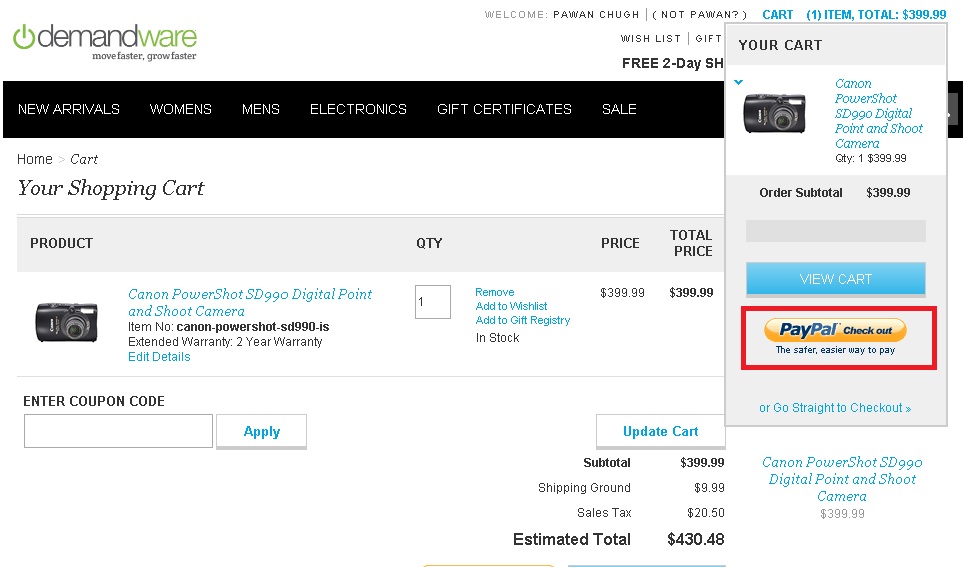
**Step 5:** Capture service request and reply — capture the amount authorized by Authorization service.

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

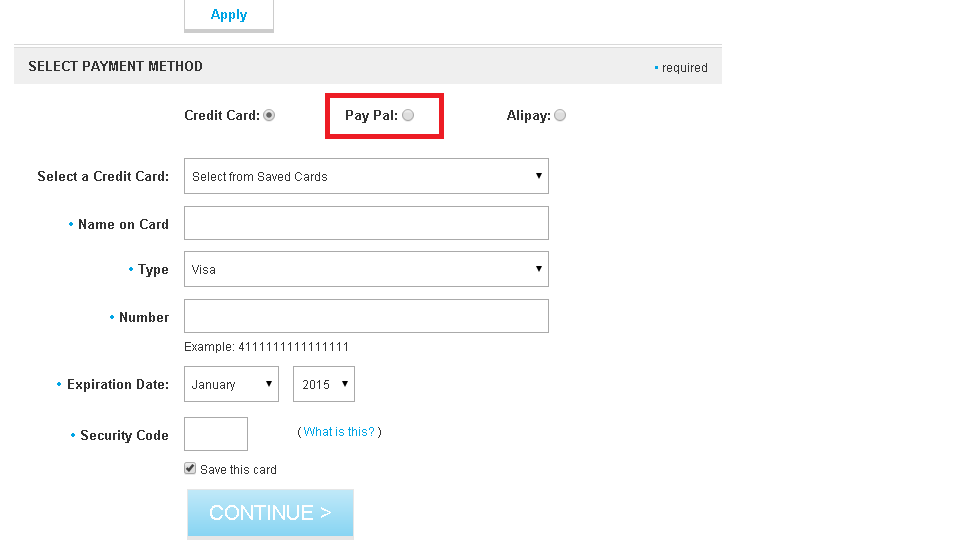
“PayPal Checkout” button has been added on Demandware reference SiteGenesis.



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 |

## 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.

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.

## Compatibility

This cartridge is supported under Demandware API release 2.10.0 and onward.

# Implementation Guide

## Custom Code

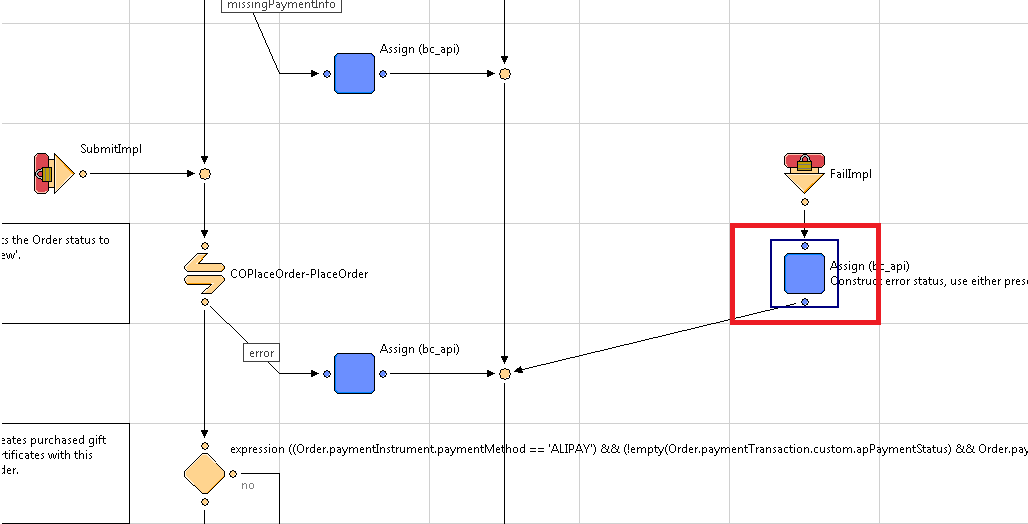
**Note: While integration either implement pipeline changes or controller file changes depending on the storefront site developed on pipeline or controller.**

### Generic Section

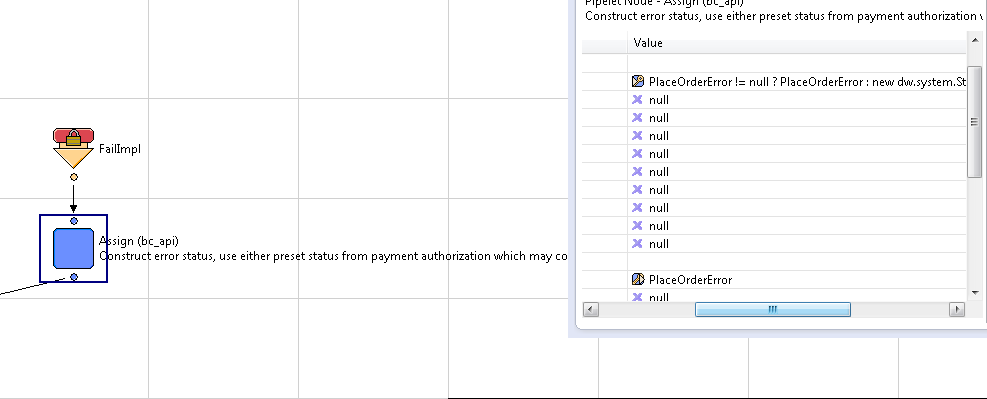
#### Pipeline - COPlaceOrder.xml

##### Update “FailImpl” node

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

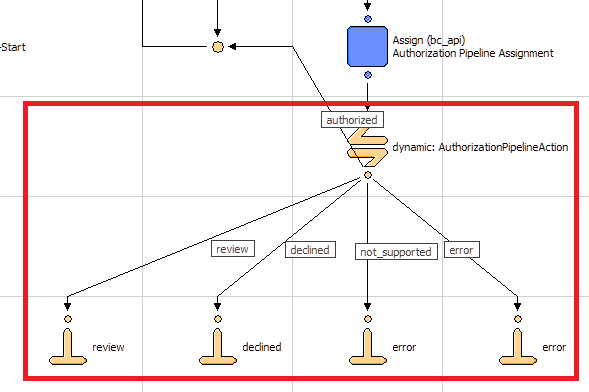


* Assign PlaceOrderError != null ? PlaceOrderError : new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined") value in the pipeline Input Dictionary as shown below.



##### Update “HandlePayments” node

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

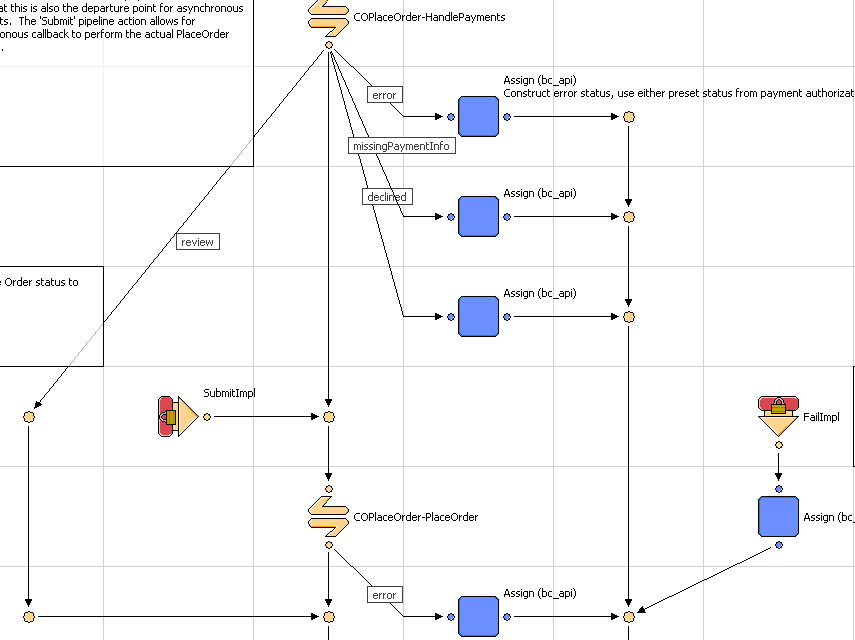


##### Update “start” node

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

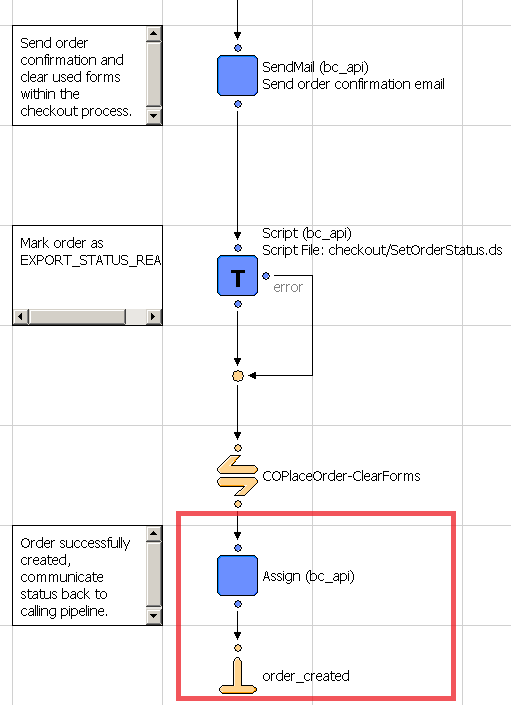
* Add assign node and set the value of **PlaceOrderError** as **PlaceOrderError != null ? PlaceOrderError : new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined") for error connector pipelet**
* Add assign node and set the value of **PlaceOrderError** as **new dw.system.Status(dw.system.Status.ERROR, "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**

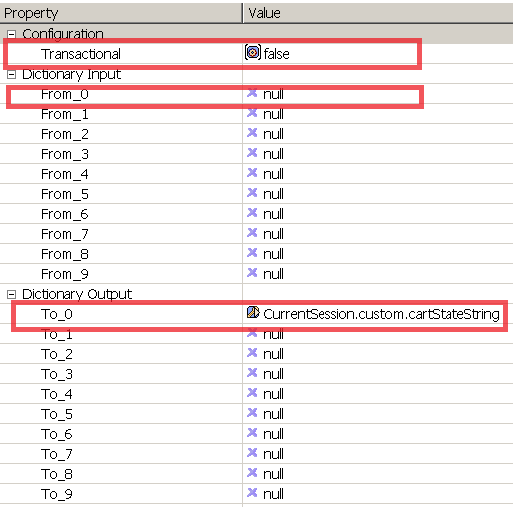
Refer below



##### 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:

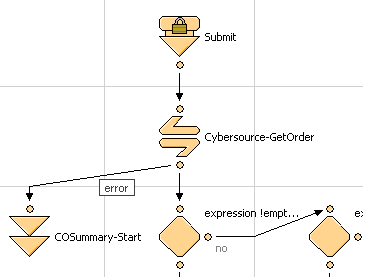




##### Update”Submit” node

###### Find correct order object to process

* Add call node for Cybersource-GetOrder, on error returned add jump node COSummary-Start



###### Order Placement for Alipay

1. Add decision node just after “GetOrder” and check if existing payment instrument in Alipay as **!empty(Order.getPaymentInstruments('ALIPAY'))**
2. If TRUE , call Cybersource-CheckAlipayPaymentStatus

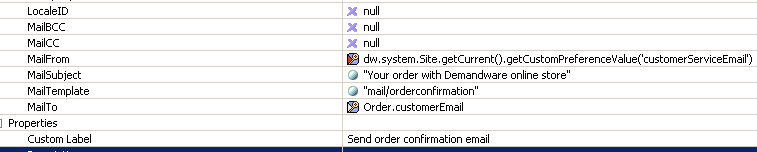
On error transition- jump node COPlaceOrder-Fail

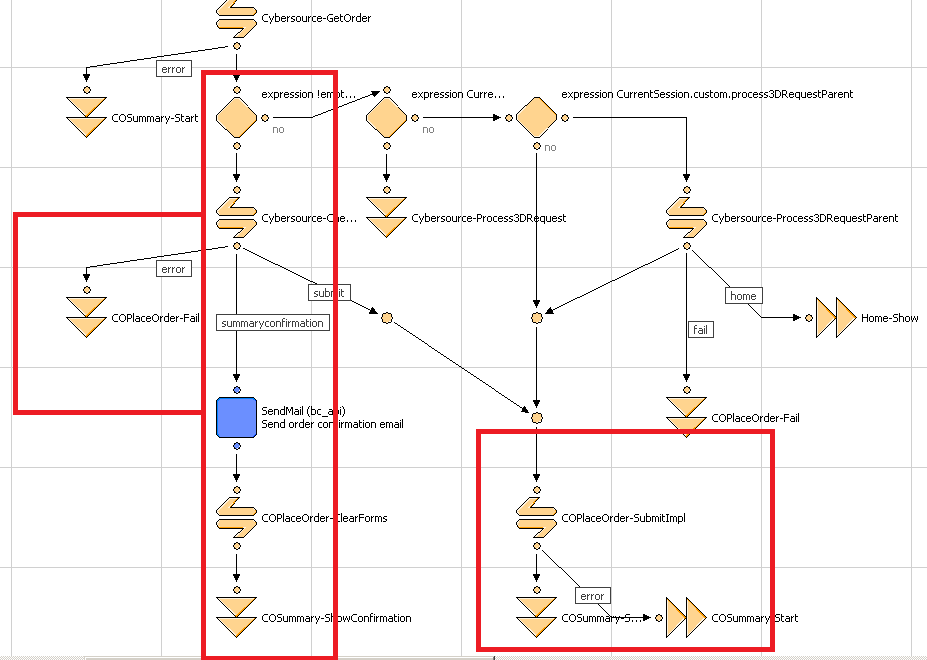
On submit transition–

* call COPlaceOrder-SubmitImpl
* On error jump node to COSummary-Start
* Else jump node COSummary-ShowConfirmation

On summaryConfirmation transition-:

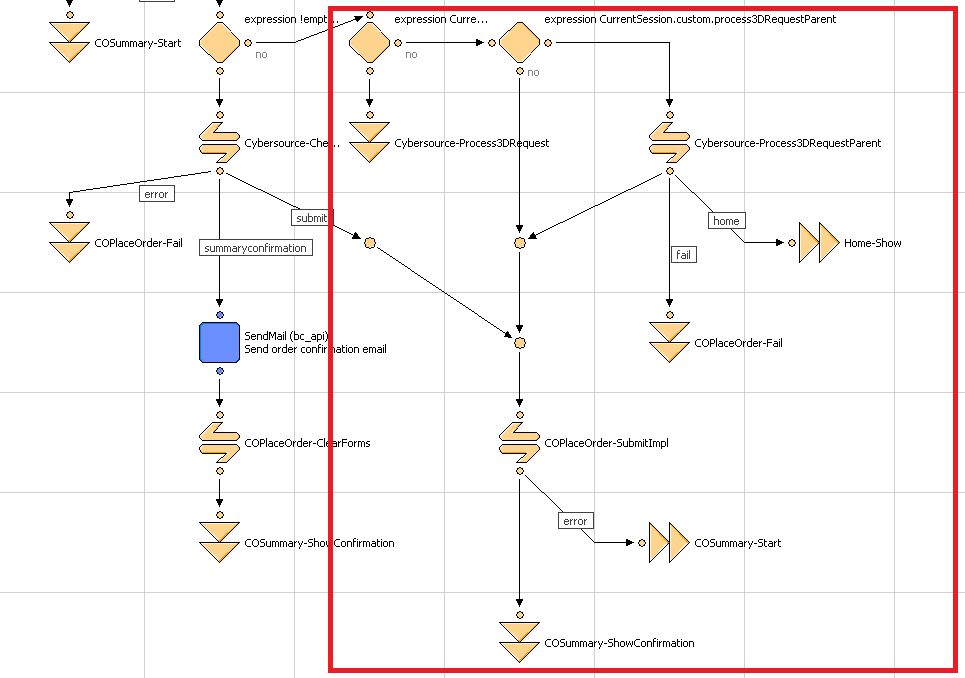
* Add sendMail pipelet and assign below values as given in screen shot:
  + mailFrom: dw.system.Site.getCurrent().getCustomPreferenceValue('customerServiceEmail')
  + MailSubject : "Your order with Demandware online store"
  + MailTemplate: "mail/orderconfirmation"
  + MailTo: Order.customerEmail
  + Custom label : Send order confirmation email
* Add Call node - COPlaceOrder-ClearForms
* Jump Node - COSummary-ShowConfirmation



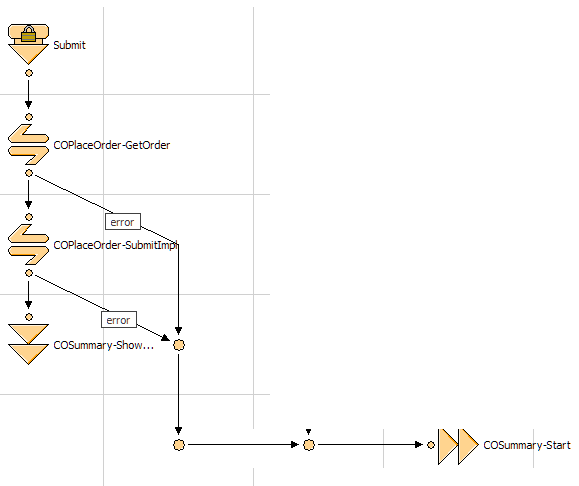


###### Order Placement for 3D Card [Payer Authentication]

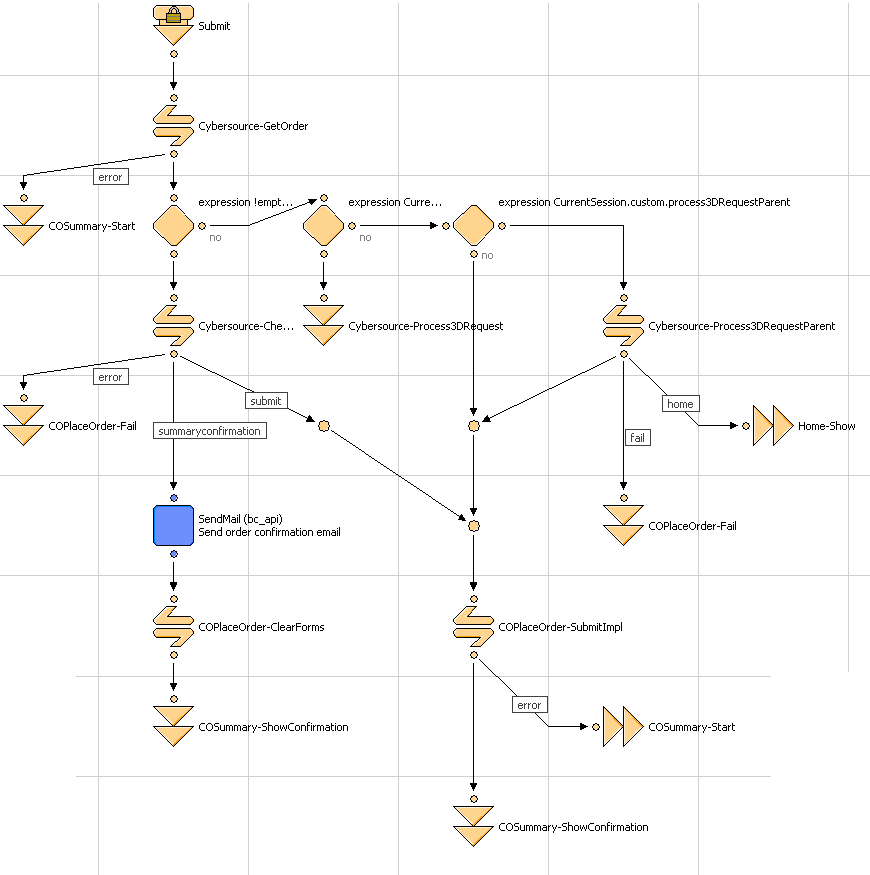
1. Check if session is established for 3D request as **CurrentSession.custom.process3DRequest** if yes then jump node Cybersource-Process3DRequest
2. If not, check session is established for parent 3D request as **CurrentSession.custom.process3DRequestParent** if yes then
   * Call Cybersource-Process3DRequestParent and handle reasponses as follows:
     + on home transition – Jump node Home-Show
     + On fail transition – JumpNode COPlaceOrder-Fail
     + else below steps
       - call COPlaceOrder-SubmitImpl
       - On error jump node to COSummary-Start
       - Else jump node COSummary-ShowConfirmation



Before



After

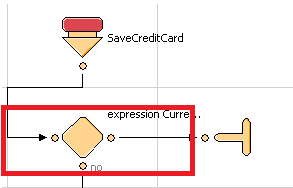


#### Pipeline - COBilling.xml

##### Update “SaveCreditCard” Node to Bypass savingcard for payment methods

1. Add a new condition as first decision node to bypass save credit card logic when payment method selected as ALIPAY or PayPal orBML on billing page

CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals(dw.order.PaymentInstrument.METHOD\_BML) || CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals("PayPal") || CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals("ALIPAY") || !empty(Basket.getPaymentInstruments("PayPal"))



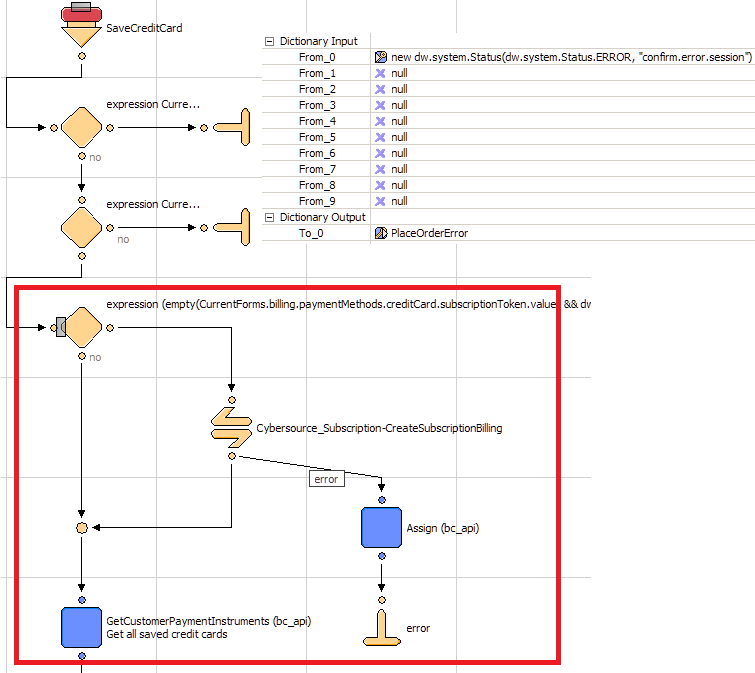
##### Update “SaveCreditCard” node to handle tokenization

1. Add Conditional Node to check if the Token is available in current forms for selected payment card. If not make a call to Cybersource pipeline to Create Subscription.

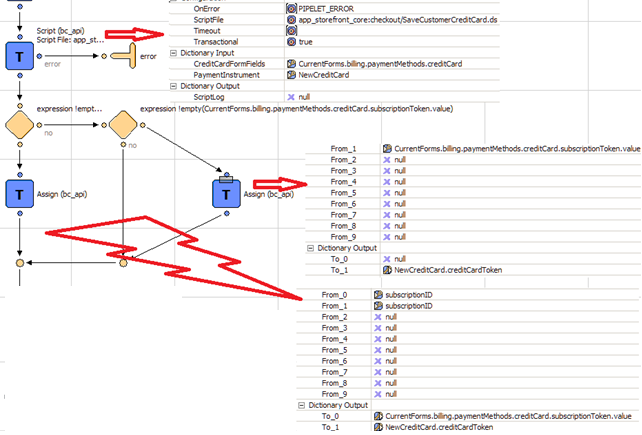
(empty(CurrentForms.billing.paymentMethods.creditCard.subscriptionToken.value) && dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable") == "YES")

1. Call “Cybersource\_Subscription-CreateSubscriptionBilling” when token not available on billing page
2. On error transition add ASSIGN node where set variable “PlaceorderError” with value as

new dw.system.Status(dw.system.Status.ERROR, "confirm.error.session")



1. After GetCustomerPaymentInstruments pipelet add an Script pipelet “SaveCustomerCreditCard.ds” and set the input as below
   * “CreditCardFormFields” with CurrentForms.billing.paymentMethods.creditCard
   * “PaymentInstrument” with NewCreditCard
2. Add decision node and check for SubscriptionId existence as 🡺 *!empty(subscriptionID)*
   1. If Subscription Id does not exist, add another decision node and check for existing credit card subscription token
      1. If exists -> Add assign node and set the value of NewCreditCard.creditCardToken as CurrentForms.billing.paymentMethods.creditCard.subscriptionToken.value
   2. If SubscriptionId exist, Add an assign node and set the value of subscriptionID in both(CurrentForms.billing.paymentMethods.creditCard.subscriptionToken.value and NewCreditCard.creditCardToken)



##### Update “SaveCreditCard” node to handle duplicate card

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;  } |

1. Add another Script pipelet and call Script File “DuplicateCheck.ds” with below input and output parameters:

Input:

cardType : CurrentForms.billing.paymentMethods.creditCard.type.value

expMonth: CurrentForms.billing.paymentMethods.creditCard.expiration.month.value

expYear : CurrentForms.billing.paymentMethods.creditCard.expiration.year.value

NoIn : CurrentForms.billing.paymentMethods.creditCard.number.value

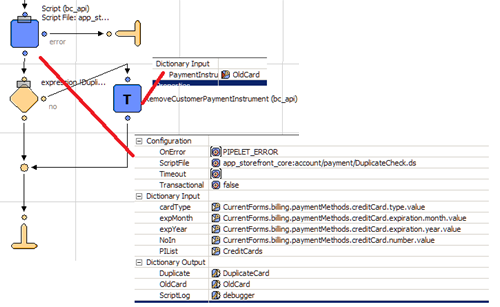
PIList : CreditCards

Output:

Duplicate : DuplicateCard

OldCard : OldCard

1. Add a decision node as !DuplicateCard to check for any duplicate card return from above step
   1. Add RemoveCustomerPaymentInstrument pipelet and remove the old card in case any duplicate card found.
2. Add and end node to complete the transition



#### Controller - COPlaceOrder.js

##### 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**  };  }  }  }  }  **return** authorizationResult;  } |

##### Update “start” function

|  |
| --- |
| **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.missingPaymentInfo) {  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) {  **return** submitImpl(order, **true**);  } **else** **if** (handlePaymentsResult.pending) {  **return** submitImpl(order, **true**);  }  **return** submitImpl(order);  } |

##### Add” fail” function

1. Add below export function line at end of file

exports.Fail = guard.ensure(['https'], fail);

1. Add new method to handle the failed order

|  |
| --- |
| /\*  \* Asynchronous Callbacks for SiteGenesis.  Identifies if an order fails, fails the order,and return to summary page.\*/  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;  } |

##### Update “submitImpl” function

Add in session cartStateString as null before end of function where order created returned as true

session.custom.cartStateString=**null**;

**return** {

Order: order,

order\_created: **true**

};

}

##### Update “submit” function

Replace the submit function with the code snippet below, it has Alipay changes also

|  |
| --- |
| function submit(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;  if (!empty(order.getPaymentInstruments('ALIPAY'))) {  var alipayResult = Cybersource.CheckAlipayPaymentStatus({Order:order});  if (alipayResult.summaryconfirmation) {  // 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.  session.forms.singleshipping.clearFormElement();  session.forms.multishipping.clearFormElement();  session.forms.billing.clearFormElement();  session.custom.cartStateString=null;  app.getController('COSummary').ShowConfirmation(order);  return;  } else if (alipayResult.error)  {  app.getController('COPlaceOrder').Fail({Order:order});  return;  }  }  if (session.custom.process3DRequest) {  Cybersource.Process3DRequest({Order:order});  return;  } else if (session.custom.process3DRequestParent) {  var process3DResult = Cybersource.Process3DRequestParent({Order:order});  if (process3DResult.fail)  {  app.getController('COPlaceOrder').Fail({Order:order,PlaceOrderError:process3DResult.PlaceOrderError});  return;  } else if (process3DResult.home) {  app.getController('Home').Show();  return;    }  }  if (submitImpl(order).error) {  app.getController('COSummary').Start();  return;  }  app.getController('COSummary').ShowConfirmation(order);  } |

#### Controller - COBilling.js

Add below code at the top of API Include section

|  |
| --- |
| var ArrayList = require('dw/util/ArrayList');  var Pipelet = require('dw/system/Pipelet'); |

##### Update “saveCreditCard” function

Replace the entire function with the below snippet, it has all customizations: The tokenization changes are highlighted in below code snippet.

|  |
| --- |
| **Function** saveCreditCard() {  **var** i, creditCards, GetCustomerPaymentInstrumentsResult, subscriptionID;  **var** cart = app.getModel('Cart').get();  **if**(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_BML) ||  app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals("PayPal") ||  app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals("ALIPAY") ||  !empty(cart.getPaymentInstruments("PayPal"))) {  **return** **true**;  }  **if** (customer.authenticated && app.getForm('billing').object.paymentMethods.creditCard.saveCard.value) {  **var** enableTokenization : String = dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable");  **if** (empty(app.getForm('billing').object.paymentMethods.creditCard.subscriptionToken.value) && enableTokenization=='YES' ){  **var** Cybersource\_Subscription = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource\_Subscription')  **var** createSubscriptionBillingResult = Cybersource\_Subscription.CreateSubscriptionBilling({Basket:cart});  **if** (createSubscriptionBillingResult.error) {  **return** {  error: **true**,  PlaceOrderError: **new** dw.system.Status(dw.system.Status.ERROR, 'confirm.error.session')  };  }  subscriptionID = createSubscriptionBillingResult.subscriptionID;  }  // var creditCards = customer.getProfile().getWallet().getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD);  creditCards = **new** ArrayList();  GetCustomerPaymentInstrumentsResult = **new** Pipelet('GetCustomerPaymentInstruments').execute({  Customer: customer,  PaymentMethod: PaymentInstrument.METHOD\_CREDIT\_CARD  });  **if** (GetCustomerPaymentInstrumentsResult.result !== PIPELET\_ERROR) {  creditCards = GetCustomerPaymentInstrumentsResult.PaymentInstruments;  }  **if** (!empty(subscriptionID)) {  app.getForm('billing').object.paymentMethods.creditCard.subscriptionToken.value = createSubscriptionBillingResult.subscriptionID;  }  Transaction.wrap(**function** () {  **var** newCreditCard = **null**;  **var** wallet = customer.getProfile().getWallet();  newCreditCard = customer.getProfile().getWallet().createPaymentInstrument(PaymentInstrument.METHOD\_CREDIT\_CARD);  // coy the credit card details to the payment instrument  newCreditCard.setCreditCardHolder(app.getForm('billing').object.paymentMethods.creditCard.owner.value);  newCreditCard.setCreditCardNumber(app.getForm('billing').object.paymentMethods.creditCard.number.value);  newCreditCard.setCreditCardExpirationMonth(app.getForm('billing').object.paymentMethods.creditCard.expiration.month.value);  newCreditCard.setCreditCardExpirationYear(app.getForm('billing').object.paymentMethods.creditCard.expiration.year.value);  newCreditCard.setCreditCardType(app.getForm('billing').object.paymentMethods.creditCard.type.value);  **if**(!empty(app.getForm('billing').object.paymentMethods.creditCard.subscriptionToken.value)) {  newCreditCard.setCreditCardToken(app.getForm('billing').object.paymentMethods.creditCard.subscriptionToken.value);  }    **var** isDuplicateCard = **false**;  **var** oldCard;    **var** ccNumber = newCreditCard.getCreditCardNumber();  **for** (i = 0; i < creditCards.length; i++) {  **var** creditcard = creditCards[i];  **var** creditcardNo = creditcard.getCreditCardNumber();  **if** (creditcard.creditCardExpirationMonth === newCreditCard.creditCardExpirationMonth && creditcard.creditCardExpirationYear === newCreditCard.creditCardExpirationYear  && creditcard.creditCardType === newCreditCard.creditCardType && creditcardNo.toString().substring(creditcardNo.length-4).equals(ccNumber.substring(ccNumber.length-4))) {  isDuplicateCard = **true**;  oldCard = creditcard;  **break**;  }  }  **if** (isDuplicateCard) {  wallet.removePaymentInstrument(oldCard);  }  });  }  **return** **true**;  } |

#### Controller – COSummary.js

##### Update “start” function

Update the function to handle the error while placing order

|  |
| --- |
| **function** start(args) {  **var** cart = Cart.get();  // Checks whether all payment methods are still applicable. Recalculates all existing non-gift certificate payment  // instrument totals according to redeemed gift certificates or additional discounts granted through coupon  // redemptions on this page.  **var** COBilling = app.getController('COBilling');  **if** (!COBilling.ValidatePayment(cart)) {  COBilling.Start();  **return**;  } **else** {  Transaction.wrap(**function** () {  cart.calculate();  });  **var** pageMeta = require('~/cartridge/scripts/meta');  pageMeta.update({pageTitle: Resource.msg('summary.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')});  **if** (**undefined** != args && args.PlaceOrderError) {  app.getView({  Basket: cart.object,  PlaceOrderError : args.PlaceOrderError  }).render('checkout/summary/summary');  } **else** {  app.getView({  Basket: cart.object  }).render('checkout/summary/summary');  }  }  } |

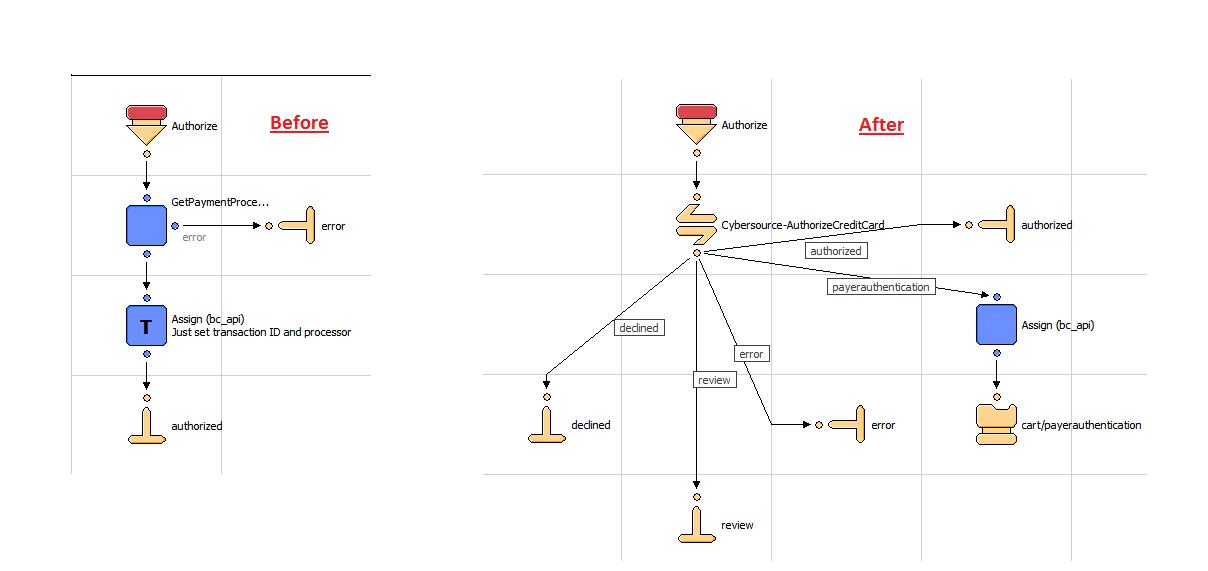
### Credit Card Auth

#### Pipeline - CYBERSOURCE\_CREDIT.xml

##### Update “Authorize” node

Update “Authorize” node to call Cybersource-AuthorizeCreditCard

Payer authentication transition is implemented for handling fraud check so refer Payer Authentication Service section for more detail.



#### Pipeline - BASIC\_CREDIT.xml

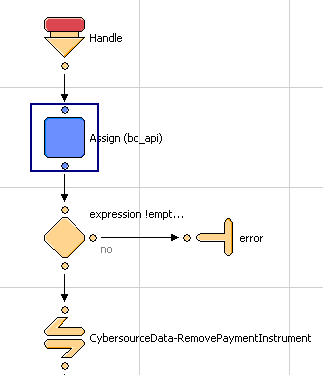
##### Update “Handle” node

Update the handle to remove exiting payment instruments from Basket using below steps:

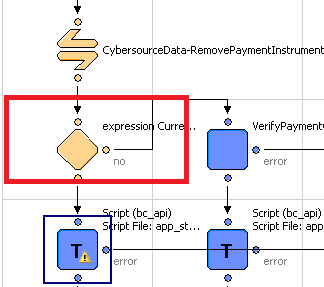
1. Add and assign node and set in input as “CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value” and output as “PaymentMethod” .



1. Add a decision node and check if existing payment method is CREDIT\_CARD as **!empty(PaymentMethod) && PaymentMethod.equals('CREDIT\_CARD')**
2. If TRUE, call CybersourceData-RemovePaymentInstrument else add error node



1. Add decision node CurrentForms.billing.paymentMethods.creditCard.subscriptionToken.value



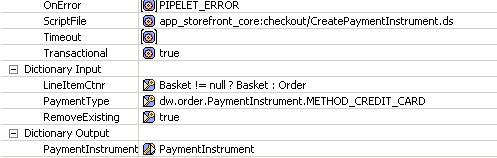
1. If TRUE, Add script pipelet as transactional and call ScriptFile checkout/CreatePaymentInstrument.ds , set the input and output as:[map to error in case of error]

Input:

|  |  |
| --- | --- |
| LineItemCtnr | Basket != null ? Basket : Order |
| PaymentType | dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD |
| RemoveExisting | TRUE |

Output:

|  |  |
| --- | --- |
| PaymentInstrument | PaymentInstrument |



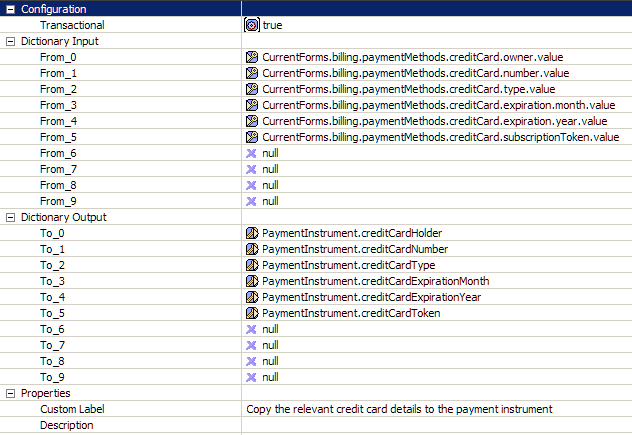
1. Add assign node and map input output as:

Input:

|  |  |
| --- | --- |
| From\_0 | CurrentForms.billing.paymentMethods.creditCard.owner.value |
| From\_1 | CurrentForms.billing.paymentMethods.creditCard.number.value |
| From\_2 | CurrentForms.billing.paymentMethods.creditCard.type.value |
| From\_3 | CurrentForms.billing.paymentMethods.creditCard.expiration.month.value |
| From\_4 | CurrentForms.billing.paymentMethods.creditCard.expiration.year.value |
|  |  |
| From\_5 | CurrentForms.billing.paymentMethods.creditCard.subscriptionToken.value |

Output:

|  |  |
| --- | --- |
| To\_0 | PaymentInstrument.creditCardHolder |
| To\_1 | PaymentInstrument.creditCardNumber |
| To\_2 | PaymentInstrument.creditCardType |
| To\_3 | PaymentInstrument.creditCardExpirationMonth |
| To\_4 | PaymentInstrument.creditCardExpirationYear |
|  |  |
| To\_5 | PaymentInstrument.creditCardToken |

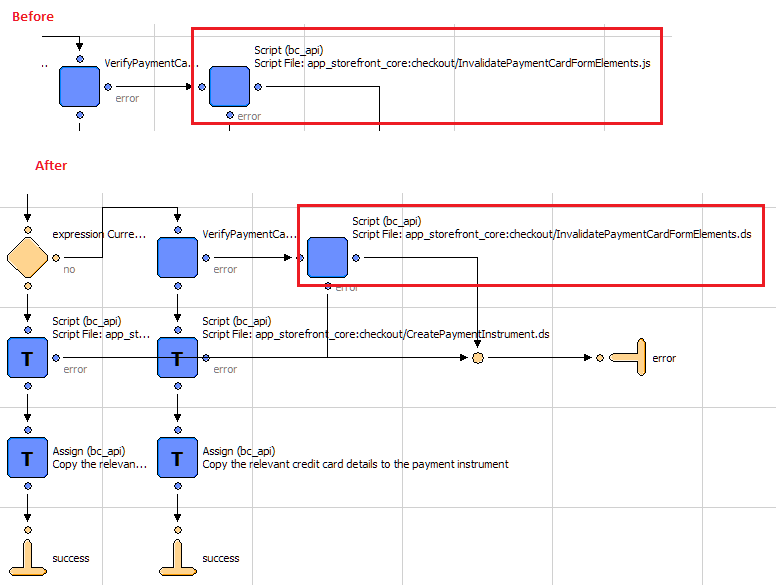




1. If [Step 4] false, continue with the existing flow.

[Note]: In existing flow incorrect mapping is done for InvalidatePaymentCardFormElements. The script pipelet is mapped with .js extension instead of .ds

Refer screenshot below



1. Add a new file InvalidatePaymentCardFormElements .ds as and paste the code of InvalidatePaymentCardFormElemets.js into this under checkout section in storefront core cartridge

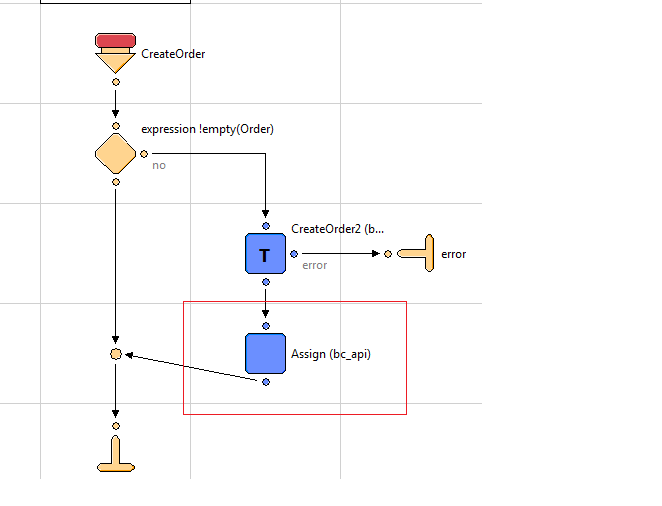
|  |
| --- |
| /\*\*  \* Invalidates the payment card form element in case specified status is ERROR.  \* If status is undefined or form is invalid the pipelet returns PIPELET\_ERROR.  \*  \* @input Status : dw.system.Status The status object.  \* @input CreditCardForm : dw.web.FormGroup The credit card form.  \*/  function execute(pdict) {  var isInvalidated = invalidatePaymentCardForm(pdict.Status, pdict.CreditCardForm);  if(!isInvalidated) {  return PIPELET\_ERROR;  }  return PIPELET\_NEXT;  }  /\*\*  \* Invalidates Payment Card Form  \*  \* @param {dw.system.Status} status - verification status of the credit card  \* @param {dw.web.Form} creditCardForm - The credit card form  \* @return {Boolean} - success status of invalidating payment card form  \*/  function invalidatePaymentCardForm(status, creditCardForm) {  var Status = require('dw/system/Status');  var PaymentStatusCodes = require('dw/order/PaymentStatusCodes');  // verify that we have a status object and a valid credit card form  if (!status || !creditCardForm.valid) {  return false;  }  // we are fine, if status is OK  if (status.status === Status.OK) {  return true;  }  // invalidate the payment card form elements  var items = status.items.iterator();  while (items.hasNext()) {  var item = items.next();  switch(item.code) {  case PaymentStatusCodes.CREDITCARD\_INVALID\_CARD\_NUMBER:  creditCardForm.number.setValue(creditCardForm.number.htmlValue);  creditCardForm.number.invalidateFormElement();  break;  case PaymentStatusCodes.CREDITCARD\_INVALID\_EXPIRATION\_DATE:  creditCardForm.expiration.month.invalidateFormElement();  creditCardForm.expiration.year.invalidateFormElement();  break;  case PaymentStatusCodes.CREDITCARD\_INVALID\_SECURITY\_CODE:  creditCardForm.cvn.invalidateFormElement();  }  }  return true;  }  module.exports = {  execute: execute,  invalidatePaymentCardForm: invalidatePaymentCardForm  }; |

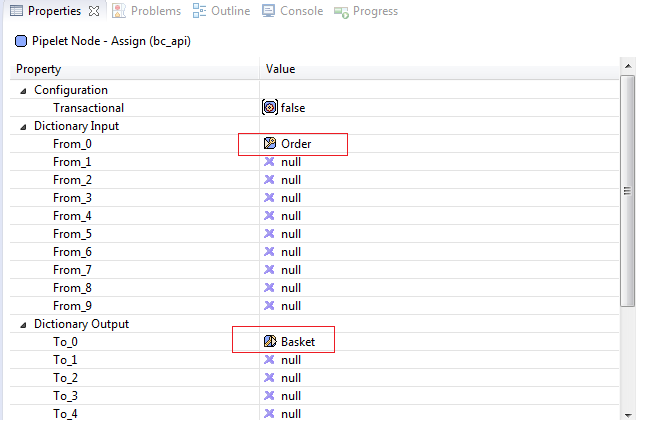
#### Pipeline - COPlaceOrder.xml

##### Update “CreateOrder” node

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

Note: This change is required only for Merchants using the Two Step Checkout Flow.

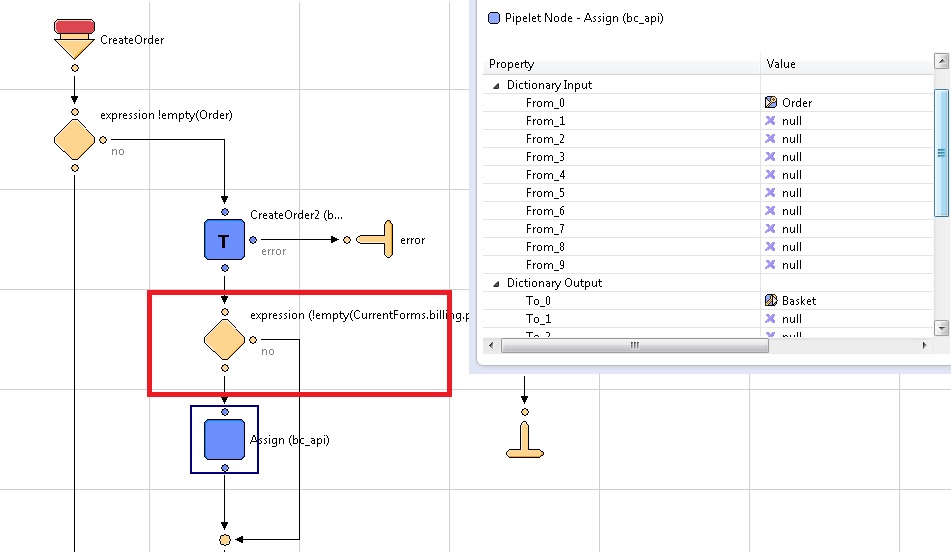




Note: Below mentioned change is required if merchant is using Paypal, Alipay payment methods along with Credit Card and BML Payment methods. COPlaceOrder-CreateOrder node need to update as mentioned below

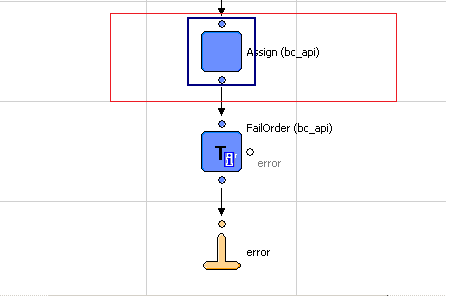
* Add below condition in the expression node to set the value of Order into Basket Object for Credit Card and BML as payment methods.

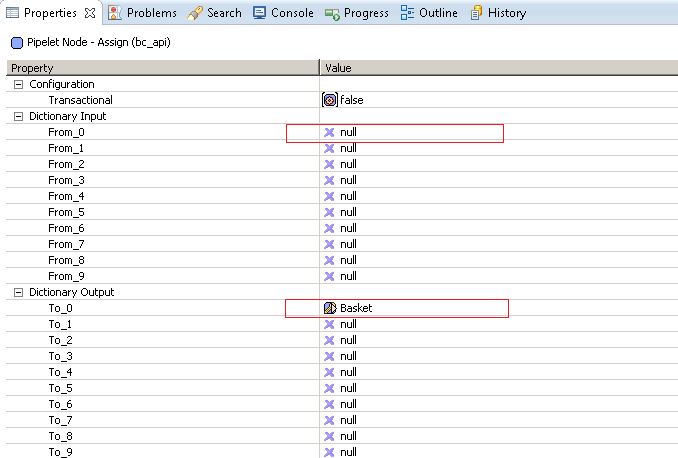
(!empty(CurrentForms.billing.paymentMethods.selectedPaymentMethodID) && (CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals('CREDIT\_CARD')||CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals('BML')))



* 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:





#### Controller - CYBERSOURCE\_CREDIT.js

##### Update “Authorize” node:

Update Authorize methods to call Cybersource AuthorizeCreditCard function.

|  |
| --- |
| **function** Authorize(args) {    **var** OrderMgr = require('dw/order/OrderMgr');  **var** app = require('~/cartridge/scripts/app');  **var** Cart = app.getModel('Cart');    **var** orderNo = args.OrderNo;  **var** Order = OrderMgr.getOrder(orderNo);  **var** PaymentInstrument = args.PaymentInstrument;    **var** Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource')    **return** Cybersource.AuthorizeCreditCard({PaymentInstrument:PaymentInstrument, Order:Order, Basket:Order});  } |

#### Controller – BASIC\_CREDIT.js

##### Update “Handle” node

Replace the handle with the changes below:

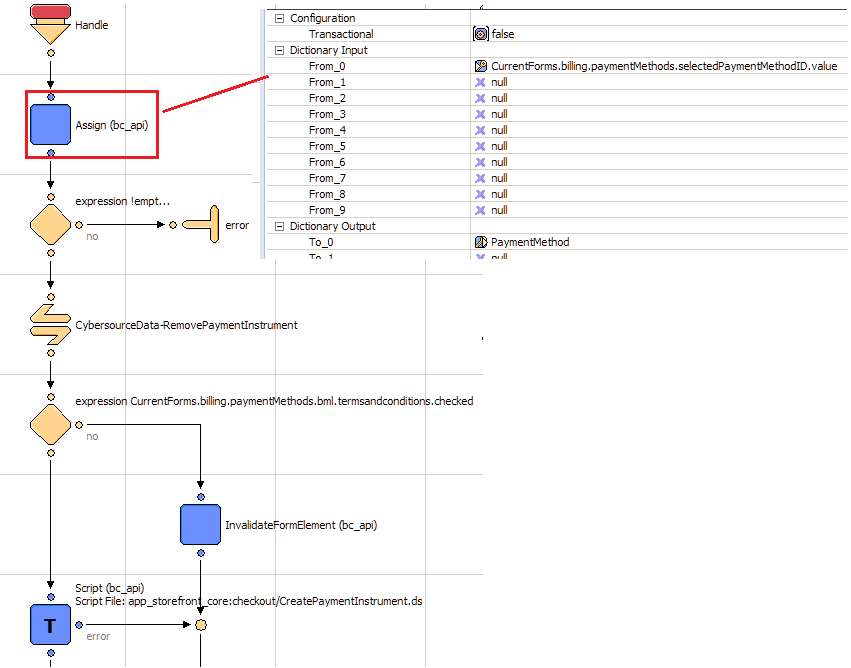
|  |
| --- |
| **function** Handle(args) {  **var** cart = Cart.get(args.Basket);  **var** PaymentMgr = require('dw/order/PaymentMgr');  **var** isToken = session.forms.billing.paymentMethods.creditCard.subscriptionToken.value;  **if** (!isToken) {  **var** VerifyPaymentCardResult = **new** dw.system.Pipelet('VerifyPaymentCard', {  VerifySecurityCode: **true**  }).execute({  PaymentCard: PaymentMgr.getPaymentCard(session.forms.billing.paymentMethods.creditCard.type.value),  CardNumber: session.forms.billing.paymentMethods.creditCard.number.value,  ExpirationMonth: session.forms.billing.paymentMethods.creditCard.expiration.month.value,  ExpirationYear: session.forms.billing.paymentMethods.creditCard.expiration.year.value,  CardSecurityCode: session.forms.billing.paymentMethods.creditCard.cvn.value  });    **if** (VerifyPaymentCardResult.result === PIPELET\_ERROR) {    **var** CreditCardStatus = VerifyPaymentCardResult.Status;    **new** dw.system.Pipelet('Script', {  Transactional: **false**,  OnError: 'PIPELET\_ERROR',  ScriptFile: 'app\_storefront\_core:checkout/InvalidatePaymentCardFormElements.ds'  }).execute({  CreditCardForm: session.forms.billing.paymentMethods.creditCard,  Status: CreditCardStatus  });    **return** {error: **true**};  }  }  Transaction.wrap(**function** () {  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_BML);  cart.removeExistingPaymentInstruments('PayPal');  cart.removeExistingPaymentInstruments('ALIPAY');  **var** paymentInstrument = cart.createPaymentInstrument(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD, cart.getNonGiftCertificateAmount());  paymentInstrument.creditCardHolder = session.forms.billing.paymentMethods.creditCard.owner.value;  paymentInstrument.creditCardNumber = session.forms.billing.paymentMethods.creditCard.number.value;  paymentInstrument.creditCardType = session.forms.billing.paymentMethods.creditCard.type.value;  paymentInstrument.creditCardExpirationMonth = session.forms.billing.paymentMethods.creditCard.expiration.month.value;  paymentInstrument.creditCardExpirationYear = session.forms.billing.paymentMethods.creditCard.expiration.year.value;  **if** (isToken) {  paymentInstrument.setCreditCardToken(session.forms.billing.paymentMethods.creditCard.subscriptionToken.value);  }  });  **return** {success: **true**};  } |

### Bill Me Later

#### Pipeline - CYBERSOURCE\_BML.xml

##### Update “Handle” Node

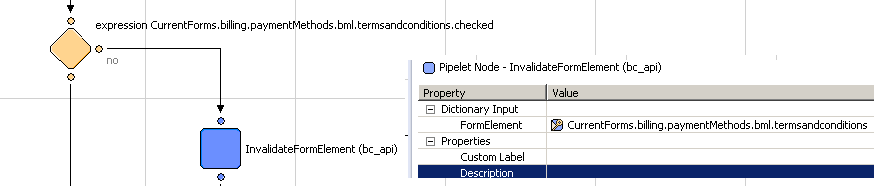
1. Assign node set variable “PaymentMethod” with “CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value”



1. Check decision node with condition

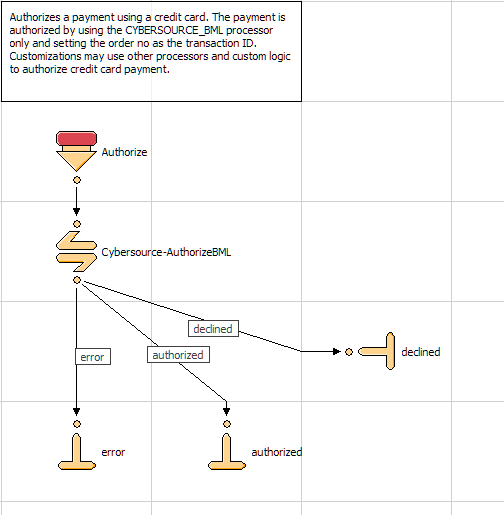
!empty(PaymentMethod) && PaymentMethod.equals('BML')

1. Call node “CybersourceData-RemovePaymentInstrument”
2. Continue with the existing changes [after step 3]



##### Update Authorize node

1. Call Cybersource AuthorizeBML
2. Return the results as “error”, “declined” and “authorized” end nodes



#### Controller - CYBERSOURCE\_BML.js

##### Update “Handle” Function

1. Retrieve basket
2. Check for terms and conditions checkbox checked on billing page
3. If not checked then invalidate the form and return {error: true;}
4. Else in transaction wrap perform
5. Remove all existing Payment instruments from basket related to cybersource payment providers like [paypal, alipay, credit card, BML].
6. Create new payment instrument for BML
7. Finally return success

|  |
| --- |
| Function Handle(args) {  var cart = Cart.get(args.Basket);  if (!session.forms.billing.paymentMethods.bml.termsandconditions.checked) {  session.forms.billing.paymentMethods.bml.termsandconditions.invalidateFormElement();  return {error: true};  } else {  Transaction.wrap(function () {  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_BML);  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  cart.removeExistingPaymentInstruments('PayPal');  cart.removeExistingPaymentInstruments('ALIPAY');  cart.createPaymentInstrument(dw.order.PaymentInstrument.METHOD\_BML, cart.getNonGiftCertificateAmount());  });  return {success: true};  }  } |

##### Update Authorize Function

1. Get order object
2. Call Cybersource AuthorizeBML and pass above order object
3. return the results

Replace Authorize method with below

|  |
| --- |
| function Authorize(args) {  var OrderMgr = require('dw/order/OrderMgr');  var app = require('~/cartridge/scripts/app');  var Cart = app.getModel('Cart');    var orderNo = args.OrderNo;  var Order = OrderMgr.getOrder(orderNo);    var Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource')    return Cybersource.AuthorizeBML({Order:Order, Basket:Order});  } |

### Tax Service

#### Script - calculate.js

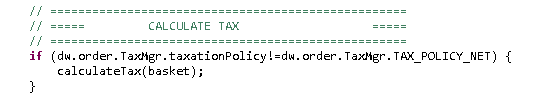
##### Avoid CalculateTax builtin Function Call for NET taxation

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.

**if** (dw.order.TaxMgr.taxationPolicy!=dw.order.TaxMgr.TAX\_POLICY\_NET) {

calculateTax(basket);

}



#### Controller Cartridge – Script CartModel.js

##### Update Calculate function

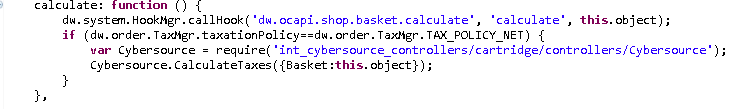
Add condition for calling calculate taxes of cybersource after hook call

**if** (dw.order.TaxMgr.taxationPolicy==dw.order.TaxMgr.TAX\_POLICY\_NET) {

**var** Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');

Cybersource.CalculateTaxes({Basket:**this**.object});

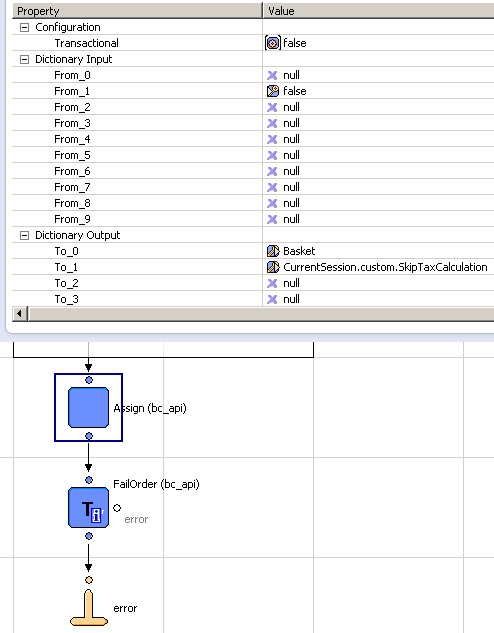
}



##### Pipeline –COPlaceOrder.xml

###### Update FailImpl Node

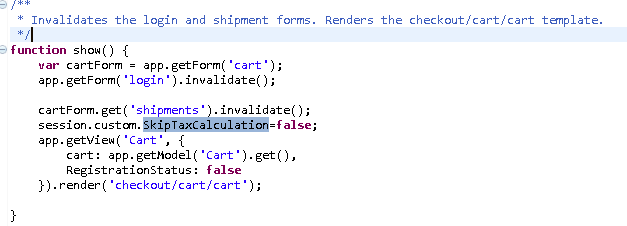
1. Add assign node before FailOrder pipelet where set variable in session SkipTaxCalculation as false



##### Controller - Cart.js

###### Update “Show” function

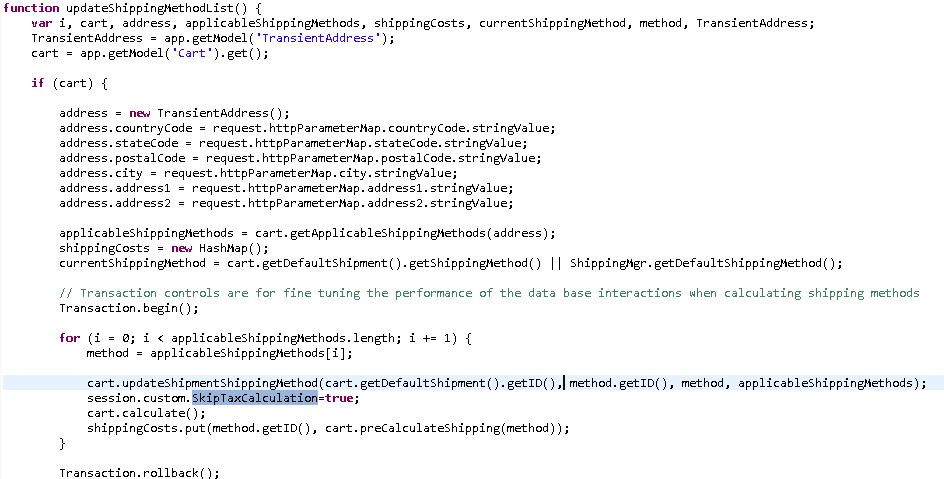
1. Set variable session.custom.SkipTaxCalculation=false; before template render



##### Controller – COShipping.js

###### Update “updateShippingMethodList“ function

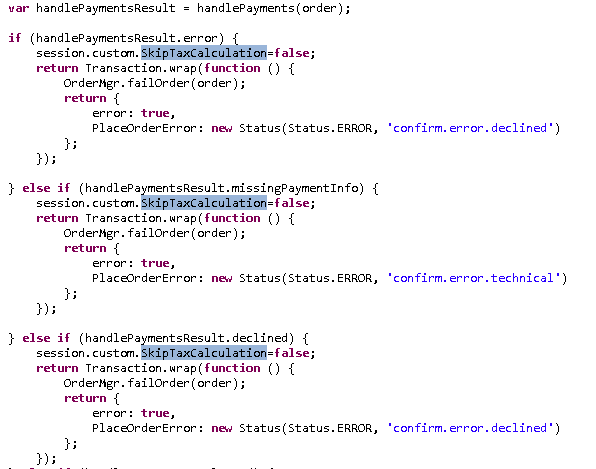
1. Set session variable session.custom.SkipTaxCalculation=true;before cart calculate

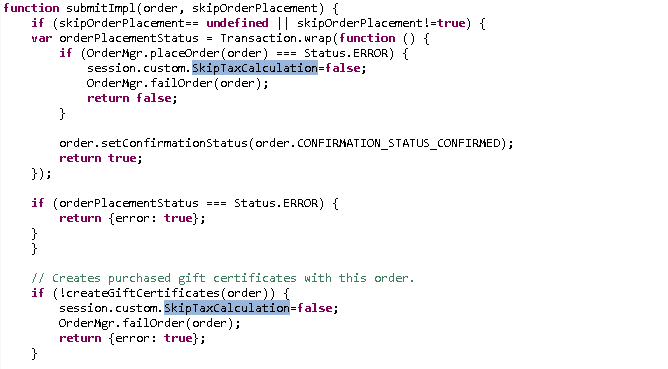


##### Controller – COPlaceOrder.js

###### Update “start“and “submitImpl” function

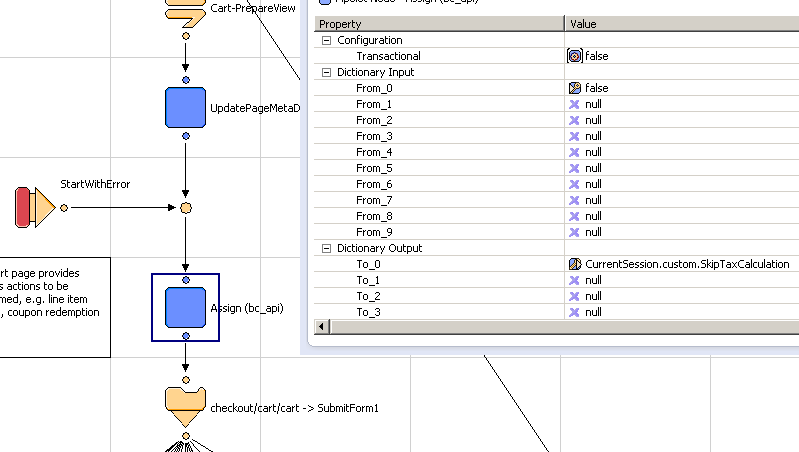
1. Set session variable SkipTaxCalculation set as false in payment ERROR scenarios





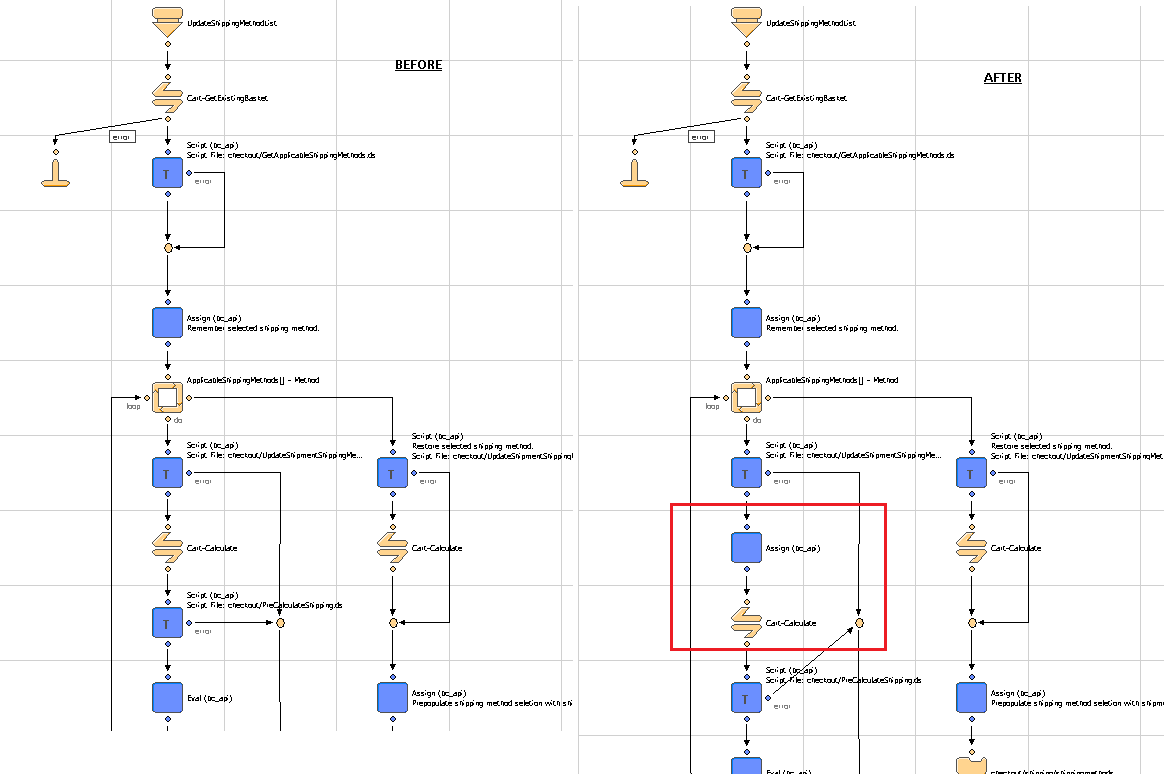
##### Pipeline - Cart.xml

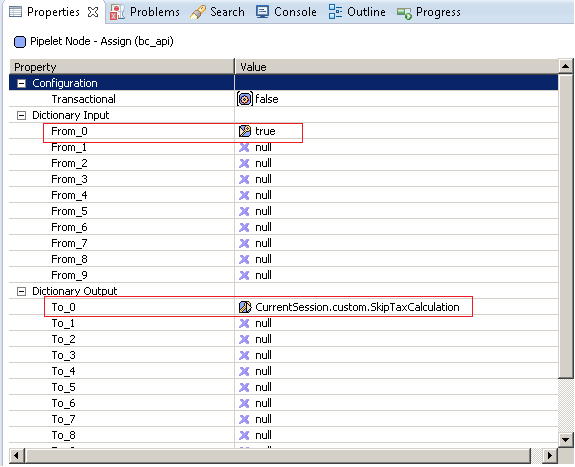
1. Add assign node before cart isml render in pipeline Cart-Show and set CurrentSession.custom.SkipTaxCalculation as false



##### Pipeline – COShipping.xml

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





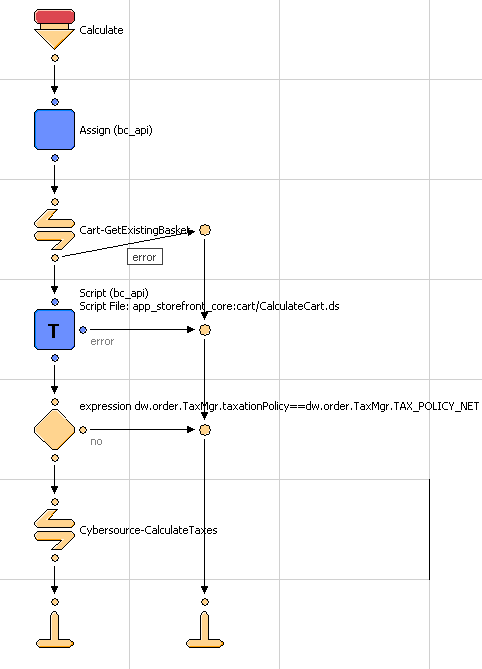
#### Pipeline –Cart.xml

##### 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.

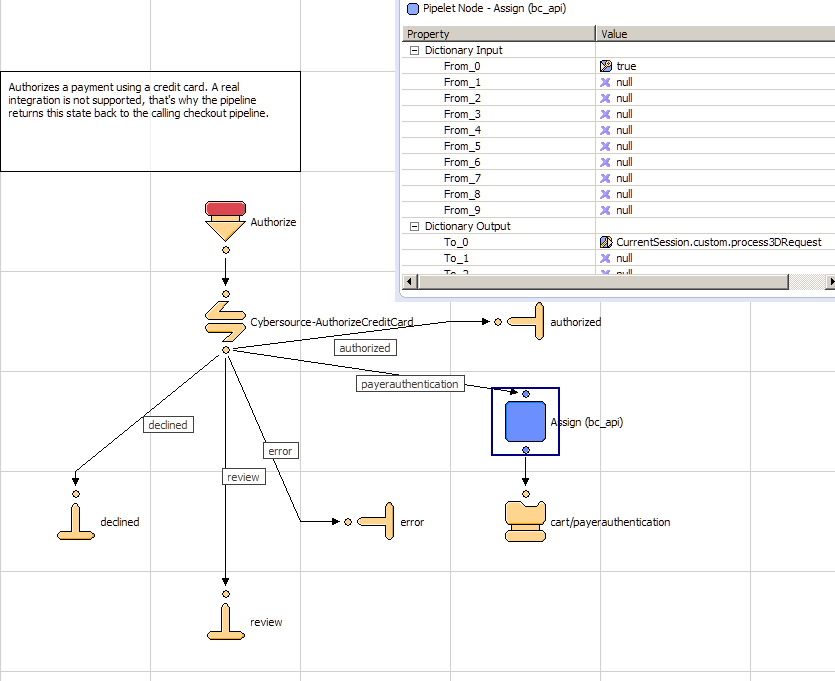
CyberSource Save Proof.xml (PA): –Determines whether to save proof.xml (received from Cybersource response) as part of order object.

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

#### Pipeline - CYBERSOURCE\_CREDIT.xml

##### Update “Authorize” node

Set the current session in assign node to process 3D request as below



### Payment Tokenization Service

#### Script billing.js [Complied into app.js]

##### Update setCCFields function.

* Add the following code block at the end of setCCFields function

$creditCard.find('[name$="creditCard\_subscriptionToken"]').val(data.subscriptionToken).trigger('change');

##### Update exports.init = function () function.

Add following code block after var selectedPaymentMethod = $selectPaymentMethod.find(':checked').val(); code line in above mentioned function

|  |
| --- |
| **var** $ccContainer = $($checkoutForm).find(".payment-method").filter(**function**(){  **return** $(**this**).data("method")=="CREDIT\_CARD";  });  **var** $ccNum = $ccContainer.find("input[name$='\_number']");    $ccContainer.find('input[name\*="\_number"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_subscriptionToken"]').val('');  });  $ccContainer.find('input[name$="\_owner"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_subscriptionToken"]').val('');  });  $ccContainer.find('select[name$="creditCard\_type"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_subscriptionToken"]').val('');  });  $ccContainer.find('select[name\*="expiration"]').on('change',**function**(e){  $($checkoutForm).find('input[name$="\_subscriptionToken"]').val('');  }); |

#### Form - creditcard.xml

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

|  |
| --- |
| <!-- field for credit card subscription -->  <field formid=*"subscriptionToken"* type=*"string"* masked=*"4"* /> |

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"*/>

#### Template - creditcardjson.isml

Retrieve subscription token of saved card to be used further

|  |
| --- |
| expirationYear:pdict.SelectedCreditCard.creditCardExpirationYear,  subscriptionToken:pdict.SelectedCreditCard.getCreditCardToken() |

#### Form - customeraddress.xml

Include the following code just before adding 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> |

#### Form - paymentinstruments.xml

Include address fromId just below new credit card Id

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

#### Template - paymentmethods.isml

##### New Hidden Field

Include the following code right after this line :

<isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.type}" type="select"/>

|  |
| --- |
| <isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.subscriptionToken}" type="hidden"/> |

#### 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

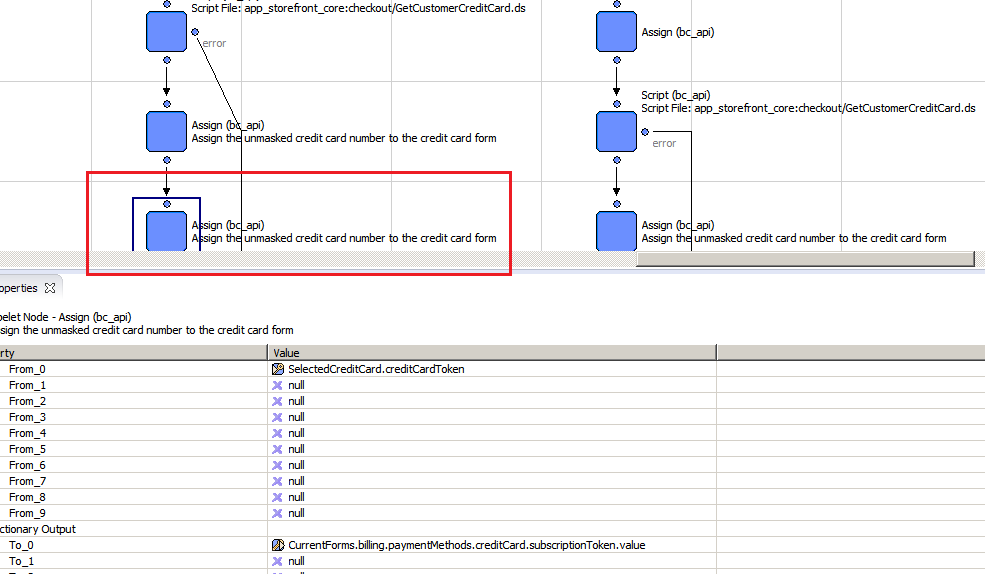
Add below resource bundle properties at end of account.properties file to display error messages

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

#### Pipeline - COBilling.xml

##### Update “SelectCreditCard” node

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

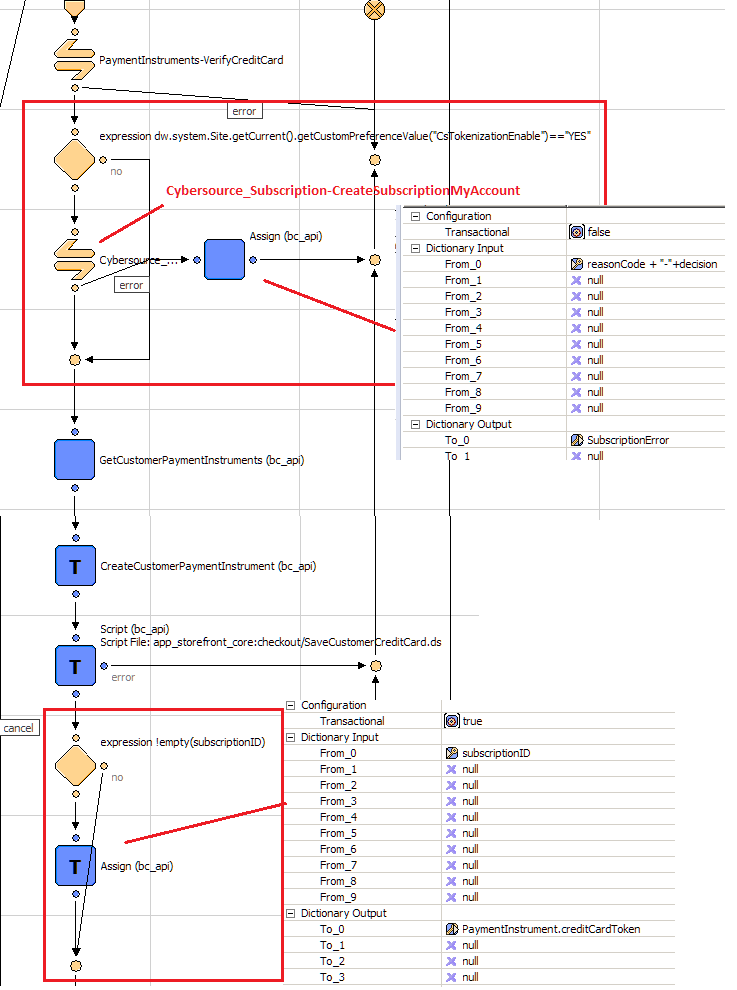


#### My Account - Pipeline – PaymentInstruments.xml

##### 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



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

Input

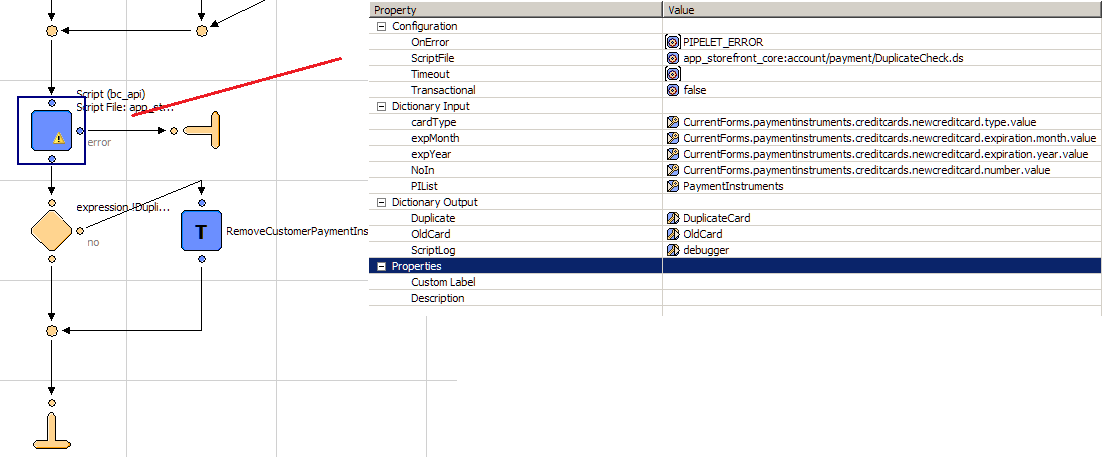
|  |  |
| --- | --- |
| cardType | CurrentForms.paymentinstruments.creditcards.newcreditcard.type.value |
| expMonth | CurrentForms.paymentinstruments.creditcards.newcreditcard.expiration.month.value |
| expYear | CurrentForms.paymentinstruments.creditcards.newcreditcard.expiration.year.value |
| NoIn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

Output:

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

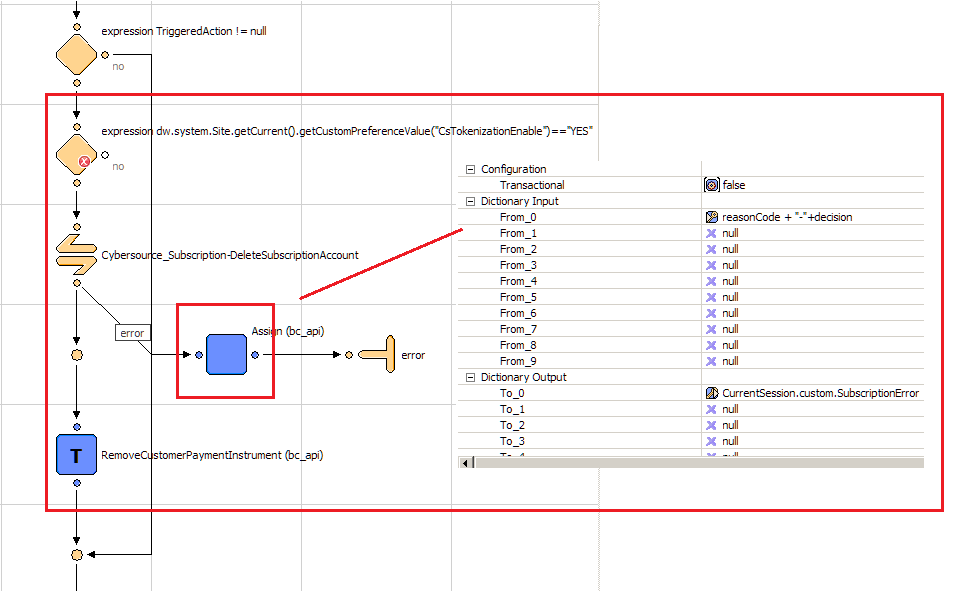
Continue with the existing flow further.

Refer below:



##### Update “Delete” node

1. Add a Decision node as **dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable")=="YES"** 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. Add an end node as 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

#### Controller - COBilling.js

##### Update “selectCreditCard” function

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

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

#### 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 as below code with **if** (!create()) {

add(**false**); 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

Replace 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') {  **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 |

#### Pipeline - CYBERSOURCE\_ALIPAY.xml

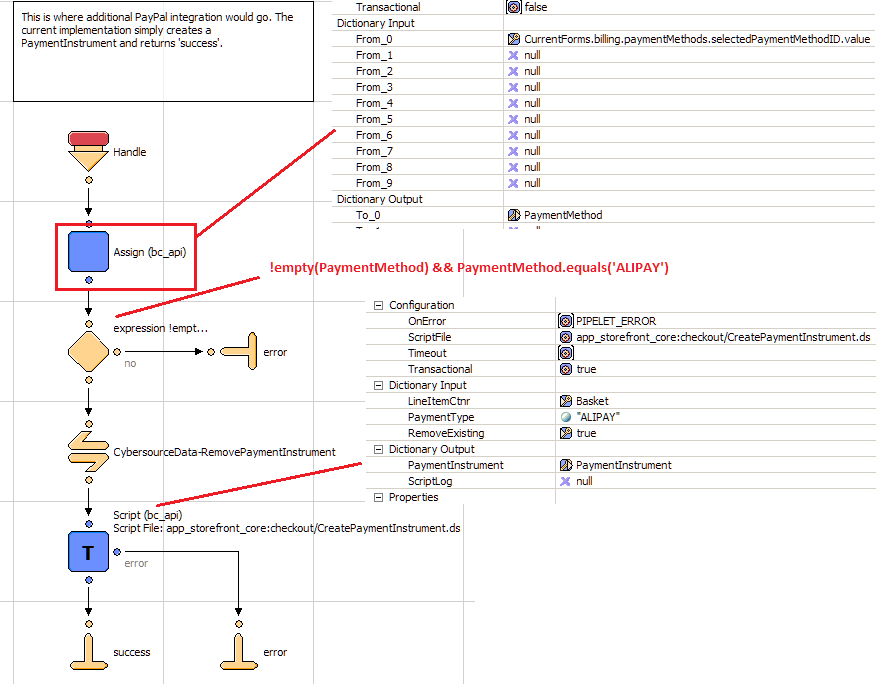
##### New Pipeline CYBERSOURCE\_ALIPAY

1. Add CYBERSOURCE\_ALIPAY pipeline under Storefront Pipelines Cartridge to call Cybersource-AuthorizeAlipay pipeline.

**Note:** Alternatively copy the sample file fromcybersource cartridge to storefront cartridge

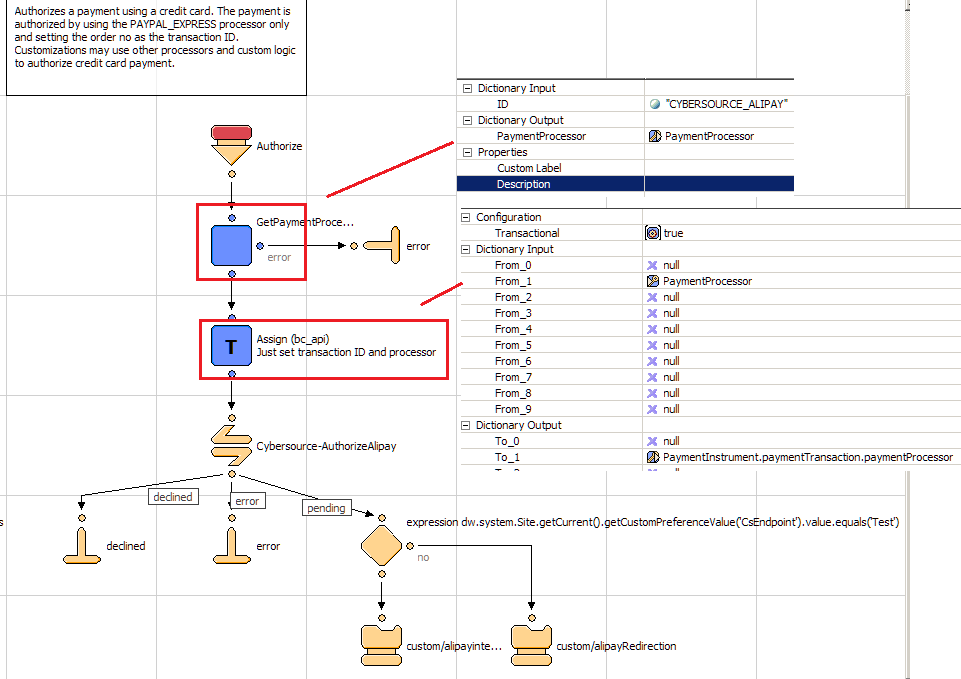
##### Create “Handle” node

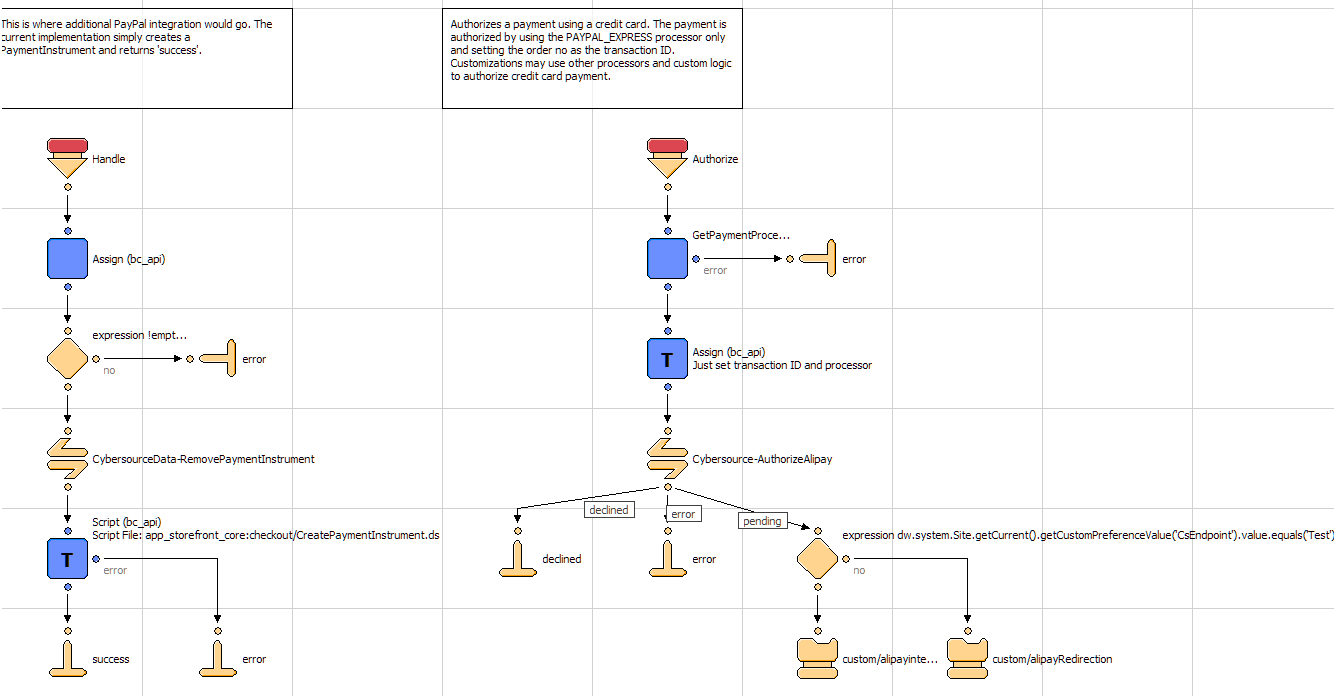
Create a new handle as below



##### Create “Authorize” node

Create new “Authorize” handle as below





Note: Once ALIPAY authorization is successful l, the site is redirected to ALIPAY test simulator (if enable from Business manager) and return back to 'COPlaceOrder-Submit’ to confirm the order.

#### Pipeline - COPlaceOrder.xml

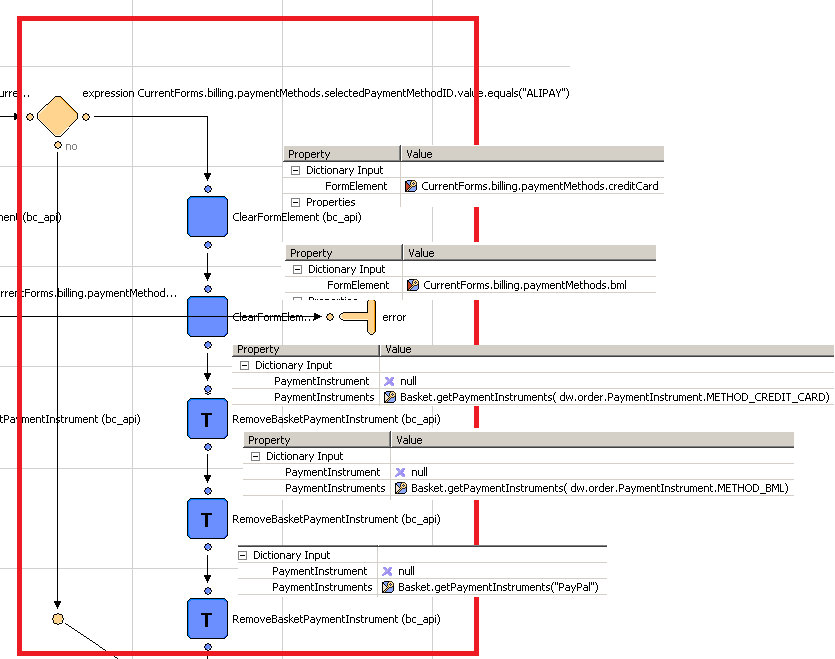
##### Update “Submit” Node

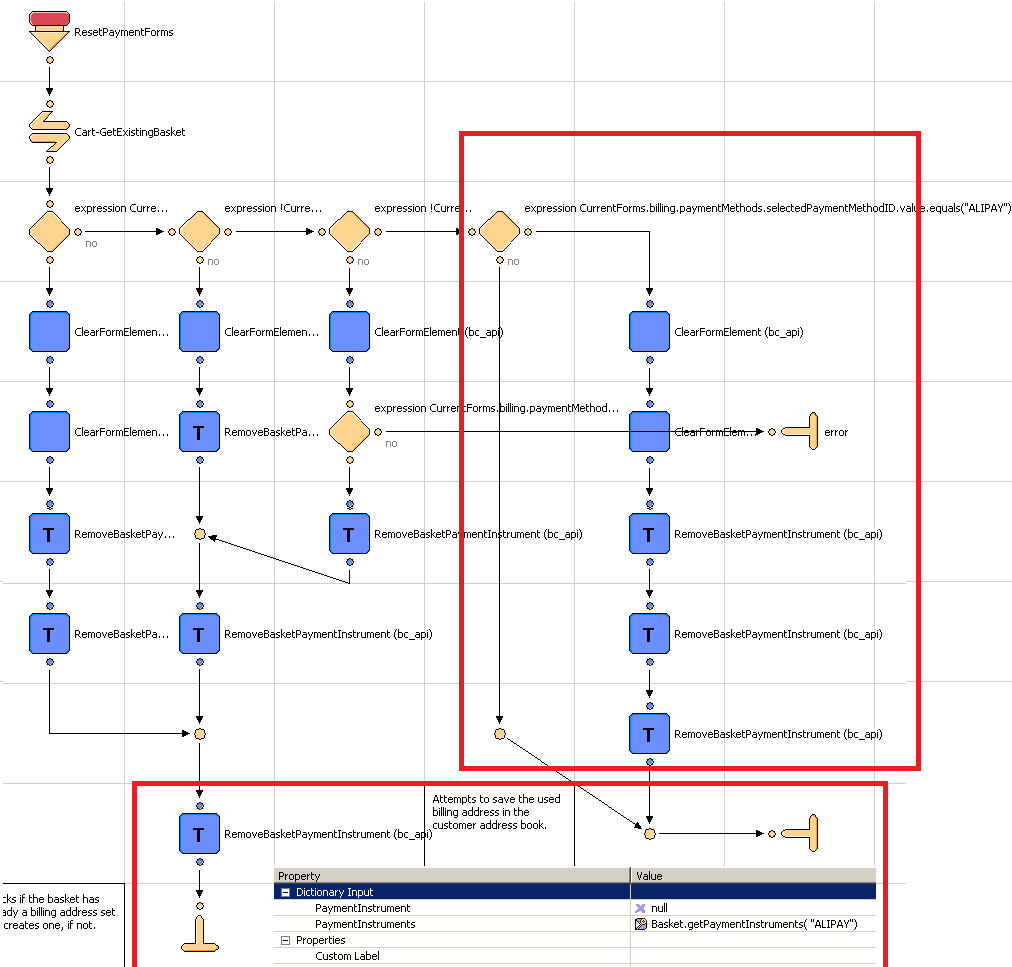
Ensure Generic section “Order Placement for Alipay” changes implemented

#### Pipeline - COBilling.xml

##### Update “ResetPaymentForms” node

1. This node use to reset the payment forms excluding the selected payment method on billing page
2. Add condition for selected payment method as ALIPAY and remove all other payment instruments
   1. CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals("ALIPAY")
3. Clear BML and Card Form by calling pipelet “ClearFormElement”
4. Remove payment instruments for Card, BML and Paypal use pipelet “removeBasketPaymentInstrument”





#### Controller - CYBERSOURCE\_ALIPAY.js

##### New Controller CYBERSOURCE\_ALIPAY.js

1. Add new CYBERSOURCE\_ALIPAY.js controller under Storefront Controller Cartridge and create Handle and Authorize functions to make a call for Alipay Authorization.

**Note**: Alternatively copy sample file from cybersource controller cartridge to storefront controller cartridge

|  |
| --- |
| 'use strict';  /\* API Includes \*/  **var** Cart = require('~/cartridge/scripts/models/CartModel');  **var** PaymentMgr = require('dw/order/PaymentMgr');  **var** Transaction = require('dw/system/Transaction');  **var** app = require('~/cartridge/scripts/app');  /\*\*  \* This is where additional BillMeLaterl integration would go. The current implementation simply creates a payment  \* method and returns 'success'.  \*/  **function** Handle(args) {  **var** PaymentMethod = session.forms.billing.paymentMethods.selectedPaymentMethodID.value;  **if** (!empty(PaymentMethod)) {  **var** cart = Cart.get(args.Basket);  **var** isPaymentInstrumentCreated = **false**;  Transaction.wrap(**function** () {  cart.removeExistingPaymentInstruments(PaymentMethod);  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_BML);  cart.removeExistingPaymentInstruments('PayPal');  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);    cart.createPaymentInstrument(PaymentMethod, cart.getNonGiftCertificateAmount());  isPaymentInstrumentCreated = **true**;  });  **if** (isPaymentInstrumentCreated) {  **return** {success: **true**};  }  }  **return** {error:**true**};  }  /\*\*  \* Authorizes a payment using a credit card. The payment is authorized by using the CYBERSOURCE\_BML processor only and  \* setting the order no as the transaction ID. Customizations may use other processors and custom logic to authorize  \* credit card payment.  \*/  **function** Authorize(args) {  **var** OrderMgr = require('dw/order/OrderMgr');  **var** app = require('~/cartridge/scripts/app');  **var** Cart = app.getModel('Cart');    **var** orderNo = args.OrderNo;  **var** Order = OrderMgr.getOrder(orderNo);  **var** paymentInstrument = args.PaymentInstrument;  **var** paymentProcessor = PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();  Transaction.wrap(**function** () {  paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;  });    **var** Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');    **var** alipayResult = Cybersource.AuthorizeAlipay({Order:Order, orderNo:orderNo, PaymentInstrument: paymentInstrument});  **if** (alipayResult.pending){  app.getView({alipayReturnUrl:alipayResult.alipayReturnUrl}).render('custom/alipayintermediate');  **return** {end:**true**};  } **else** {  **return** alipayResult;  }  }  /\*  \* Module exports  \*/  /\*  \* Local methods  \*/  exports.Handle = Handle;  exports.Authorize = Authorize; |

#### Controller - COPlaceOrder.js

##### Update “submit” Function

1. This function use for handling responses returned from payment processor Authorize results. This is asynchronous callback function which identifies the order exists based on order token available in URL and session order, when matched then decide based on response to place order or fail order
2. Check if the order has ALIPAY payment instrument then invoke cybersource function “CheckAlipayPaymentStatus”
3. This cybersource function check the payment status of initiated payment request through alipay and change the status in demandware for placed order to NEW, CREATED, FAILED after getting from service call response in relation to Alipay payment status such as COMPLETED, PENDING, ABANDONED and TRADE\_NOT\_EXIST.
4. If response received by above function is summaryconfirmation then
5. send email to customer
6. clear forms and session attributes
7. If response received by above function is error then call “COPlaceOrder-Fail” function

|  |
| --- |
| /\*  \* Asynchronous Callbacks for SiteGenesis.  \* Identifies if an order exists, submits the order, and shows a confirmation message.  \*/  **function** submit(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;  **if** (!empty(order.getPaymentInstruments('ALIPAY'))) {  **var** alipayResult = Cybersource.CheckAlipayPaymentStatus({Order:order});  **if** (alipayResult.summaryconfirmation) {  // 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.  session.forms.singleshipping.clearFormElement();  session.forms.multishipping.clearFormElement();  session.forms.billing.clearFormElement();  session.custom.cartStateString=**null**;  app.getController('COSummary').ShowConfirmation(order);  **return**;  } **else** **if** (alipayResult.error)  {  app.getController('COPlaceOrder').Fail({Order:order});  **return**;  }  } |

#### Controller - COBilling.js

##### Update “resetPaymentForms” Function

1. This function use to reset the payment forms excluding the selected payment method on billing page
2. Add below line to remove the ALIPAY payment instrument in all possible selected payment method for resetting of payment forms

cart.removePaymentInstruments(cart.getPaymentInstruments("ALIPAY"));

1. Add condition for selected payment method as ALIPAY and remove all other payment instruments

|  |
| --- |
| **function** resetPaymentForms() {  **var** cart = app.getModel('Cart').get();  **var** status = Transaction.wrap(**function** () {  **if** (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals('PayPal')) {  app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();  app.getForm('billing').object.paymentMethods.bml.clearFormElement();  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD));  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_BML));  cart.removePaymentInstruments(cart.getPaymentInstruments("ALIPAY"));  } **else** **if** (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_CREDIT\_CARD)) {  app.getForm('billing').object.paymentMethods.bml.clearFormElement();  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_BML));  cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));  cart.removePaymentInstruments(cart.getPaymentInstruments("ALIPAY"));  } **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));  cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));  cart.removePaymentInstruments(cart.getPaymentInstruments("ALIPAY"));  } **else** **if**(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals("ALIPAY")){  app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();  app.getForm('billing').object.paymentMethods.bml.clearFormElement();  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD));  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_BML));  cart.removePaymentInstruments(cart.getPaymentInstruments("PayPal"));  }  **return** **true**;  });  **return** status;  } |

#### Controller Cartridge - hooks.json

##### ALIPAY payment processor hook addition

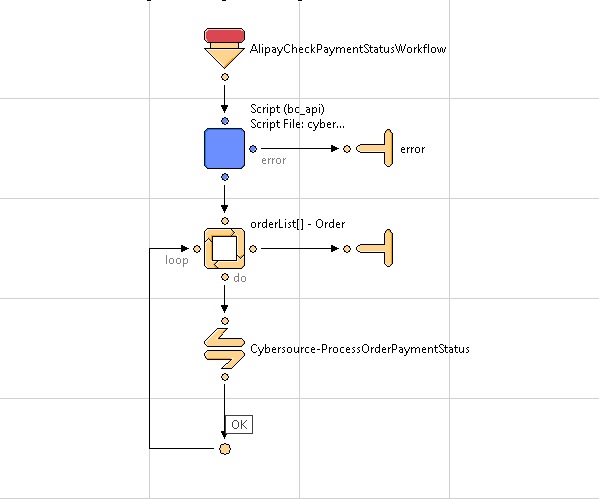
1. Add a hook for payment processor as CYBERSOURCE\_ALIPAY at the end of hooks.json in cartridge app\_storefront\_controllers

|  |
| --- |
| ,  {  "name": "app.payment.processor.CYBERSOURCE\_ALIPAY",  "script": "./payment/processor/CYBERSOURCE\_ALIPAY"  } |

### Alipay Batch Job

#### Cybersource- AlipayCheckPaymentStatusWorkflow

This pipeline is used to process the orders in Demandware after getting response from Alipay Check Payment Status service. This pipeline will be called from job scheduler to change the status of Demandware orders. It will process all the orders which are in New, Open and Created state and will update the order status after getting response from service call.



### 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.

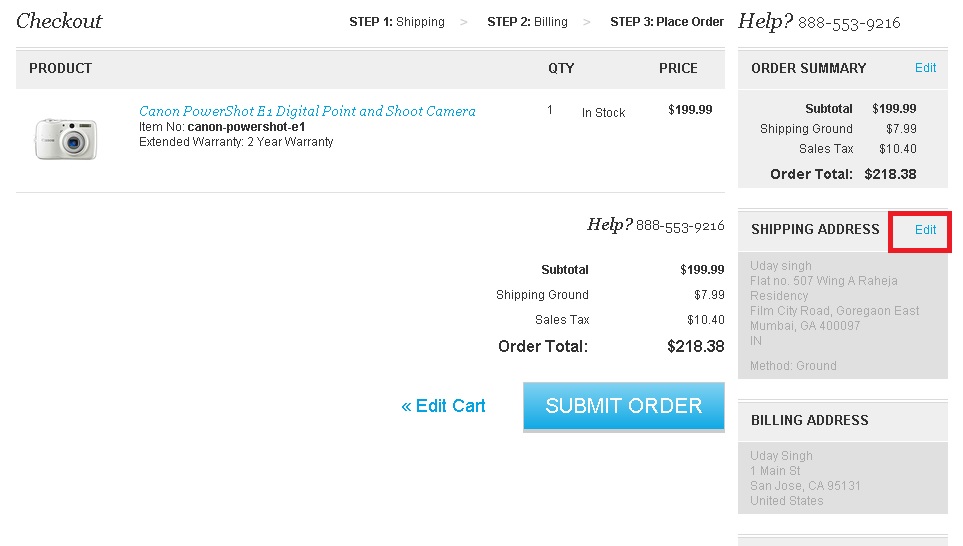
|  |
| --- |
| <isif condition=*"${enableCheckout}"*>  <button class=*"button-fancy-large"* type=*"submit"* value=*"${Resource.msg('global.checkout','locale',null)}"* name=*"${pdict.CurrentForms.cart.checkoutCart.htmlName}"*>  ${Resource.msg('global.checkout','locale',null)}  </button>  <isif condition=*"${dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==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> |

Note: Below mention changes for PayPal Express is merchant specific. Merchant can refer to the changes mentioned below in case of below mentioned requirement.

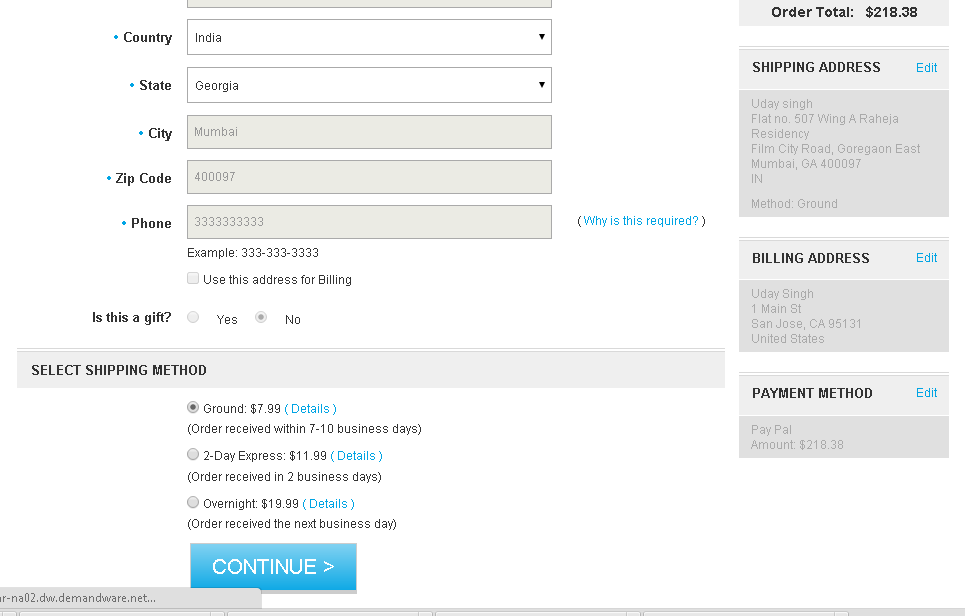
Customization has been done for PayPal Express Checkout when user choose PayPal Express as Payment option and user redirected back to Demandware order summary page after PayPal, user has a provision to change only Shipping method. After user clicks on Edit button of shipping address as mentioned below, User will redirect to shipping page with all the fields disabled other than shipping method.

Refer to the screen shot below for Edit button on Order summary page.

Refer to the screen shot below for Edit button on Order summary page.



Refer to the screen shot below for disabled rest of the fields other than shipping method.



After selecting payment method user will be redirected back to order summary page and can place the order with updated shipping method.

Please refer the code changes for the above mentioned requirement.

#### 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 |

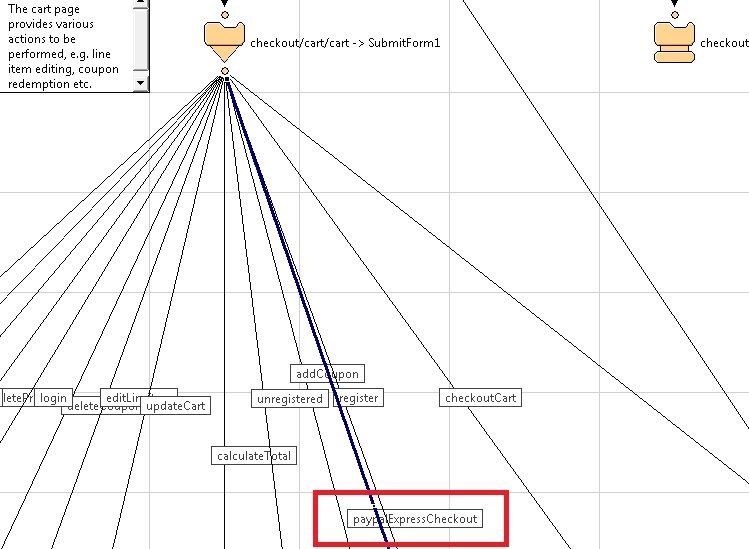
#### Pipeline - Cart.xml

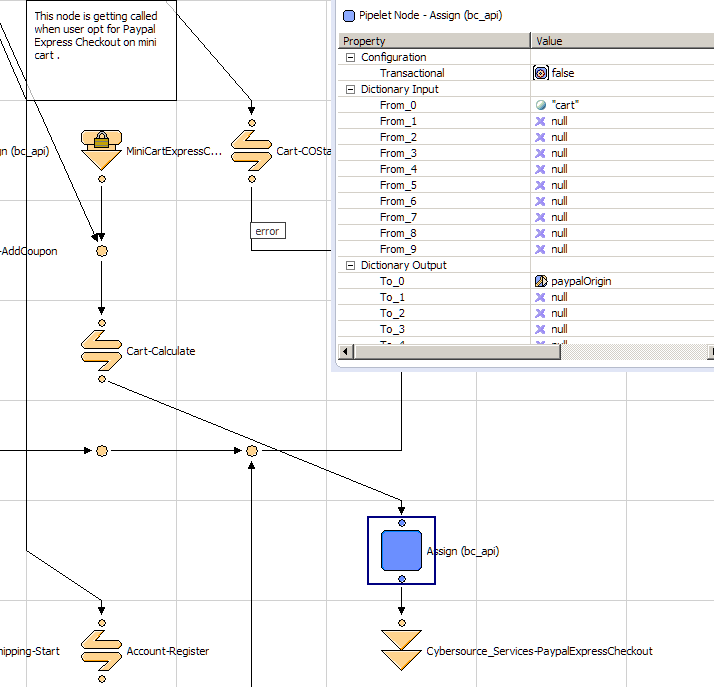
#### Pipeline - Cart.xml

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.





#### Pipeline - PAYPAL\_EXPRESS.xml

##### Update “Handle” Node

Refer Paypal Checkout[From billing page]

##### Update Authorize Node

1. Update assign node just after GetPaymentProcessor pipelet with below attributes as input and output:

Input: null, PaymentProcessor, null, null, null

Output: null, PaymentInstrument.paymentTransaction.paymentProcessor, CurrentSession.custom.correlationID, CurrentSession.custom.ppXpressRequestToken, CurrentSession.custom.ppXpressRequestId

1. Call Cybersource-PaypalOrderSetup , if response is OK, goto [step 3] else return error.
2. Call Cybersource-AuthorizePaypal and handle various responses

If declined, add assign node and set **new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined") for PlaceOrderError**

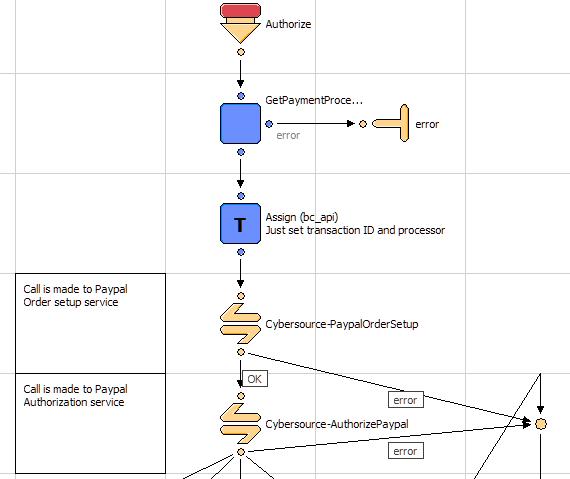
If review, add assign node and set **new dw.system.Status(dw.system.Status.ERROR, "confirm.error.review") for PlaceOrderError**

If pending, jump to **Cybersource-ProcessPendingOrders**

If error, return to error

If authorized, Add decision node **CybersourceHelper.getPaypalPaymentOption()=="AuthorizeAndCapture"**

If TRUE, call **Cybersource-CapturePaypal and authorize the transition else**



#### 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);    } |

#### Controller - PAYPAL\_EXPRESS.js

##### Variable declarations

Include API at the start of the file as below:

|  |
| --- |
| var Transaction = require('dw/system/Transaction');  var Pipeline = require('dw/system/Pipeline');  var OrderMgr = require('dw/order/OrderMgr');  var PaymentMgr = require('dw/order/PaymentMgr');  //TODO  var app = require('~/cartridge/scripts/app'); |

##### Update “Authorize” function

To handle different PayPal responses

|  |
| --- |
| function Authorize(args) {  var orderNo = args.OrderNo;  var order = OrderMgr.getOrder(orderNo);  var paymentInstrument = args.PaymentInstrument;  var paymentProcessor = PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();  Transaction.wrap(function () {  // paymentInstrument.paymentTransaction.transactionID = orderNo;  paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;  });  var Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');  var pdict = Cybersource.PaypalOrderSetup({Order:order, PaymentInstrument: paymentInstrument});  if(pdict.EndNodeName == "OK"){  var paypalResult = Cybersource.AuthorizePaypal({Order:order, PaymentInstrument: paymentInstrument, orderSetupRequestId:pdict.orderSetupRequestId,  orderSetupRequestToken:pdict.orderSetupRequestToken,  orderSetupTransactionId:pdict.orderSetupTransactionId});  if (paypalResult.review){  var PlaceOrderError = new dw.system.Status(dw.system.Status.ERROR, "confirm.error.review");  return {error: true, PlaceOrderError : PlaceOrderError};  } else if (paypalResult.declined) {  var PlaceOrderError = new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined");  return {declined: true, PlaceOrderError : PlaceOrderError};  } else {return paypalResult; }}  return { error:true }  } |

### 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 – PAYPAL\_EXPRESS.xml

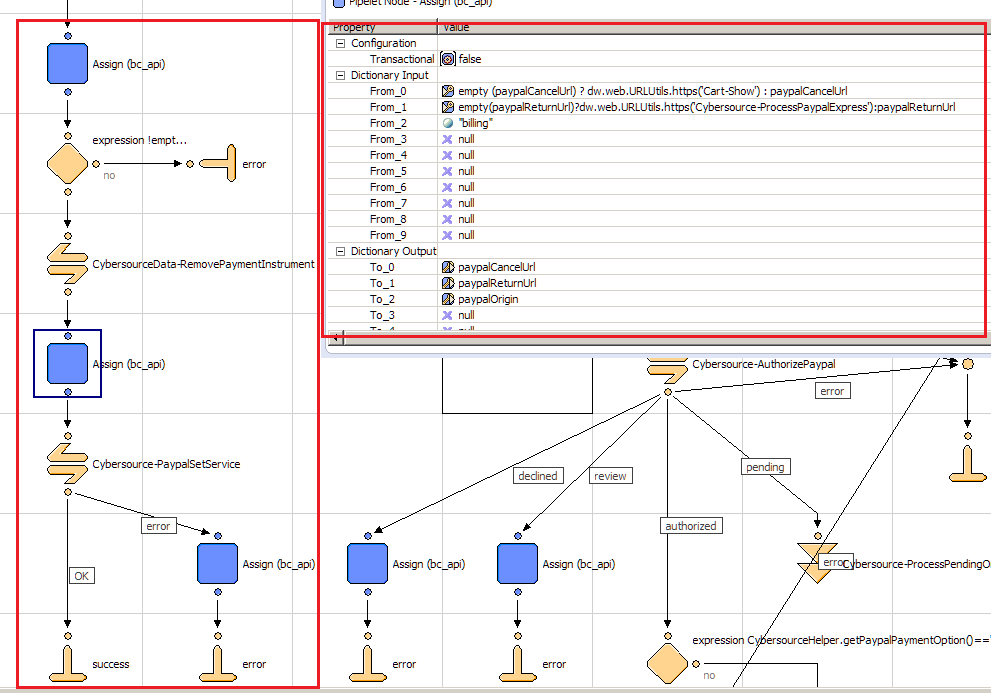
##### Update Handle Node

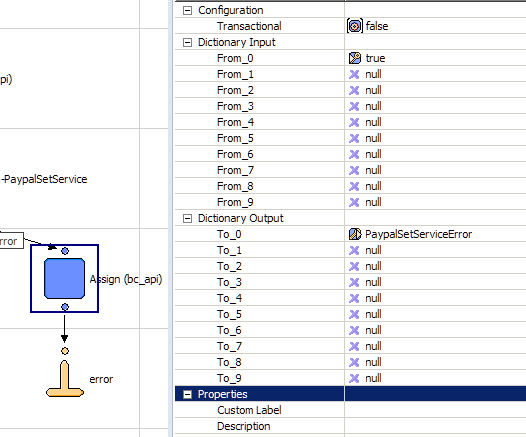
1. Add assign node and set input as **CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value and output as PaymentMethod**
2. Add decision not to check if existing payment method is Paypal as  **!empty(PaymentMethod) && PaymentMethod.equals('PayPal')**
3. Call CybersourceData-RemovePaymentInstrument to remove existing payment instruments
4. Add assign node and set below attributes as input:
5. empty (paypalCancelUrl) ? dw.web.URLUtils.https('COBilling-Start') : paypalCancelUrl
6. empty(paypalReturnUrl)?dw.web.URLUtils.https('Cybersource-ProcessPaypalExpress'):paypalReturnUrl
7. "billing"

As output:

1. paypalCancelUrl
2. paypalReturnUrl
3. paypalOrigin
4. Call Cybersource-PaypalSetService
5. On error set in assign node variable “PaypalSetServiceError” as true
6. End the Paypal transition with “end” and “success” node respectively.

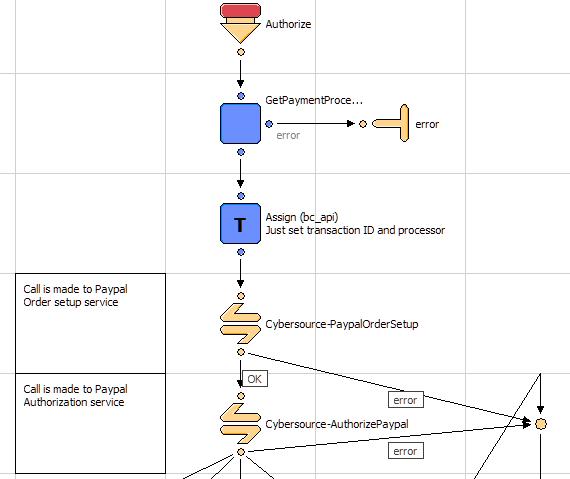
Refer below screen shot





##### Update Authorize Node

1. Update Authorize node to call Paypal Order Setup, Authorize and Capture , Process the pending orders from PayPal, and handle the various responses return from PayPal.
2. Alternatively copy sample file from cybersource cartridge to storefront cartridge



#### Pipeline - COShipping.xml

##### Update “InitForms” Node

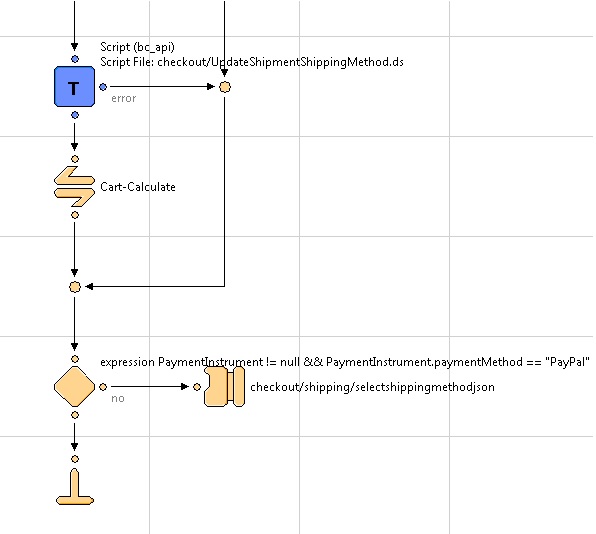
1. In COShipping pipeline also change the COShipping-InitForms at the bottom and add a condition for PayPal as !empty(Basket.getPaymentInstruments()) && !empty(Basket.getPaymentInstruments('PayPal'))
2. Assign node set variable “PaymentMethod” as “PayPal”
3. CallCybersourceData-RemovePaymentInstrument



##### 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.

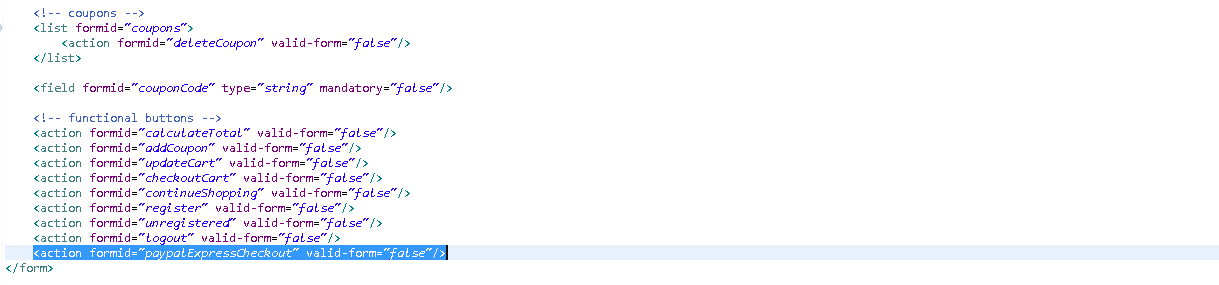
PaymentInstrument != null && PaymentInstrument.paymentMethod == "PayPal"



#### 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.

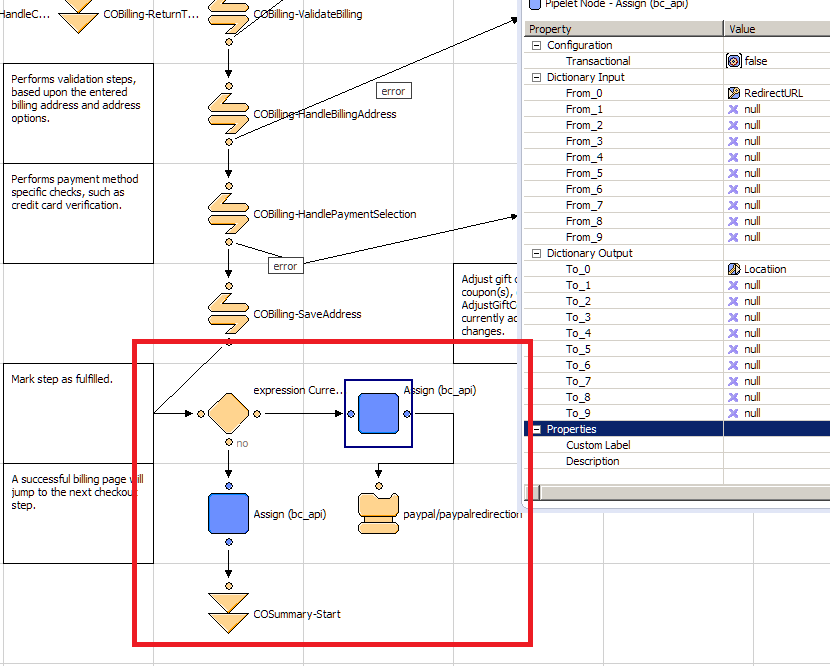
Note: Update PAYPAL\_EXPRESS-Handle and PAYPAL\_EXPRESS-Authorize node same as mentioned above for PayPal Express Checkout.

#### Pipeline - COBilling.xml

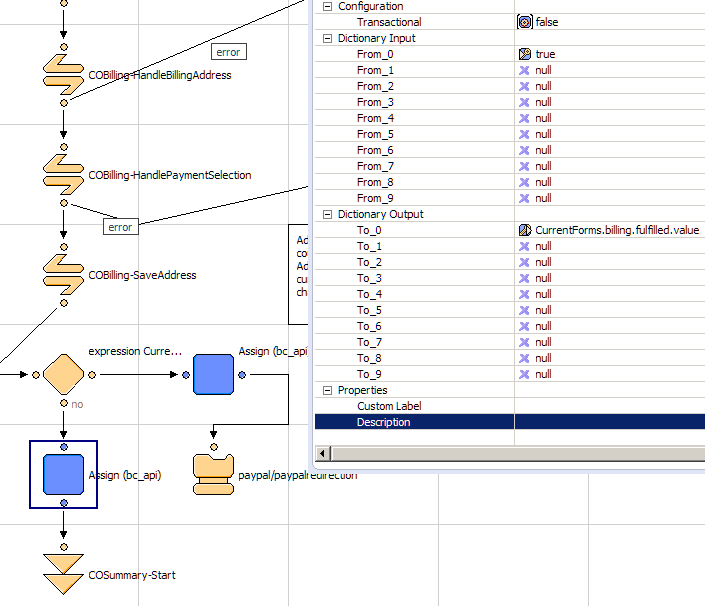
##### Update Start Node after save on billing page

Add below line in decision node to check for PayPal payment method and set PayPal redirection

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



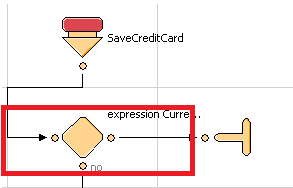
Set the value in assign node if selected payment method is not PayPal



##### Update “SaveCreditCard” Node

1. Ensure below condition available in first decision node to bypass save credit card logic when payment method selected as PayPal on billing page

CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals(dw.order.PaymentInstrument.METHOD\_BML) || CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals("PayPal") || CurrentForms.billing.paymentMethods.selectedPaymentMethodID.value.equals("ALIPAY") || !empty(Basket.getPaymentInstruments("PayPal"))



#### Controller - PAYPAL\_EXPRESS.js

##### Variable declarations

Include API at the start of the file as below:

|  |
| --- |
| var Transaction = require('dw/system/Transaction');  var Pipeline = require('dw/system/Pipeline');  var OrderMgr = require('dw/order/OrderMgr');  var PaymentMgr = require('dw/order/PaymentMgr');  //TODO  var app = require('~/cartridge/scripts/app'); |

##### Update “Handle” function

Replace the existing handle to process paypal from billing page

Changes are done to handle Paypal errors, remove payment instruments and call to Cybersource.PaypalSetService

|  |
| --- |
| **function** Handle(args) {  **var** cart = Cart.get(args.Basket);  **var** paymentMethod = app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value;  **if** (empty(paymentMethod) || paymentMethod !='PayPal') {  **return** {error :**true**};  }  Transaction.wrap(**function** () {  cart.removeExistingPaymentInstruments('PayPal');  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_CREDIT\_CARD);  cart.removeExistingPaymentInstruments(dw.order.PaymentInstrument.METHOD\_BML);  cart.removeExistingPaymentInstruments('ALIPAY');  });  **var** paypalCancelUrl = dw.web.URLUtils.https('COBilling-Start');  **var** paypalReturnUrl = dw.web.URLUtils.https('Cybersource-ProcessPaypalExpress');  **var** Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');  **return** Cybersource.PaypalSetService({Basket:cart, paypalOrigin:'billing', paypalCancelUrl:paypalCancelUrl, paypalReturnUrl:paypalReturnUrl});  } |

##### Update “Authorize” function

To handle different PayPal responses.

|  |
| --- |
| function Authorize(args) {  var orderNo = args.OrderNo;  var order = OrderMgr.getOrder(orderNo);  var paymentInstrument = args.PaymentInstrument;  var paymentProcessor = PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();  Transaction.wrap(function () {  // paymentInstrument.paymentTransaction.transactionID = orderNo;  paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;  });  var Cybersource = require('int\_cybersource\_controllers/cartridge/controllers/Cybersource');  var pdict = Cybersource.PaypalOrderSetup({Order:order, PaymentInstrument: paymentInstrument});  if(pdict.EndNodeName == "OK"){  var paypalResult = Cybersource.AuthorizePaypal({Order:order, PaymentInstrument: paymentInstrument, orderSetupRequestId:pdict.orderSetupRequestId,  orderSetupRequestToken:pdict.orderSetupRequestToken,  orderSetupTransactionId:pdict.orderSetupTransactionId});  if (paypalResult.review){  var PlaceOrderError = new dw.system.Status(dw.system.Status.ERROR, "confirm.error.review");  return {error: true, PlaceOrderError : PlaceOrderError};  } else if (paypalResult.declined) {  var PlaceOrderError = new dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined");  return {declined: true, PlaceOrderError : PlaceOrderError};  } else {  return paypalResult;  }  }  return { error:true }  } |

#### 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 “saveCreditCard” function

Ensure savecreditcard logic excluded for Paypal

|  |
| --- |
| function saveCreditCard() {  var i, creditCards, GetCustomerPaymentInstrumentsResult, subscriptionID;  var cart = app.getModel('Cart').get();  if(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_BML) ||  app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals("PayPal") ||  app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals("ALIPAY") ||  !empty(cart.getPaymentInstruments("PayPal"))) {  return true;  }  if (customer.authenticated && app.getForm('billing').object.paymentMethods.creditCard.saveCard.value) { |

##### 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

Replace 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');  }  } |

### Remove Duplicate Payment Methods while Checkout

Customization has been done to remove duplicate payment method while making the payment. This functionality is not provided by Site Genesis by default in the storefront cartridge. So merchant need to make the changes in the existing cartridge to remove duplication of payment methods.

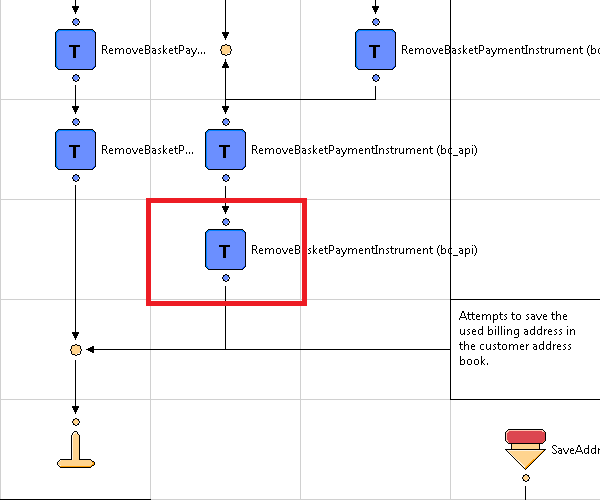
For Alipay and Paypal, code has been written in the Cybersource pipeline cartridge to remove the duplication. This issue was coming when user change the payment method from summary page by clicking Edit button in Payment Method section. So after changing the payment method user was getting both the methods on order summary page instead of overwritten method. To resolve this issue CybersourceData-RemovePaymentInstrument has been added in both CYBERSOURCE\_ALIPAY and PAYPAL\_EXPRESS at Handle node. This change is already there in Cybersource cartridge.

One more issue was there in the storefront, when user select credit card as payment method and change the payment method to Alipay on summary page the same issue was coming. For Paypal as Payment method, this issue was already handled in Storefront cartridge but issue was still there with Alipay as Payment Method. Add a RemoveBasketPaymentInstrument pipelet in COBilling-ResetPaymentForms pipeline [Refer Alipay authorization –pipeline flow].

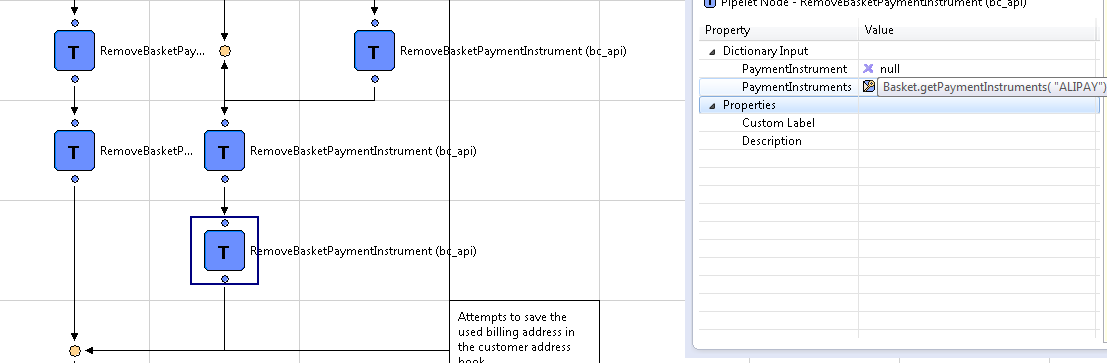
Similar changes are done in COBilling.js and CYBERSOURCE\_ALIPAY.js for Alipay Controller Cartridge [Refer Alipay Authorization-Controller flow]

CYBERSOURCE\_BML is also updated to remove duplicate payment instruments[Refer BML section]

BASIC\_CREDIT is also updated to remove duplicate Payment Instruments[ Refer CC section]

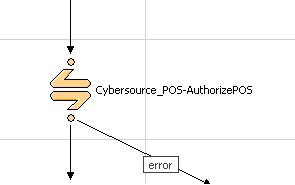


Set the Payment Instrument value to Basket.getPaymentInstruments( "ALIPAY")

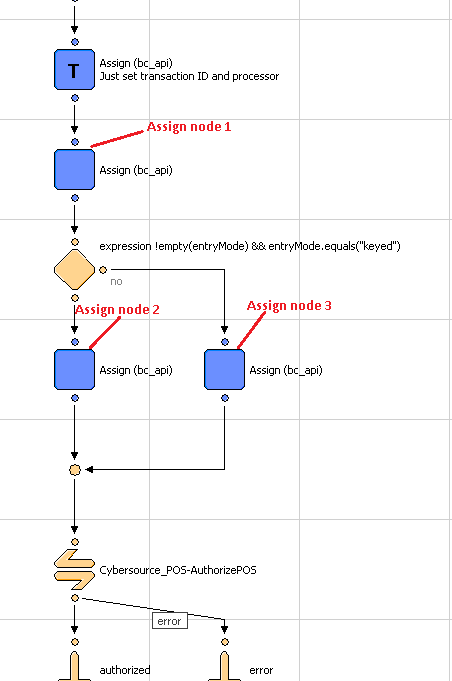


### Retail POS

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:



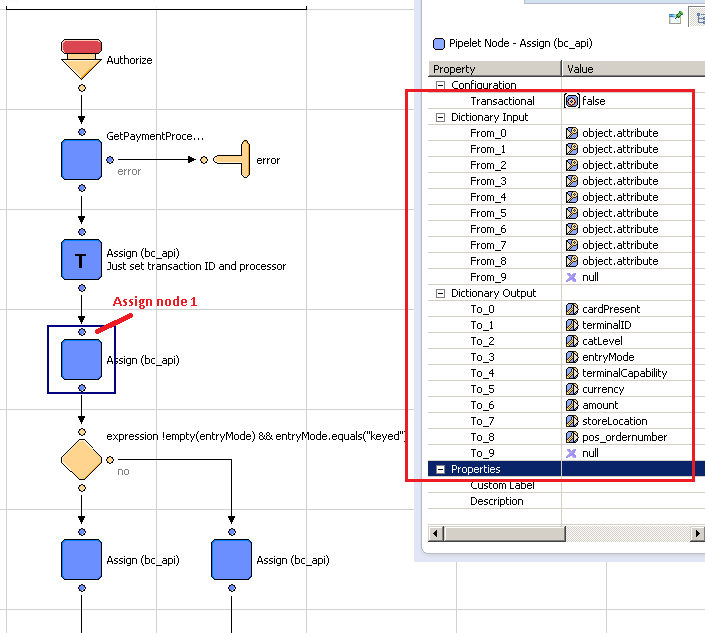
This required call node should be integrated at EACreditCard-Authorize pipeline of DSS app as show below. The track data, expiration date or account number should not be encrypted and may need to be decrypted prior to calling Cybersource\_POS-AuthorizePOS depending on the payment terminal used.



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.

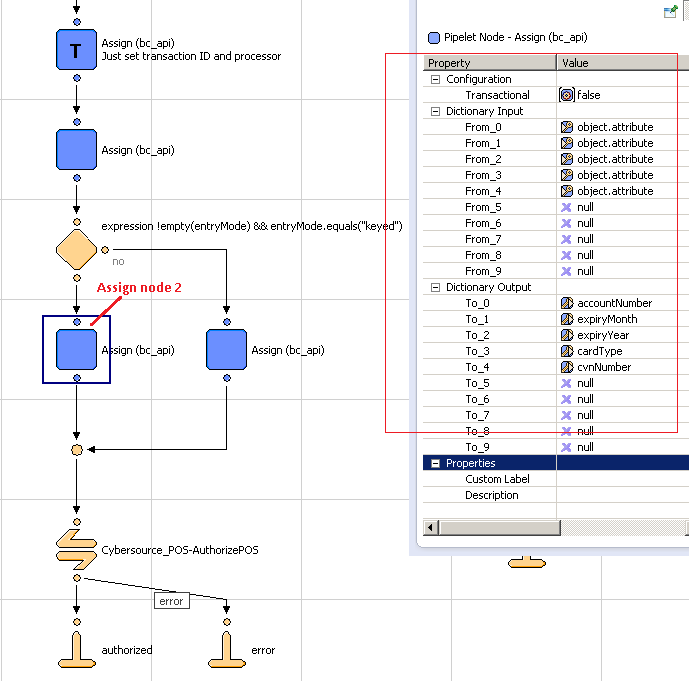
1. Use of Assign node labeled as “Assign node 1” above. This will set the common variables for the transaction irrespective of entry mode used. Below is the screenshot of variables used in this assign node:



Example input variables from DSS:



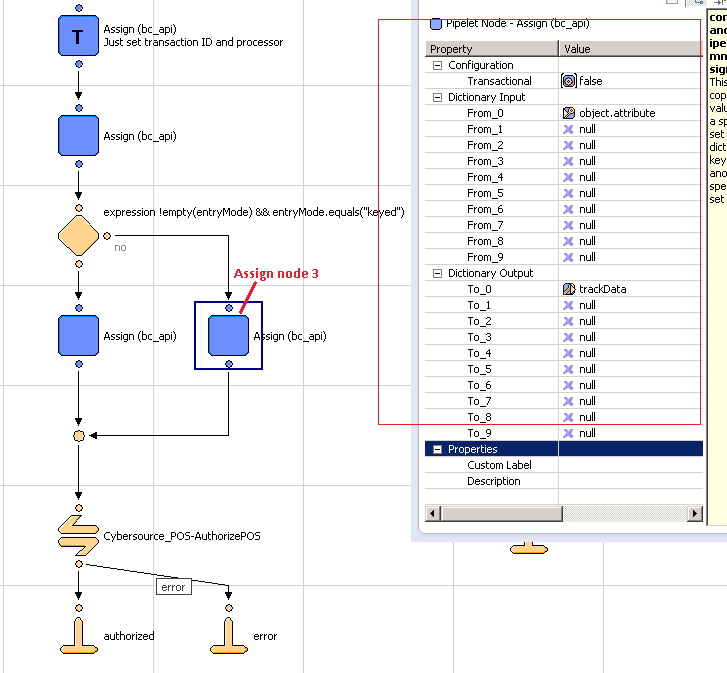
2. Use of Assign node labeled as “Assign node 2” if keyed entry mode is used on the POS terminal device. See below screenshot for the list of variables to set:



Example input variables from DSS:

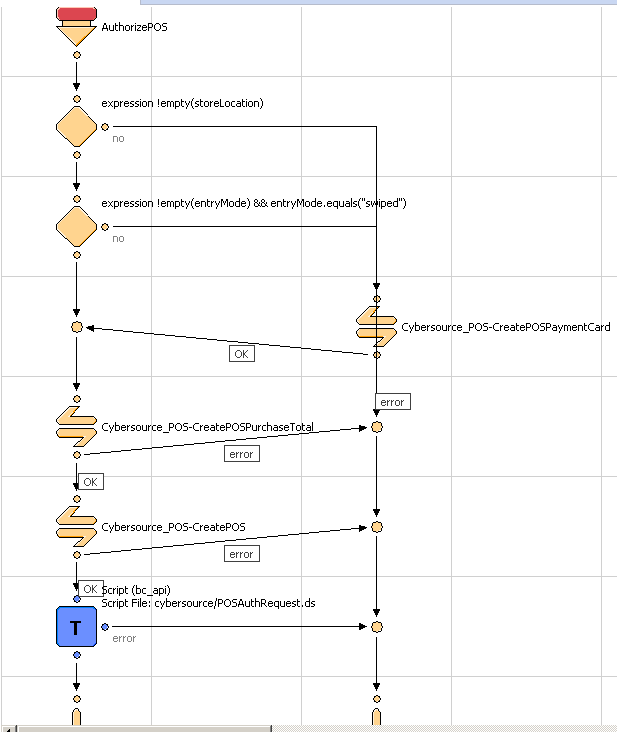


1. Use of Assign node labeled as “Assign node 3” if swiped entry mode is used on the POS terminal device. See below screenshot for the variable to set:



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.

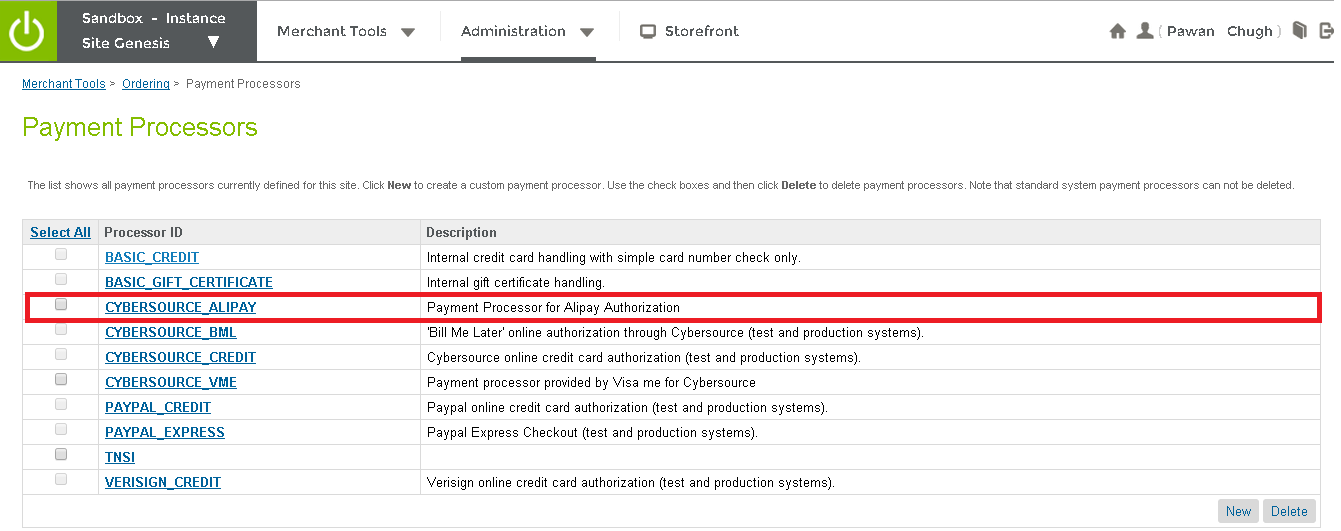
CREDITCARD\_UNCONFIRMED=400|InvalidCreditCardException|Unconfirmed|Unconfirmed

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

## Site Configuration

### Configure Payment Processor for Alipay

Go to Site -> Ordering -> Payment Processors, add a new payment processor name CYBERSOURCE\_ALIPAY to handle Alipay related requests while order processing through Alipay as payment method. Please refer to the screen shot below:



### 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 site preferences
* /int\_cybersource/configuration/ Cybersource\_PaymementTransaction\_updates.xml – add custom attributes to the Payment transaction object.
* /int\_cybersource/configuration/CyberSource\_POS\_CustomObjectDefinitions.xml – add 2 custom objects for mapping of device’s serial number to Terminal ID and set of Merchant ID and key pair(s).

### Import Payment Methods and Cards

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

* /int\_cybersource/configuration/Cybersource\_Alipay\_PayPal\_Payment\_Method.xml- add new payment method as Alipay and Paypal
* /int\_cybersource/configuration/Cybersource\_Cards\_BML\_Payment\_Method.xml- updates payment method of credit card and bml, and update the allowed cards information. The allowed card types are Visa, Amex, MasterCard, Discover, and Maestro. Merchant if interested in other card types then customization required in codebase to handle cardType name vs numeric value mapping.

### Configure Services

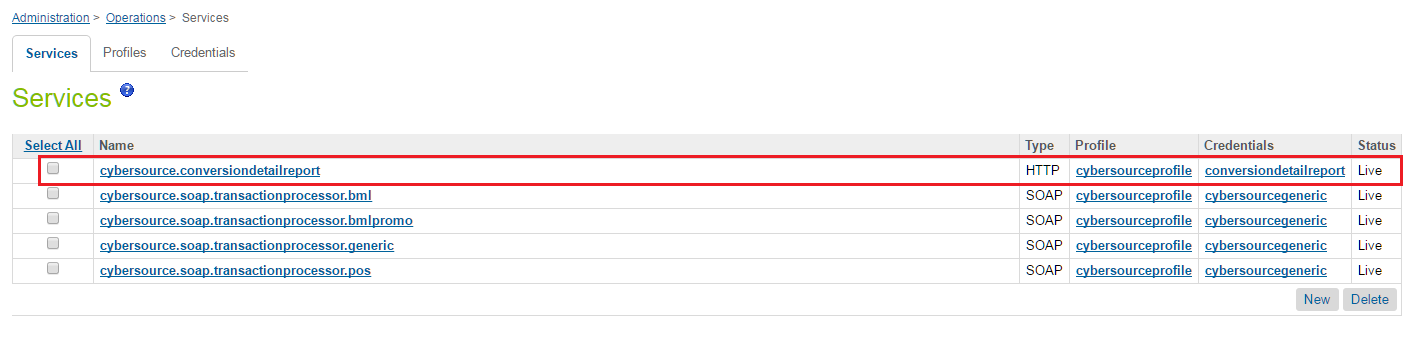
To import the following Service configuration Go to Administration > Operations > Import & Export-> upload the below mentioned file and import the configuration under services

* /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.bml
  2. Cybersource.soap.transactionprocessor.bmlpromo
  3. Cybersource.soap.transactionprocessor.generic
  4. Cybersource.soap.transactionprocessor.pos
  5. Cybersource.conversiondetailreport

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

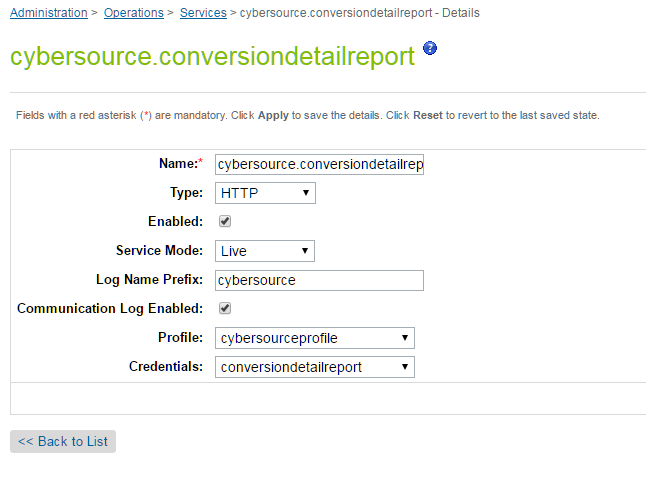
Similarly, merchant can create or update existing credential settings for each service stated above.

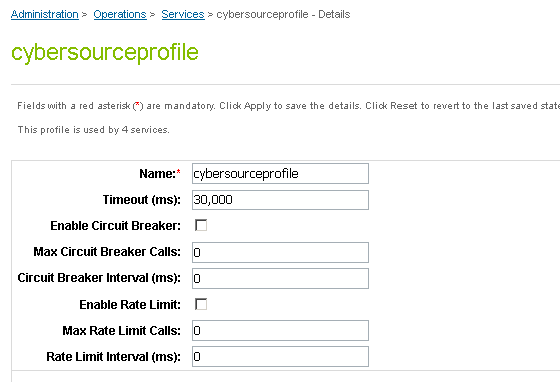
There is Cyber Source detailed report service created in DemandWare with below separate Credentials as follows:

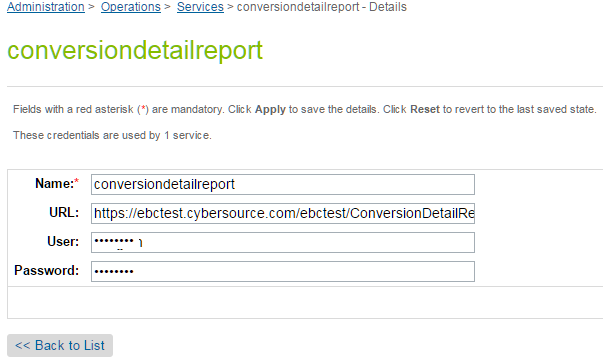
1. URL: Specify below report location along with the requested parameter ,the parameter values are replaced at runtime by the JOB code
   * Test environment URL is "https://ebctest.cybersource.com/ebctest/ConversionDetailReportRequest.do?merchantID={merchantID}&userName={userName}&password={password}&startDate={startDate}&startTime={startTime}&endDate={endDate}&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:



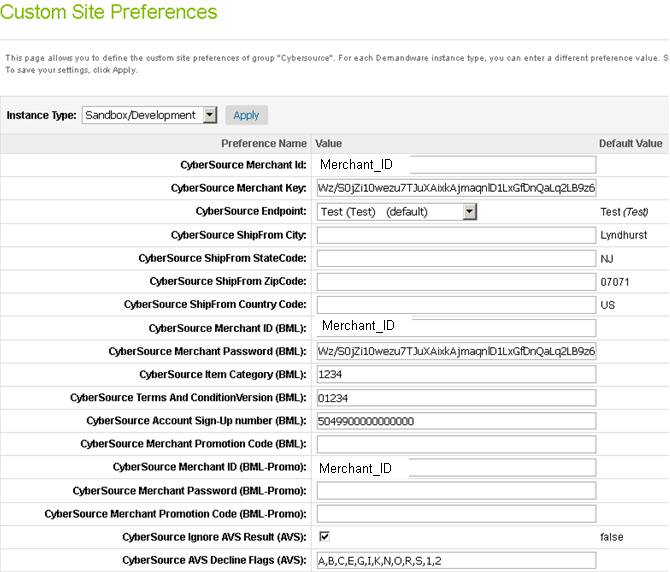


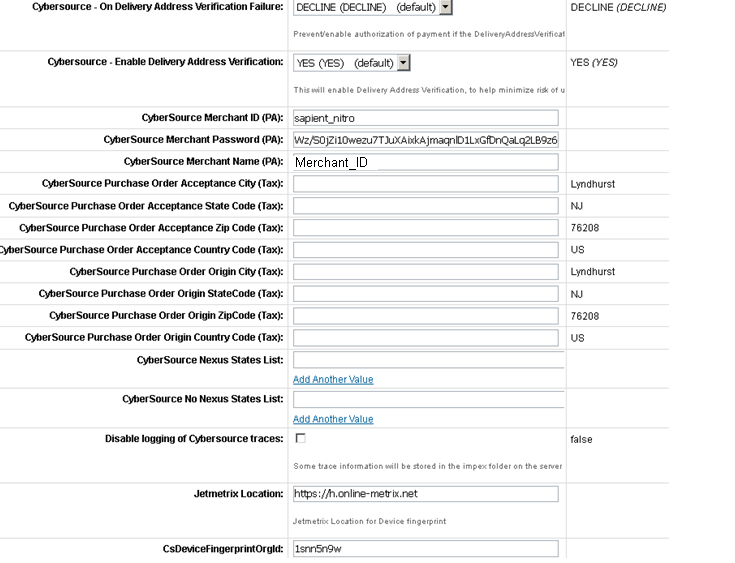


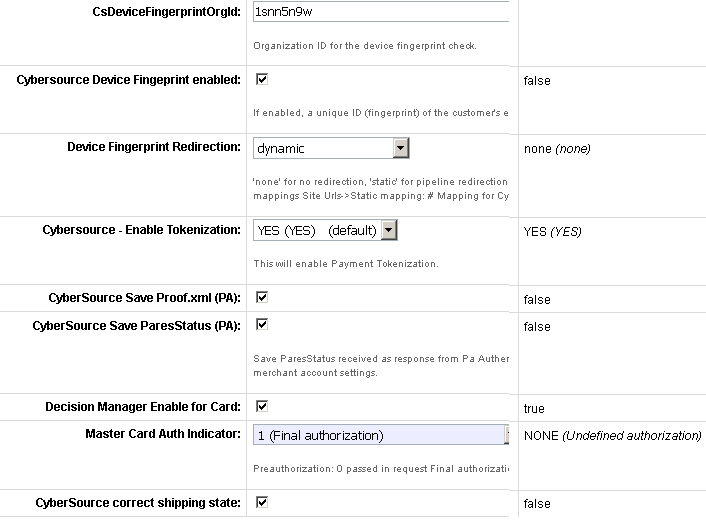
### Configure Site Preferences

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

The screen shot below depicts the site preferences configuration:







### 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

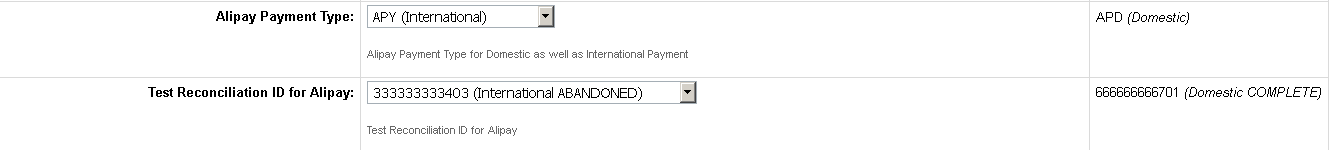
Update shipping method preference through Business Manager >StoreFront Site> Ordering> Shipping Methods > Name > CyberSource Shipping ID

The screen shot below depicts the site preferences configuration:



### Configure Site Preferences for Alipay

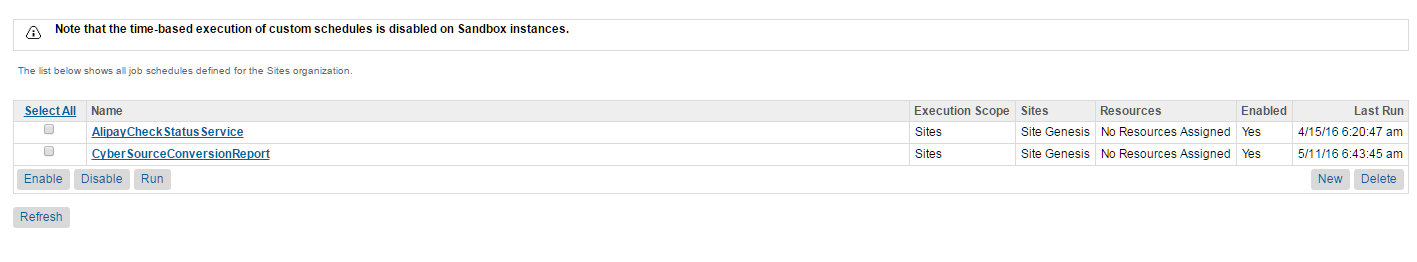
Verify Alipay Site Preferences in already existing custom preferences group “Cybersource”.



### 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



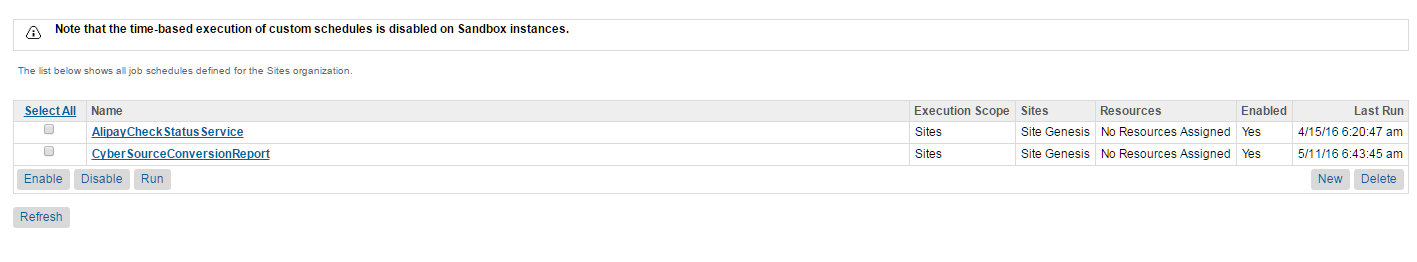
### Configure Batch Job for Conversion Detail Report

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.

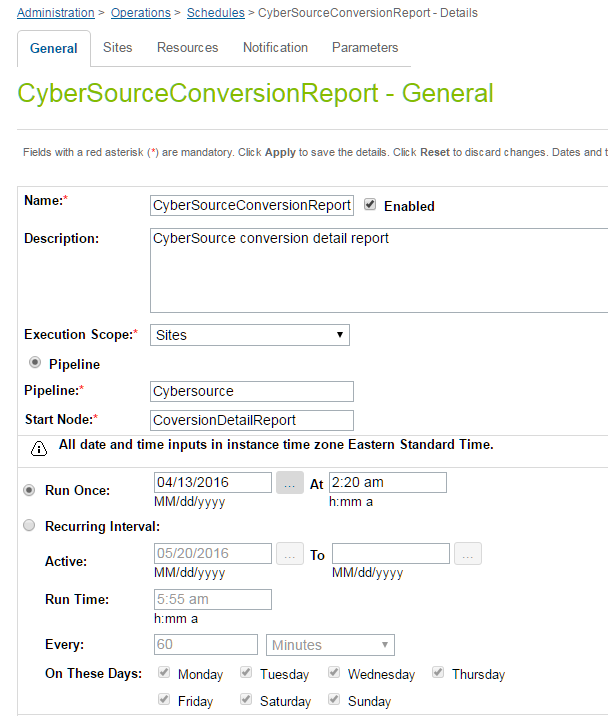
Verify the newly added batch jobs for Conversion detail report service.

Go to Administration - > Operations -> Job Schedules



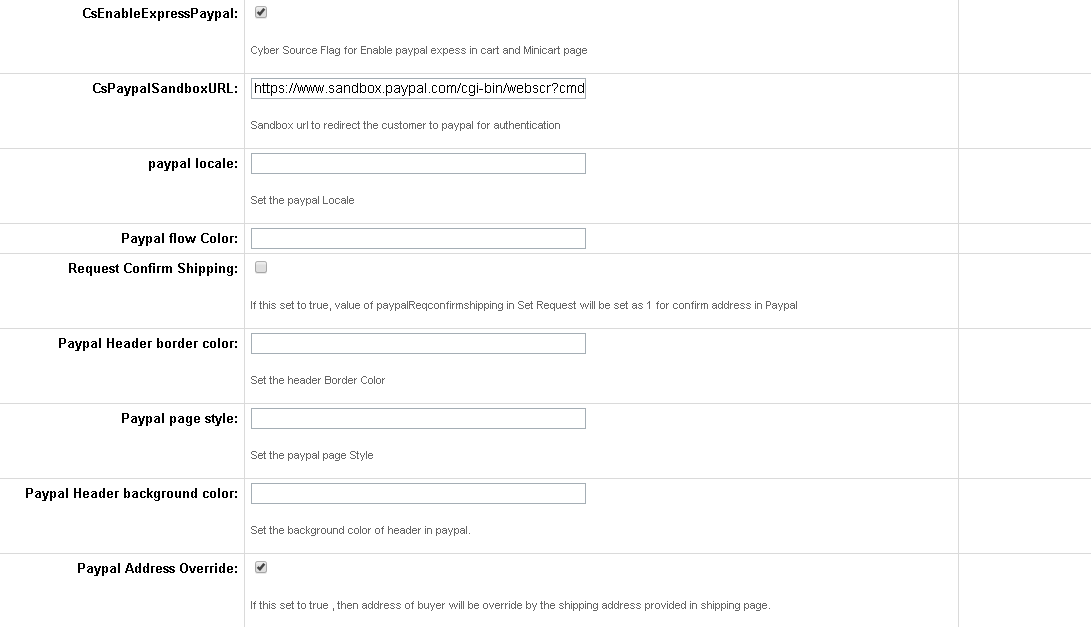
The batch job created for cybersource conversion detail report specified below, it updates the status of order in demandware which are in CREATED state and mark them as “CANCELLED” for rejected order or “NEW” for accepted order. The accepted orders are marked for “READY FOR EXPORT” as well.

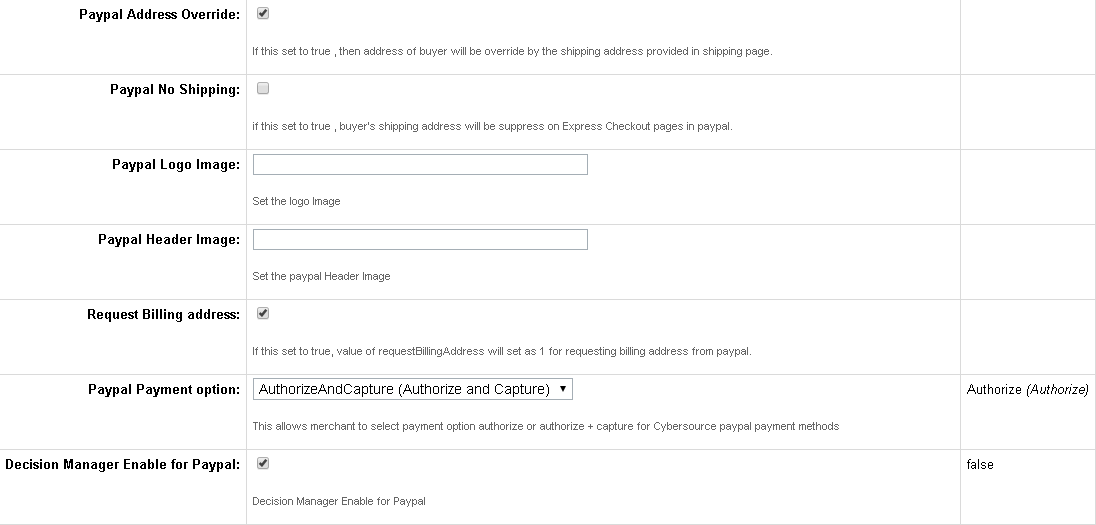
* Batch Job Name: **CyberSourceConversionReport**
* Pipeline Name: CyberSource
* Start Node Name: CoversionDetailReport



### Configure Site Preferences for PayPal and PayPal Express Checkout

Verify Cybersource\_paypal as newly added Custom Preferences Group



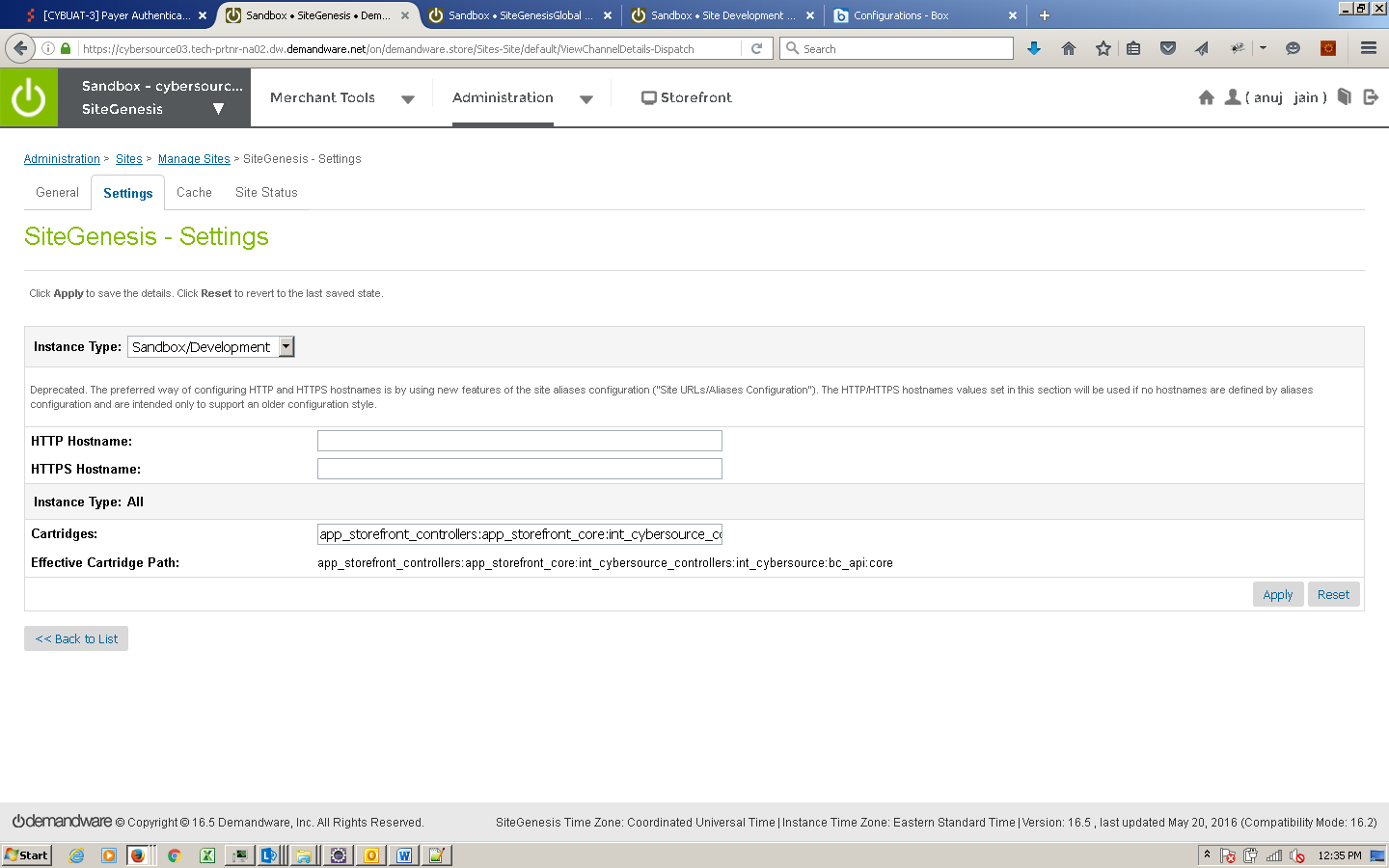


### Applying CyberSource Cartridge to the Site

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:



### 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





## 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”

## Cartridges Structure and Reference

### Pipelines & Controllers

The controllers has functions which are acting as wrapper to call corresponding pipeline

##### Cybersource

This contains the nodes that make the actual call to CyberSource based on what is needed, cc auth, bml, tax or address verification.

##### CybersourceData (Only Pipeline)

This contains the node that creates the objects needed for the CyberSource pipeline. Each node in the pipeline creates a needed object for each request. This is the pipeline that will be modified by customers, as each customer may need to pass different information based on their needs

##### CybersourceUnitTesting

Used for testing.

##### Cybersource\_Subscription

This contains the nodes that makes call to different Subscription related services. This is the pipeline that will be referred and modified by the merchants, as each merchant may need to pass different information based on their needs.

##### Cybersource\_POS

This contains the main node to be integrated and related sub node for integration of retail POS transaction. This pipeline also contains other node for testing purpose.

##### Cybersource\_Services

This contains the main node to be integrated for PayPal express and PayPal authorization. This also contains node for full authorization reversal.

### Scripts

There is a JS object for each request data elements. List of DS objects used:

* Cybersource\_BillTo\_Object.ds
* Cybersource\_BML\_Object.ds
* Cybersource\_Card\_Object.ds
* Cybersource\_Item\_Object.ds
* Cybersource\_POS\_Object.ds
* Cybersource\_PurchaseTotals\_Object.ds
* Cybersource\_ShipFrom\_Object.ds
* Cybersource\_ShipTo\_Object.ds
* Cybersource\_TaxRequest\_Object.ds
* CreateCyberSourceBillToObject\_UserData.ds
* CreateCyberSourcePaymentCardObject\_UserData.ds
* CreateCyberSourcePurchaseTotalsObject\_UserData.ds

The following scripts are used in CyberSource pipeline to make the CyberSource web service calls.

* TaxationRequest.ds
* BMLAuthRequest.ds
* CCAuthRequest.ds
* DAVRequest.ds
* PayerAuthEnrollCheck.ds
* PayerAuthValidation.ds
* AlipayInitiatePaymentRequest.ds
* AlipayCheckPaymentStatusRequest.ds
* PaypalSetEcSetRequest.ds
* PaypalGetEcRequest.ds
* PaypalOrderSetupEcRequest.ds
* PaypalAuthorizationRequest.ds
* PaypalCaptureRequest.ds

The following scripts are used in CyberSource to make the CyberSource Test web service calls.

* CCCaptureRequest.ds
* TestBML.ds
* TestCCAuth.ds
* TestTax.ds
* TestDAVRequest.ds
* TestPayerAuthEnrollCheck.ds
* TestPayerAuthValidation.ds
* TestAlipayCheckStatusRequest.ds
* TestAlipayInitiateRequest.ds
* PaypalCaptureRequest.ds

The following scripts are used in Cybersource\_Subscription pipeline to make the Payment Tokenization web service calls.

* CreateSubscription.ds
* DeleteSubscription.ds
* OnDemandSubscription.ds
* UpdateSubscription.ds
* ViewSubscription.ds

The following scripts are used to create the objects needed. These are the scripts that customers would change to fit their needs

* CreateCybersourceBillToObject.ds
* CreateCybersourceBMLObject.ds
* CreateCybersourceBMLPaymentCardObject.ds
* CreateCybersourcePaymentCardObject.ds
* CreateCybersourcePurchaseTotalsObject.ds
* CreateCybersourceShipFromObject.ds
* CreateCybersourceShipToObject.ds
* CreateCybersourceTaxationItemsObject.ds
* CreateCybersourceTaxationPurchaseTotalsObject.ds
* CreateCyberSourceTaxRequestObject.ds
* AlipaySetProductParameters.ds
* CreateCSPurchaseTotalForAlipay.ds
* CreateCybersourceBillToObjectForExpressCheckout.ds
* CreateCybersourceItemForPaypal.ds
* AddAddressToBasket.ds
* AlipayGetOrderForCheckStatusWorkflow.ds

These are the pipelines that use the scripts from above and may have need storefront specific integration:

* CybersourceData:CreateBillTo
* CybersourceData:CreateShipTo
* CybersourceData:CreatePaymentCard
* CybersourceData:CreatePurchaseTotals
* CybersourceData: CreateShipFrom
* CybersourceData: CreateTaxItems
* CybersourceData: CreateTaxService

The following is a library script that is used by the Request scripts to build the XML request that is passed to CyberSource. This lib script contains integration independent code and it doesn’t need any storefront specific changes.

* libCybersource.ds

### Templates

* payerauthentication.isml
* payerauthenticationredirect.isml
* pt\_payerauthentication.isml
* alipayintermediate.isml
* fingerprint.isml
* fingerprintredirect.isml
* Contains templates used by unit test pipeline

### Configuration Files

Contains below configuration files as follows:

* Cybersource-Services.xml
* Cybersource\_subscription\_metadata.xml
* CyberSource\_POS\_CustomObjectDefinitions.xml
* Cybersource\_Alipay\_Paypal\_Payment\_Method.xml
* Cybersource\_Cards\_BML\_Payment\_Method.xml
* Cybersource\_Alipay\_CheckStatusService\_BatchJob.xml
* CyberSource\_Conversion\_Report\_BatchJob.xml
* Cybersource\_PaymentTransaction\_update.xml
* Cybersource-metadata.xml

### Retail POS Cartridge Components

|  |  |
| --- | --- |
| **Component** | **Files** |
| **Pipelines** | *Cybersource\_POS.xml*  *CybersourceUnitTesting.xml* |
| **Forms** | *pos.xml* |
| **Scripts** | *cybersource/Cybersource\_POS\_Object.ds*  *cybersource /libCybersource.ds*  *cybersource/POSAuthRequest.ds*  *POS/CreateCyberSourcePaymentCardObject\_UserData.ds*  *POS/CreateCyberSourcePOSObject\_UserData.ds*  *POS/CreateCyberSourcePurchaseTotalsObject\_UserData.ds*  *Cybersource\_UnitTesting/TestPOSAuth.ds* |
| **Templates** | *custom/pos\_scripterror.isml*  *pos/createpos.isml*  *pos/postransactionresult.isml* |
| **Metadata** | *CyberSource\_POS\_CustomObjectDefinitions.xml* |

# 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 |
| Bill Me Later (BML) | **0.5**– Person Day | * Setup Account with Bill Me Later. |
| 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 |

\*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-TestBMLAuth

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

# CyberSource Site Preferences

### Site preference and description

|  |  |
| --- | --- |
| 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 Merchant ID(CsBmlMerchantId) | BML Merchant ID |
| CyberSource Merchant Password(CsBmlPassword | BML Merchant Key |
| CyberSource Item Category(CsBmlItemCategory) | BML Item Category |
| CyberSource Terms AndConditionVersionCsBmlTCVersion | BML Terms and Condition Version |
| CyberSource Account Sign-Up numberCsBmlNewAcctNo) | BML CyberSource account sign-up number |
| CyberSource Merchant Promotion Code(CsBmlPromoCode) | BML promo code |
| CyberSource Merchant ID(CsBmlPromoMerchantId) | BML promo merchant ID |
| CyberSource Merchant Password(CsBmlPromoMerchantPassword) | BML promo merchant key |
| CyberSource Merchant Promotion Code(CsBmlPromoPromoCode) | BML promo promo code |
| 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 |

# Cybersource\_paypal Site Preferences

### Site preference and description

|  |  |  |
| --- | --- | --- |
| **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 |

# 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.

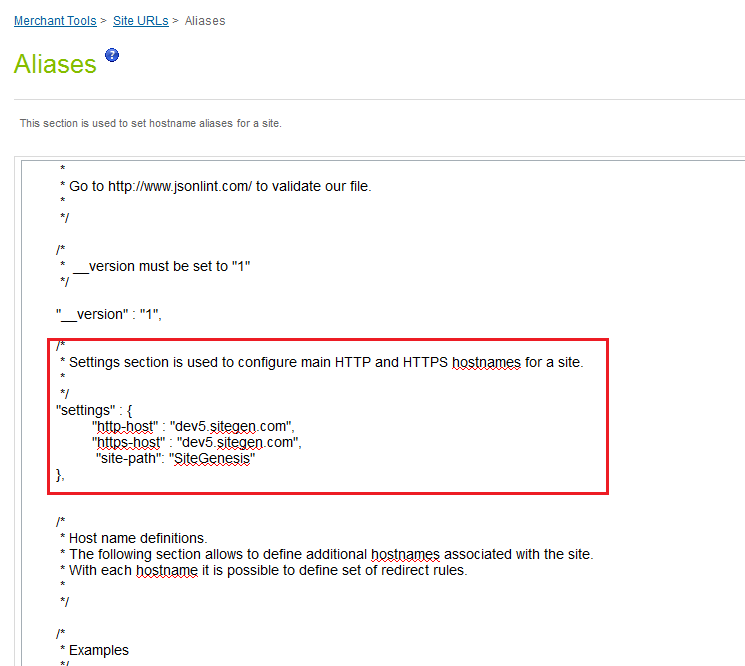
All URLs matching the pattern will be redirected by the Demandware Server.



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:



### Modified Scripts and pipelines for the device fingerprint

* libCybersource .ds->addCCAuthRequestInfo modified.
* CCAuthRequest.ds (updated WSDL reference)

# 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/CC_Svcs_IG_BML_Supplement/html/>
9. <http://apps.cybersource.com/library/documentation/dev_guides/Verification_Svcs_IG/20091012_Verification_IG.pdf>
10. <http://www.cybersource.com/support_center/support_documentation/services_documentation/tax.php>
11. <http://apps.cybersource.com/library/documentation/dev_guides/Tax_IG/Tax_Guide.pdf>
12. <http://apps.cybersource.com/library/documentation/dev_guides/Retail_SO_API/Retail_SO_API.pdf>
13. <http://apps.cybersource.com/library/documentation/dev_guides/PayPal_Express_SO/PayPal_Express_SO_API.pdf>
14. <http://apps.cybersource.com/library/documentation/dev_guides/AliPayDom/AliPay_Dom_SO_API.pdf>
15. <http://apps.cybersource.com/library/documentation/dev_guides/AliPayInt/AliPay_Int_SO_API.pdf>

# 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 |