CyberSource LINK Cartridge

Version 21.2.0

Aug 2021

Contents

Summary

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

- **Credit Card Authorization**
- **CyberSource Address Verification**
- **Delivery Address Verification**
- **Payment Tokenization**
- **Payer Authentication**
- **Tax Service**
- **Credit Card Secure Acceptance Web /Mobile**
    - **Redirect method**
    - **Iframe method**
- **Credit Card Secure Acceptance Silent Order Post**
- **Visa Checkout**
- **AliPay**
    - **Domestic**
    - **International**
- **Retail POS**
- **Klarna**
- **WeChat Pay**
- **Bank Transfer APM's**
    - **SOFORT**
    - **BANCONTACT**
    - **GIROPAY**
    - **EPS**
    - **IDEAL**
- **Apple Pay REST Based Interface**
    - **To Process Authorization for Encrypted Payload**
    - **To Process Authorization for Cryptogram**
- **Android Pay REST Based Interface**
    - **To Process Authorization for Encrypted Payload**
    - **To Process Authorization for Cryptogram**
- **PayPal Express Checkout**
    - **PayPal Standard/Custom Order**
    - **PayPal Billing Agreements**
    - **PayPal Credit Order**

- **Batch Jobs**
    - **Alternate Payment Check Status Batch Job**
    - **Secure Acceptance Merchant Post Notification Processing Batch Job**
    - **Conversion Details Report Batch Job**
    - **IDEAL Bank Option Batch Job**

Component Overview

Functional Overview

Credit Card Authorization Service

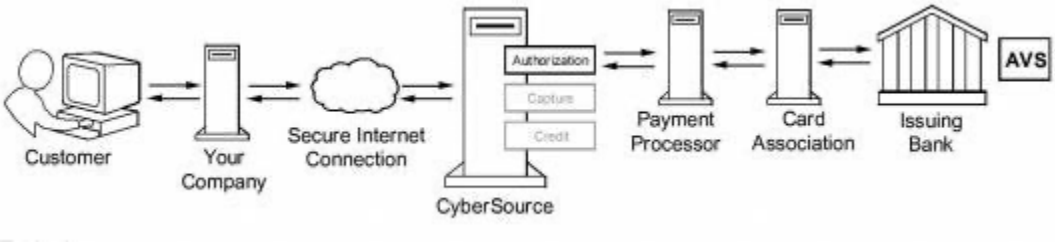
The credit card authorization service controller allows storefront application to request for credit authorization for the total order amount. The controller makes the credit card authorization web service call to CyberSource authorization service and receive confirmation about the availability of the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

The Demandware Cybersource–AuthorizeCreditCard 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:
    a. Capture pertinent DAV result information & DAV Reason Code
    b. 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:
    a. Exit immediately with declined or review response, as merchant defines
9. Capture pertinent AVS information
10. Validate authorization reason code and set corresponding values, 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.

CyberSource 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.

Merchant Defined Data (MDD) changes

CyberSource cartridge enables merchant to send additional information in authorization service using MDD fields. This information can be used in OMS. Cartridge does not support to send MDD fields into request, however merchant can customize the Autherise request to pass these additional fields.

CyberSource 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)
- Method returns either: authorized, declined or error (authorized==success, declined==failure)
- Captured validation information is extracted from arguments to present user with choices to correct problems, confirm "standardized" formatting or try again
- If service is successful, allow Shipping Address save operation to continue

Decision Manager

- **The CyberSource Decision Manager provides Merchant and ability to set business rules, provide case management, and Reporting.**
- **The CyberSource Decision Manager Business rule engine allows Merchant to analyze the order data based on predefined or custom rules. The business rules can be set by orders, by category, or by SKU.**
- **The Demandware CyberSource Cartridge is using an alternate "Conversion Detail Report" Job for transaction status updates**

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. 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 Cyber Source'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

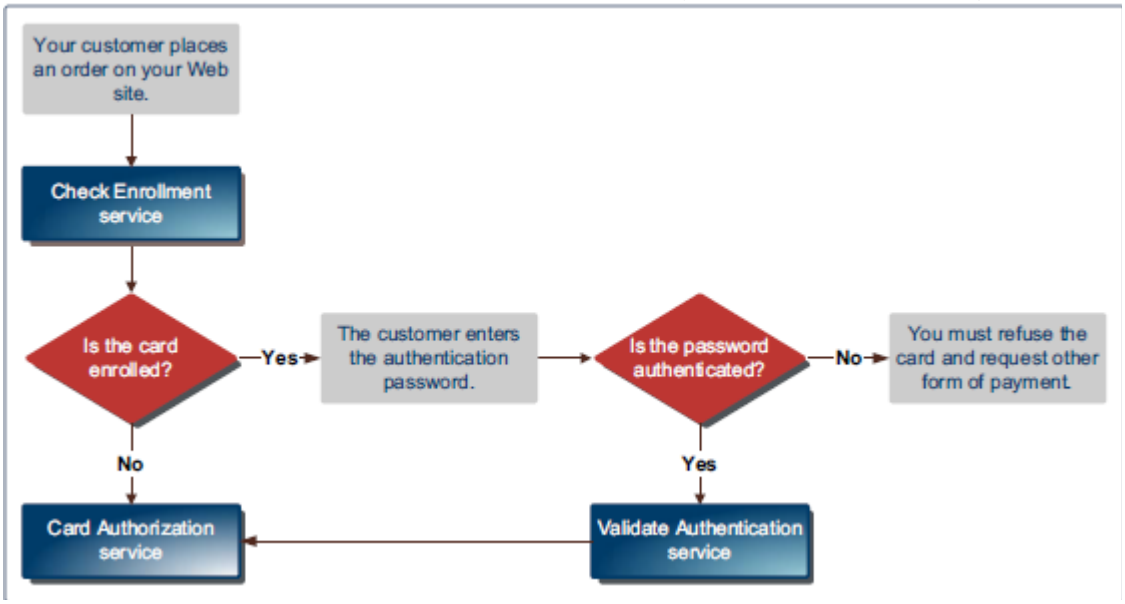
CyberSource Payer Authentication services enable you to add support to your web store for card authentication services, including Visa Verified by VisaSM, MasterCard® and Maestro® SecureCode™ (UK Domestic and international), American Express SafeKeySM, and JCB J/Secure™.

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.

Tax Service

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.

Secure Acceptance Authorization

Secure Acceptance payment gateway is used to process transaction requests directly from the customer browser so that sensitive payment data does not pass through Demandware servers. Secure Acceptance feature is implemented using these secure acceptance payment methods:

1. **Secure Acceptance – Redirect**
2. **Secure Acceptance – Iframe**
3. **Secure Acceptance – Silent Post**
4. **Secure Acceptance – Flex**

All the above secure Acceptance methods provide a common feature of handling the secure information on secure pages only.
Secure Acceptance Redirect: Customer will get redirect to secure Acceptance payment gateway when clicking on Place Order from Review Page
Secure Acceptance Iframe: Customer will get redirect to secure Acceptance payment gateway within an Iframe embedded in a new summary page added into checkout flow
Secure Acceptance Silent Order Post: Credit Card form data is posted to secure acceptance silent post URL and token is generated and user is redirected on review page and normal card authorization flow is being used to further process the transaction.
Secure Acceptance Flex microform: Credit Card number and cvv filed will replace with iframe and will get tansit token form flex microfom which will use for futher transction.
Secure Acceptance Web/Mobile Authorization Sequence flow:

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

Secure Acceptance Silent Order Post Authorization Sequence flow:

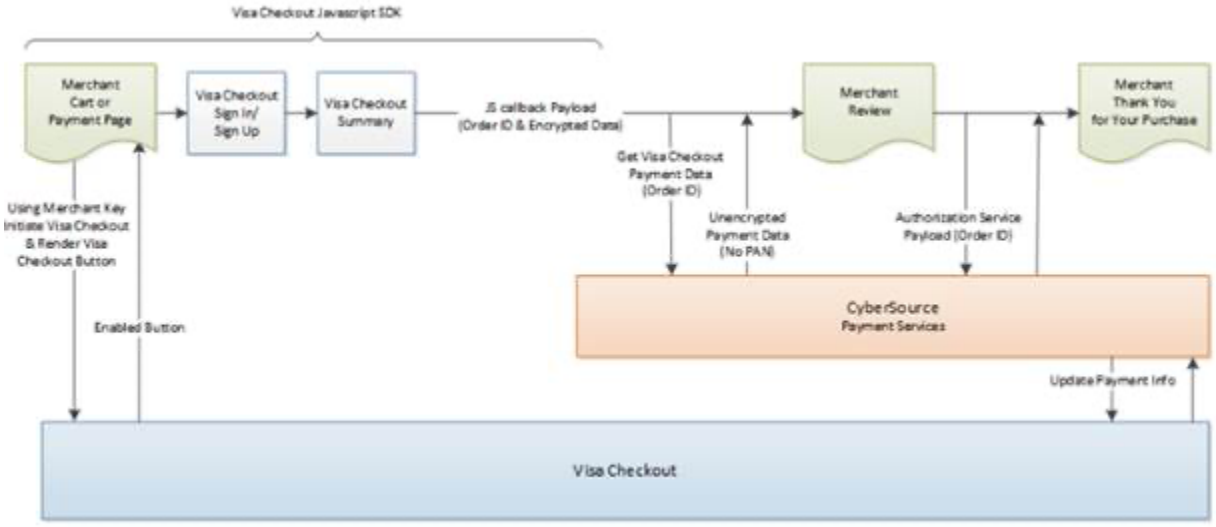
1. **An Ajax function is created to call Secure Acceptance silent post controller to prepare request data except card details**
2. **Card details are populated within Ajax to prevent security breach , further the details are posted to selected APM form action URL**
3. **Silent post will create or update token based on request details and return the response on Demandware custom controller method which parse the response of CurrentHttpParameterMap and return back to corresponding pages[summary/billing/cart]**

On Place Order, Secure Acceptance authorization is called which internally completed the flow using CyberSource Authorization [refer Credit Card Authorization service]

Visa Checkout

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

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



1. Your web site integrates directly to Visa Checkout to display the Visa Checkout button on your checkout page.
2. CyberSource provides the get Visa Checkout data service, which retrieves the Visa Checkout payment data, except the PAN. You can use the retrieved data to help the customer confirm the purchase.
3. You submit an authorization request to CyberSource for credit card processing. Instead of including payment information in the authorization request, you include the Visa Checkout order ID.
4. At various points in the transaction cycle, you notify the customer of the transaction status.

AliPay Authorization

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

The Demandware CyberSource- AuthorizeAlipay 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 variables.
5. If initiation is successful, then assign the required values in Demandware Payment Transaction object and 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 variables
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

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.

Klarna

The Klarna authorization service controller allows storefront application to request for credit authorization for the total order amount. The controller initially makes the call to CyberSource Init Session service to initialize the Klarna widget and Klarna JS API authorization call along with authorization web service call to CyberSource authorization service and receive confirmation about the availability of the funds.

The Demandware KLARNA_CREDIT–Authorize populates the authorization request with ship-to, bill-to, Klarna Item data, and purchase total data from the basket and invokes the authorization web service call using CyberSource web service API.

Klarna sequence flow:

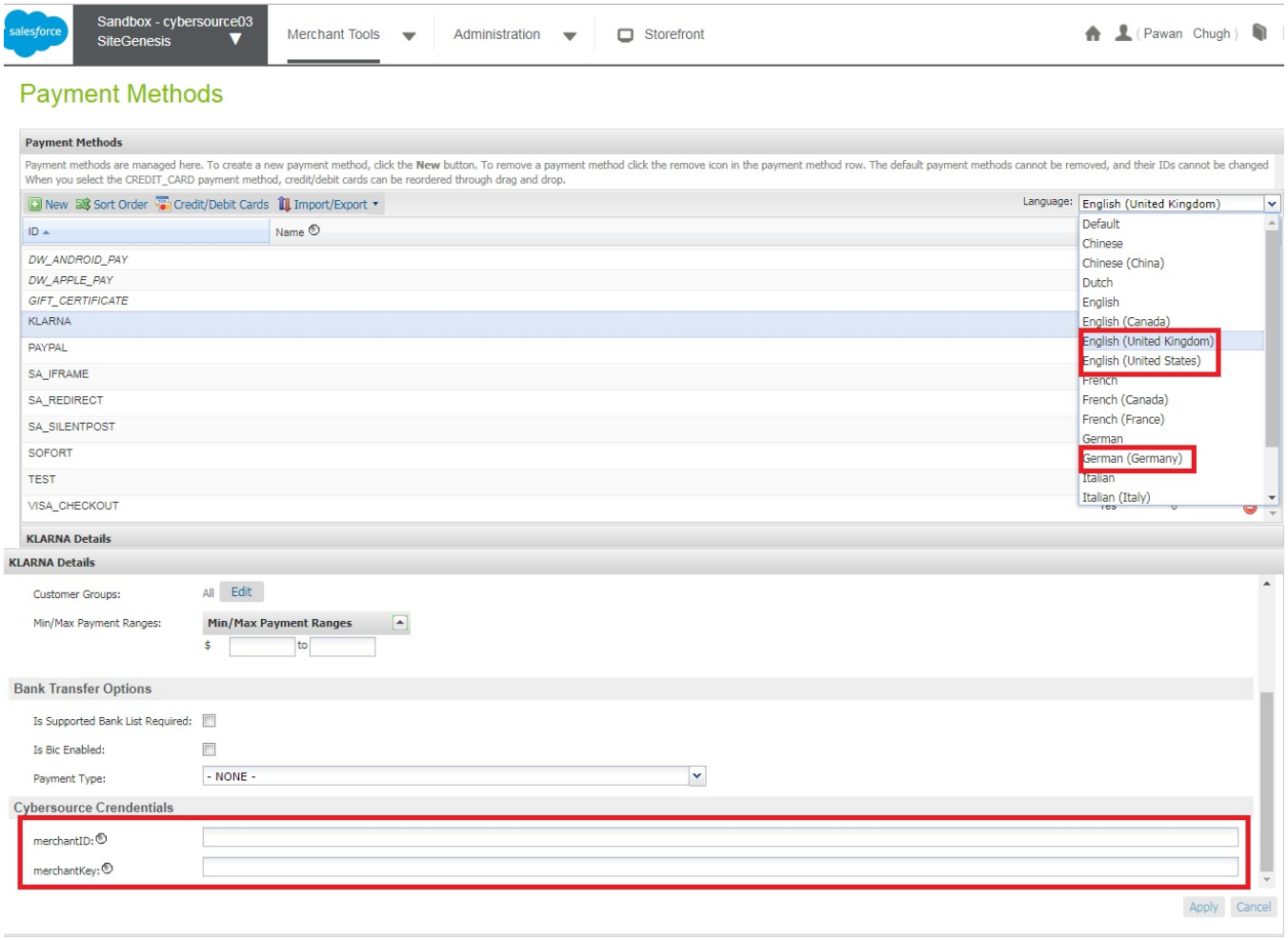
1. Creates CyberSource Init session request using ship-to, bill-to, item data, and purchase total data from the current basket
2. Make actual service call to CyberSource Init session service
3. If service returns ACCEPT as decision and 100 as reason code, get the processor token from session service response and set its value into a session variable
4. If service returns any other decision apart from ACCEPT and 100 as reason code, display an error message on billing page
5. Pass the value of processor token Klarna JS API to load the Klarna widget on summary page
6. Create CyberSource authorization request using ship-to, bill-to, item data, and purchase total data from the current basket
7. If Decision Manager is configured in site preference, pass its value to true else false in CyberSource authorization call
8. Click Pay button to first authorize the request through Klarna JS API and then pass the pre-approved token returned by JS API authorization request in CyberSource authorization request
9. If authorization service returns 'ACCEPT' as decision, 100 as reason code and 'authorized' or 'pending' as payment status and If merchant URL redirection is configured in site preference, redirect the user to merchant URL and return back to merchant site to complete the order
10. If authorization service returns 'ACCEPT' as decision and 100 as reason code, 'authorized' as payment status and merchant URL redirection is false, complete the order and modify order and export status
11. If authorization service returns 'ACCEPT' as decision and 100 as reason code, 'pending' as payment status and merchant URL redirection is false, CyberSource check status service would be called to complete the transaction
12. If authorization service returns 'ACCEPT' as decision, 100 as reason code and 'failed' as payment status, exit immediately and change the status of order to failed
13. If authorization service returns 'REJECT' or 'ERROR' as decision, exit immediately and change the status of order to failed
14. If authorization service returns 'REVIEW' as decision, complete the order transaction but order status would be created itself
15. If payment status is 'pending', CyberSource check status service call would be made for both merchant URL redirected orders and non-redirected orders
16. If check status service returns 'ACCEPT' as decision, 100 as reason code and 'authorized' or 'settled' as payment status, complete the order and modify order and export status
17. If check status service returns 'ACCEPT' as decision, 100 as reason code and 'pending' as payment status, complete the order without modifying order and export status
18. If check status service returns 'ACCEPT' as decision, 100 as reason code and 'abandoned' or 'failed' as payment status, exit immediately and change the status of order to failed
19. If check status service returns 'REJECT' or 'ERROR' as decision, exit immediately and change the status of order to failed
20. If check status service returns 'REVIEW' as decision, complete the order transaction bur order status would be created itself

Validate authorization reason code and set corresponding values, based on Auth response code.

Merchant Id/Key Specific Changes for Klarna

Different countries and specific currencies could be configured to run Klarna with different Merchant Id/Key specific to different sites. Functional flows would be similar on different sites. Merchant Id/Key could be configured at Merchant Tools -> Ordering -> Payment Methods -> Klarna. In this release, Klarna has been supported for US, UK and Germany with different sites and corresponding Merchant Ids/Key. To update the value of merchant Id/Key specific to the sites, follow below mentioned steps.

- **Change the language to either English(United States), English(United Kingdom) or German(Germany)**
- **Select Klarna as payment method and enter merchantID and merchantKey field in CyberSource Credentials section of payment method**
- **Select the appropriate language setting in the Bill-To Language field under the 'Klarna' section.**



Bank Transfer APM's

The Bank Transfer service controller allows storefront application to request to sale total order amount. The controller makes the call to CyberSource sale service to authorize the purchase amount and in return a call has been made to check status service to complete the functional flow.

The Demandware BANK_TRANSFER–Authorize populates the sale request with bill-to, Item data, purchase total data and merchant descriptor data from order and invokes the sale web service call using CyberSource web service API.

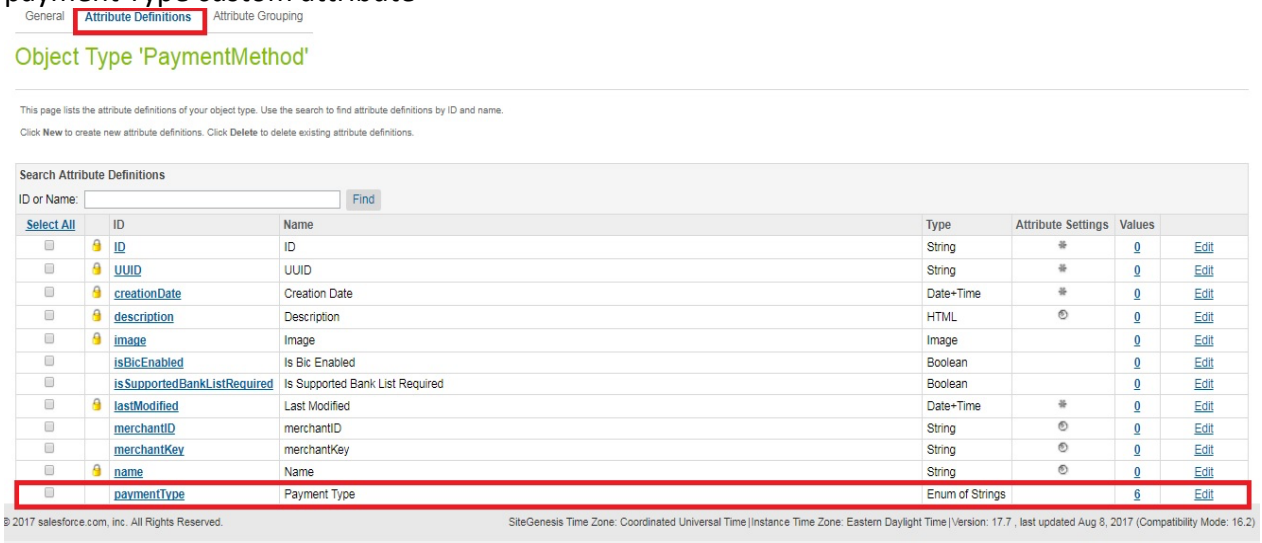
Note: For Bank Transfer, same processor name BANK_TRANSFER has been used for different APMs. If merchant want to add a new APM which consist of CyberSource sale and check status service, new APM could be added choosing BANK_TRANSFER as processor while creating new payment method in payment setting. Following APMs have been addressed as a part of this release.

- SOFORT
- BANCONTACT
- IDEAL
- EPS
- GIROPAY

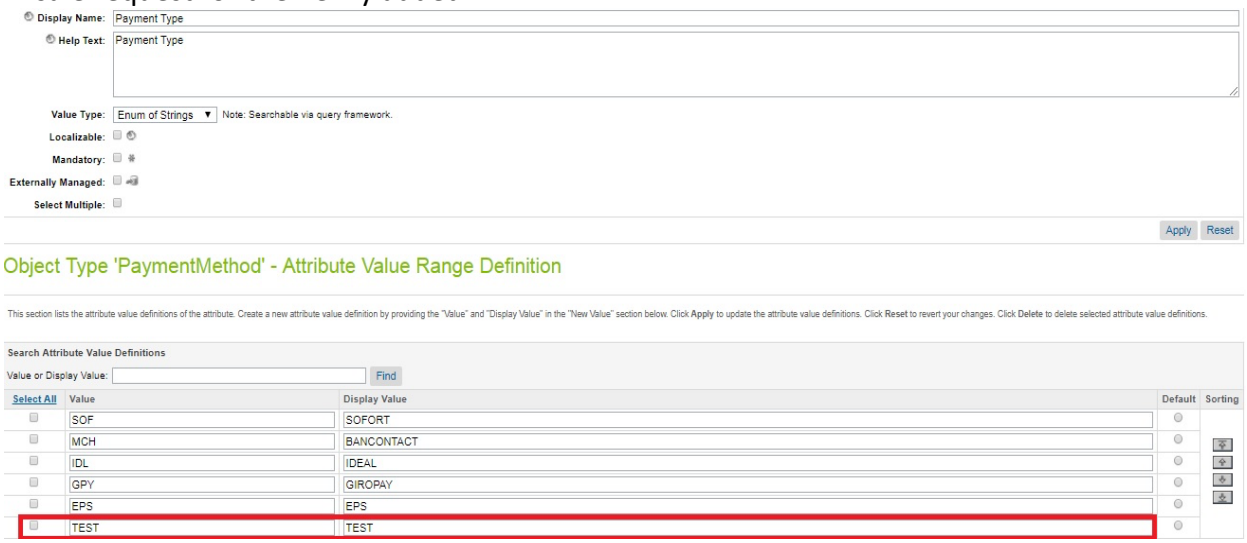
Configure New Bank Transfer Type APM using Business Manager Console

New Bank Transfer Type APM which consist of CyberSource sale and check status service could be configured at Merchant Tools -> Ordering -> Payment Methods. Follow below mentioned steps to add a new APM.

- Go to Administration -> System Objects -> Payment Methods -> Attribute Definitions -> select payment Type custom attribute

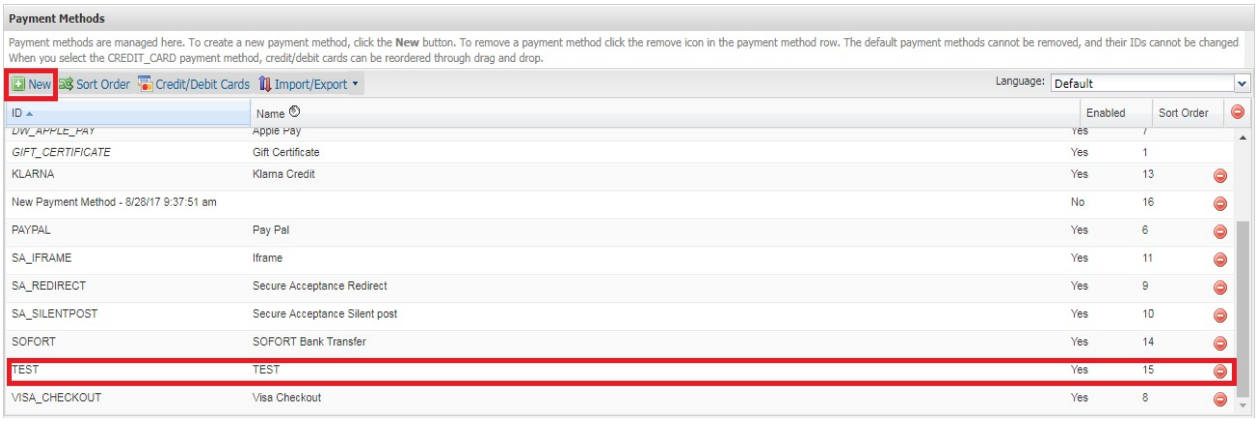


- Add new value of payment type for newly added APM, Value refers the apPaymentType value in sale request for the newly added APM

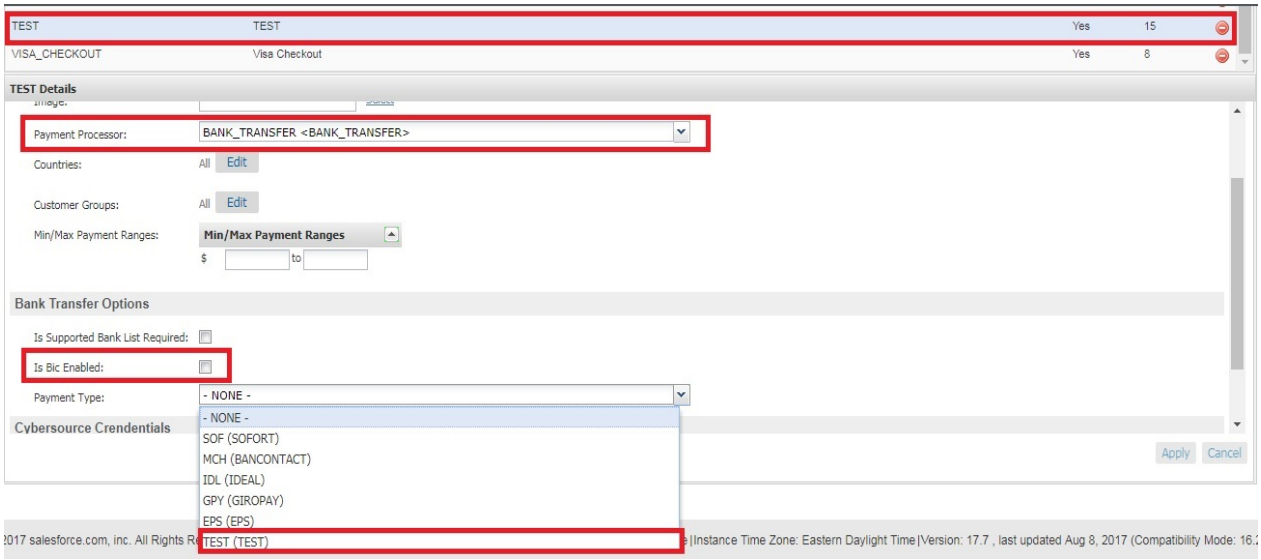


- Go to Merchant Tools -> Ordering -> Payment Methods and click New button
- Provide ID and Name of new APM and select Yes from enabled drop down to enable that APM

Payment Methods



- Select payment processor to BANK_TRANSFER for newly added APM and select payment type to newly added payment type for new APM
- If BIC field is required to display on billing screen, select **Is Bic Enabled** checkbox to true



- With all these changes, new APM would be displayed on billing page to process with Bank Transfer payment

Bank Transfer sequence flow:

1. Select SOFORT or BANCONTACT payment methods from billing page and proceed with the payment
2. For IDEAL, select bank from bank list, for EPS and GIROPAY, enter BIC number and proceed with the payment
3. Creates CyberSource sale service request using bill-to, item data, purchase total data and merchant descriptor data from the current basket
4. Make actual service call to CyberSource sale service
5. If service returns 'ACCEPT' as decision, 100 as reason code and 'pending' as payment status, redirect the user to bank site
6. If service returns 'ACCEPT' as decision, 100 as reason code and 'failed' as payment status, exit immediately, redirect back the user to merchant site along with error message and change the status of order to failed
7. If service returns 'REJECT' as decision, exit immediately, redirect back the user to merchant site along with error message and change the status of order to failed
8. If service returns 'REVIEW' as decision, complete the order transaction but order status would be created itself
9. After successful authorization of amount on bank site, user would be redirected back to merchant site. After redirection, call to CyberSource check status service would be made to complete the transaction
10. If check status service returns 'ACCEPT' as decision, 100 as reason code and 'authorized' or 'settled' as payment status, complete the order and modify order and export status
11. If check status service returns 'ACCEPT' as decision, 100 as reason code and 'pending' as payment status, complete the order without modifying order and export status
12. If check status service returns 'ACCEPT' as decision, 100 as reason code and 'abandoned' or 'failed' as payment status, exit immediately and change the status of order to failed
13. If check status service returns 'REJECT' or 'ERROR' as decision, exit immediately and change the status of order to failed

14. If check status service returns ‘REVIEW’ as decision, complete the order transaction bur order status would be created itself

Apple Pay

Developed REST Interface as a standalone services and cartridge does not have end-to-end direct integration with DW native Apple Pay Web/APP functionality. However interface has mechanisms to integrate individual methods with DW Native Apple Pay web/APP.
REST interface support can accept two type of parameter in JSON format.

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

Android Pay

Developed REST Interface as a standalone services and cartridge does not have end-to-end direct integration with DW native Android Pay Web/APP functionality.
REST interface support can accept two type of parameter in JSON format.

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

PayPal Express

PayPal Express provides set of services which enables you to do the checkout in faster and safer way. PayPal integration with Cybersource provides 3 ways to complete the checkout.

1) **Minicart**
2) **Cart Page**
3) **Billing Page**

Cybersource cartridge provides in-context checkout option i.e. when customer clicks on Checkout with PayPal on checkout page or mini cart, the website remains in the view while PayPal Window appears. The Customer logs in and selects a payment method and shipping address and confirms the payment and PayPal redirects the customer on order review page. Cybersource cartridge enables merchant to select the order type from BM i.e. Custom or Standard.

Custom Order

PayPal Custom Order enables you to perform multiple authorizations and multiple captures for each authorization. Below are the service requests for custom order

- **Sessions Service- Creates a payment with PayPal to set up an order**
- **Check Status Service- Requires the request ID value that was returned in Sessions Service and returns customer information**
- **Order Service- Requires the request ID value that was returned in Sessions Service and Payer**

ID, creates the order in anticipation of one or more authorization

- **Authorization Service- Requires request ID value that was returned in the order response, obtains the authorization**
- **Capture Service-Requires the request ID value that was returned in the authorization response and enables you to capture the entire authorized amount**

Standard Order

PayPal Standard Order enables merchants to accomplish authorize and capture at the same time. Below are the service requests for Standard order

- **Sessions Service- Creates a payment or Billing agreement with PayPal to set up an order**
- **Check Status Service- Requires the request ID value that was returned in Sessions Service and returns customer information**
- **Order Service- Requires the request ID value that was returned in Sessions Service and Payer ID, creates the order in anticipation of one or more authorization**
- **Sale Service- Requires the request ID value that was returned in order response, this service obtains authorization, and captures the authorized amount**

PayPal Credit

The PayPal credit button on your checkout page enables you to offer customer's PayPal Credit as a standalone payment method. PayPal Credit leverages the PayPal Express implementation. For PayPal credit only an additional flag paymentOptionID as true need to include in Sessions service request. Below are the service requests for PayPal Credit.

- **Sessions Service with additional flag paymentOptionID**
- **After getting the payment Transaction ID and request ID from sessions response , same service flow will be used as mentioned in PayPal express**

PayPal Billing Agreement

A PayPal Express Checkout billing agreement enables you to use Billing agreement ID for billing without requiring customer to specifically authorize each payment. Once the agreement created for customer, customer's Billing agreement ID would be used to Authorize the order. PayPal Billing agreement is applicable only for logged user, when customer checks Billing agreement checkbox from Billing page additional flag billingAgreementIndicator need to include in Session service request. Request ID returned in session service will be used in PayPal Billing agreement service, Billing Agreement ID would be saved in customer profile, this billing agreement ID would be used in further transaction. Cybersource Cartridge allows merchants to enable/disable billing agreement from BM site preferences. Below are the service requests for Billing Agreement

- **Sessions Service – Creates Billing agreement with PayPal to setup an order**

- **Billing Agreement Service- if customer profile does not contain Billing Agreement ID, this service would create the Billing agreement and saves the Billing agreement ID in customer profile. It requires the request ID value returned in sessions response**
- **Check Status Service- If customer profile contains billing agreement ID , sessions service would be skipped , billing agreement ID would be used in Check Status service**
- **Sale Service – Requires billing agreement ID returned in billing agreement service response. This service obtains authorization, and captures the authorized amount**

WeChat Pay

Wechat Pay authorization service allows storefront application to request for authorization for total ordered amount along with the currency. This make the web service call to CyberSource Wechat Pay apSale service to initiate payment request and authorize the amount and generates QR code which is to scanned by the customer for the final authorization on Wechat Pay Mobile App. After successful generation of QR code and authorizing it, controller make the web service call to check the payment status of initiated request.

The Salesforce Commerce Cloud CYBERSOURCE_WECHAT- Handle populates the payment initiate request with purchase total data and Wechat Payment type such as WQR and invoke the initiate web service call using CyberSource web service API.

Wechat Pay Authorization Sequence Flow:

1. Create CyberSource Wechat Pay Sale request using purchase total data, billing data(optional) from the current order object.
2. Set Wechat payment type to WQR.
3. After configuration make actual service call to Wechat Pay Sale request.
4. Validate Reason code and Decision of sale request and accordingly set the corresponding variables
   and get the QR Code Merchant url returned in the sale Response.
5. Display the QR code to the customer and ask him to scan the QR code for final authorization on Wechat Pay Mobile App.
6. If initiation is successful, then assign the required values in Demandware Payment Transaction object and create CyberSource Wechat Pay Check Status Request using Request ID of Initiate service response.
7. Make service call to Wechat Pay Check Status request to return the payment status of initiated sale request.
8. Wechat Pay Check Status service is done multiple times on the storefront page where QR code is displayed via AJAX and number of calls to be made and interval between the calls is configured in the BM as sitepreference.
9. Validate Reason Code and Payment status of check status service response and set the corresponding variables

10. If ReasonCode = 100 then check the payment status. If payment status is **settled** for service call then complete the checkout flow and place the order with “New” as order status and “Paid” as order payment status.
11. If ReasonCode = 100 and PaymentStatus = **pending**, complete the checkout flow by showing error message on the summary Page.
12. If ReasonCode = 100 and PaymentStatus = **abandoned**, fail the order and show message on the screen.

Validate authorization reason code and set corresponding values, based on Auth response code.

Business Manager Configuration

- Ensure you have imported ‘CyberSource/metadata/site_genesis_meta/sites/yourSiteIID/payment-methods.xml’
- Ensure you have imported ‘/Users/ksalemna/Documents/Krish/SFCC/Git/CYBS_ENT_Dev/cybersource-plugins-salesforce-1/CyberSource/metadata/site_genesis_meta/meta/Cybersource-metadata.xml’

1. Go to: “Merchant Tools > Site Preferences > CyberSource WeChat Pay
2. Configure Test Reconciliation ID for WeChat Pay to one of the drop values for the expected status to test flow.
3. Configure WeChatPayTransactionTimeout for the the transaction timeout of the QR code.
4. Configure CheckStatusServiceInterval for the amount of time to wait before each AP check status call.
5. Configure NumofCheckStatusCalls for the max amount of calls to check status before user is redirected to billing page.

Configurations

Site Preferences: Cybersource_WeChat Pay

Preference Name	Usage
Test Reconciliation ID for WeChat Pay	Sets the status of the AP Sale such as settled, pending, abandoned, or failed
WeChatPayTransactionTimeout	Transaction Timeout for QR Code in WeChat Pay in seconds
CheckStatusServiceInterval	Interval in seconds before checking status of AP sale
NumofCheckStatusCalls	Max number of calls to check status for each AP sale

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.

Secure Acceptance Merchant Notification Post Batch Job

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

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

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

2. Order not updated in checkout journey then merchant post response read from custom object in JSON form and information updated in the order
    a. Billing/shipping address
    b. Order status as New/Failed
    c. Payment authorization response
    d. Card get saved for logged in user if customer opted in checkout journey

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

IDeal Bank Options List Service Batch Job

Cartridge will provide a batch Job to fetch the options details for Ideal and store them into salesforce commerce cloud Business Manager Custom objects. Job can be configured to run at any defined interval using SFCC Job framework. Ideally it should not run on frequent basis but it'spurelyconfigurable to run at merchant choice and need and SFCC capabilities. Job will populate/update the SFCC custom objects with the response data. If Shopper selects "IDEAL" as payment method on commerce cloud billing page, Bank options list will be displayed as drop down for shopper to select a preferred bank to proceed. The selected bank name will be passed into IDEAL sale service request.

Alternate Payment Check Status Batch Job

AP check status batch job process Demandware orders placed by ALIPAY/Bank Transfer/Klarna as payment method by making web service call to AP Check Status Service.
The Demandware APCheckStatusJob.js script module is called from batch job that populates the check status request with Request ID and Payment Type generated and stored in Demandware Payment Transaction custom attribute for every order placed by Alipay/Bank Transfer/Klarna as payment method and invokes the Check Status web service call using CyberSource web service API. The Job Picks all the orders placed before 30 minutes of the current time (i.e. LagTime) which is configurable through Job as per merchant need.

AP Check Status Batch Job Sequence Flow:

1. Query on all the Demandware orders placed with order status as 'CREATED', export status as 'NOT EXPORTED' and Lag time as set in Jobs parameter.
2. Iterate through all orders whose Payment processor are CYBERSOURCE_ALIPAY / BANK_TRANSFER / KLARNA_CREDIT and get the Request Id stored in Order Payment Transaction custom object attribute.
3. Pass the Request Id and Payment Type to AP Check Status Service and make the actual service call.
4. Validate Decision, Reason Code and Payment status of check status service response and set the corresponding variables
5. If Decision = ACCEPT and ReasonCode = 100 then check the payment status as:
   - For Payment Status as COMPLETED/ AUTHORIZED/ SETTLED - Update the order with Order status to "New", Confirmation Status to "Confirmed" and export status to "Ready For Export". Also update Order Payment Transaction custom object attributes as apPaymentStatus, order payment status to "PAID".
   - For Payment Status as PENDING, no need to update any Demandware status in case of PENDING Payment Status
   - For Payment Status as ABANDONED/ TRADE_NOT_EXIST/FAILED, fail the order.
6. With Decision = REJECT/ERROR and any other ReasonCode except 100, fail the order.

Use Cases Scenarios

Credit Card / Visa Checkout / Apple Pay 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 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 fatal error (email alert)	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 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	

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.

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
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

Upgrade to 3DS2.0

If you are currently using CYBS cartridge and would like to upgrade to 3DS2.0, please refer below doc. If you are not able to download find the doc under cartridge documentation folder.

3DS2.x Dev Guide for Controller.docx

Tax Service

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	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.

shipping price totals or adjusted basket total price has changed.	

Secure Acceptance Authorization

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

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

VISA Checkout Decrypt

List of use cases and appropriate action taken listed below:

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

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	CYB Code	CYB 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.

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
2. **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

Klarna & Bank Transfer

Response	DW Storefront Action	CYB Code	CYB Suggested response
Successful transaction.	Continue Checkout	100	
Validation Errors			
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.

General decline by the processor	Should not occur as validation should catch this Show user “denied” error message Log error message into Demandware logs	203	Processor declined the transaction because of funding source problems, or the transaction was flagged as high risk.
General decline by the processor	Should not occur as validation should catch this Show user “denied” error message Log error message into Demandware logs	204	Payment declined because of insufficient funds in the account.
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.

PayPal Express / Credit / Billing Agreement

The following table outlines the possible SFCC 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	CYB 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 SFCC 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 SFCC 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 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 error	151	Wait a few minutes and resend the request.
The request just wait and then timeout, ends up as exception on the SFCC 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 SFCC. 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 SFCC logs	223	
General decline by PayPal.	Show user “Unable to process – Call Customer Service” error message Log error message into SFCC 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 SFCC logs	234	Do not resend the request. Contact Customer Support to correct the configuration problem.
PayPal rejected the transaction. A successful transaction was already	Show user “Unable to process – Call Customer Service” error message	238	

completed for this PayPal Token value.	Log error message into SFCC logs		
Fraud Management			
The order is marked for review by Decision Manager.	Proceed with checkout Leave SFCC order "unconfirmed"	480	
The order is rejected by Decision Manager.	Show user "Unable to process – Call Customer Service" error message Log error message into SFCC logs	481	

CyberSource PayPal / PayPal Credit Transactional Flow:
Step 1: Sessions Service request and reply— accept item object, bill to, ship to objects, purchase data to generate the PayPal payment transaction ID.
Step 2: Check Status Service request and reply — accept request id, payer id and PayPal payment transaction ID generated by sessions service and return address verification response, payer details and address details.
Step 3: Order Service 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.
CyberSource PayPal Billing Agreement Transactional Flow:

Step 1: If Billing Agreement exits for the customer Step 2 will executed. If not Session service will execute.

Step 2: Billing agreement Service request and reply – accept request id of session service.

Step 3: Check Status Service request and reply – accept Customer billing agreement ID.

Step 4: Sale Service request and reply – accept customer billing agreement ID.

Use Case 1: Checkout using PayPal Express Checkout on Cart Page
"PayPal Checkout" button has been added on SFCC reference Site Genesis.
Use Case 2: Checkout using "PayPal Checkout" button on mini cart

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

Use case 4: Checkout using PayPal Credit as payment method on Payment page.

Use case 5: Checkout using PayPal Billing agreement as payment method on Payment page.

Conversion Detail Report

Integration Overview

This job uses a simple API to retrieve order decisions from CyberSource and update the order confirmation status in SFCC.
As described in the previous section, when Decision Manager is enabled, a certain number of orders will be flagged for review, and not fully confirmed in your SFCC storefront. When integrating with your OMS, you must decide if you want to export orders that are set to 'Not Confirmed'. If you only send Confirmed orders to your OMS, you will need this job to update the confirmation status of the orders that have been reviewed and accepted.

Implementation

If you are not using Decision Manager, you do not need this job. If your OMS asks you to send them 'Not Confirmed' Orders, you may or may not want this job. You will need to determine if the Order confirmation status is required, or desired in SFCC for your business needs.
To Integrate this job into your site, follow the below steps:

1. Navigate to 'Administration > Operations > Job Schedules'
2. Create the Job '**CyberSource: Decision Manager Order Update (Controllers & Pipelines)**'
3. Select the 'Step Configurator tab.
4. Select the Sites you want the Job to run on, from the 'Scope' button.
5. Navigate to the 'Schedule and History' tab and configure the frequency you would like the job to run.
6. Create a step that executes the script:
   int_cybersource/cartridge/scripts/jobs/DMOrderStatusUpdate.js function : orderStatusUpdate
   and ensure it is set to transactional.
7. Ensure the 'Enabled' check box is selected.
8. Update Custom Parameters MerchantId, SAFlexKeyID and SAFlexSharedSecret.

When moving to a production environment, the URL for the API call needs to be updated in the service credentials in Business Manager.

This job is configured based on various site preferences described below:

Configuration

Configure values for below **job parameters** in step **UpdateOrderStatus**

Parameter Name	Usage
MerchantId	CS Merchant ID for the account to get Decisions from.
SAFlexKeyID	Key ID. Work with CS to generate this value.
SAFlexSharedSecret	Shared secret. Work with CS to generate this value.

Job Parameters: UpdateOrderStatus

Preference Name	Site Pref Group	Usage
CS Decision Manager OrderUpdate Lookback time	Core	Number of hours the job will look back for new decisions. CS does not support lookbacks over 24 hours. Do not set above 24.
Secure Acceptance Flex Host Name	Secure Acceptance	Host Name. CS can provide this value.

Alternate Payment Check Status Job

List of use cases and appropriate action taken listed below:

Decision	Reason Code	Payment Status	Description	Result
ACCEPT	100	COMPLETED authorized settled	Successful transaction.	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
ACCEPT	100	PENDING pending	Successful transaction.	No Demandware Order status updated
ACCEPT	100	ABANDONED TRADE_NOT_EXIST Failed abandoned	Successful transaction.	Oder FAILED in Demandware
REJECT/FAILED	102,150,203, 204,233	failed	One or more fields in the request	Oder FAILED in Demandware

			contain invalid data. Processor declined the transaction because of funding source problems, or the transaction was flagged as high risk. Payment declined because of insufficient funds in the account Processor declined the transaction because of tax errors or government compliance errors	

CyberSource Decision and DW Order Status Mapping

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

Limitations, Constraints

- 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.
- Cartridge does not provide changes to support the styling of error and validation messages. Merchant need to make the required changes to meet the style guide for error and validation messaging as per their storefront implementation

- Cartridge support s DW provided form field validations only

Currently implemented with limitations and constraints:

- 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.
- Merchant to decide “Master Card Auth Indicator” as “Pre Authorization” “FinalAuthorization ” or “Undefined” from site preferences for master card.
- Cybersource must take into account Fraud and Risk details, AVS and card security codes available in Payload during transaction authorization, Cartridge will not be performing any additional security/risk checks except the existing CC Auth response handling
- **Unit Test Interface:**
    1. Unit Test Services are developed for the standalone testing purpose only and should not be used directly into production
    2. Custom user interface for view, update and delete subscription. All functionalities are created and working in stand-alone mode in **CYBServicesTesting.js** controller. They have to customized and integrated as per the merchant specific needs
    3. Custom user interface for Full Authorization Reversal. Full Authorization reversal is created and working in stand-alone mode in **CYBServicesTesting.js** controller. It has to customized and integrated as per the merchant specific needs
- **Alipay Authorization:**
    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
    2. CNY is the only hardcoded currency value that has been used for Alipay Domestic requests
    3. 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
- **Secure Acceptance:**
    1. Limit storefront order setting must be disable if Merchant post URL is configured
    2. Cartridge supports five types of cards in secure acceptance (Visa, master card, amex, maestro international, discover)
- **Visa Checkout:**
    1. “Save Card” option will not be available in the demandware checkout journey, which means tokenization will not be applicable for Visa Transactions
- **Apple Pay REST Interface:**
    1. Tokenization and Payer authentication is not supported with Apple Pay Transactions
    2. Developed REST Interface are just standalone services only and does not support direct integration with DW native Apple Pay Web functionality, however interface is developed in such a way that Merchant can use individual methods to integrate with DW Native Apple Pay web
- **Bank Transfer**

1. Bank Transfer functionality is specific to APMs with sale and check status service. If service implementation changes apart from sale and check status or service input changes are required for any other APM, code changes would be required to made to successfully execute the Bank Transfer functionality

Compatibility

This cartridge is tested with Demandware Site genesis release code base 17.2 and compatibility mode of 16.2.

Implementation Guide

Custom Code

Pre-Requisite: Make sure the controller cartridges of site site-genesis is (say, e.g. app_storefront_controllers and "int_cybersource, int_cybersource_controllers" are specified in Site Settings path under Manage Sites > Merchant Site as per current site
Modify the references of actual storefront cartridges in CyberSource cartridges under CybersourceConstants.ds during CyberSource integration. Cybersource cartridge is developed assuming storefront cartridge naming conventions as:

- app_storefront_core
- app_storefront_controllers

Generic Section

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
        };
    }
/**
```

```
    *SetsthetransactionIDforthepaymentinstrument.
    */
    var handlePaymentTransaction = function () {
            paymentInstrument.getPaymentTransaction().setTransactionID(order.getOrderNo());
        };

    for (var i = 0; i < paymentInstruments.length; i++) {
    var paymentInstrument = paymentInstruments[i];

    if (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor() === null) {

              Transaction.wrap(handlePaymentTransaction);

           } else {

    authorizationResult =PaymentProcessor.authorize(order, paymentInstrument);

    if (authorizationResult.not_supported || authorizationResult.error || authorizationResult.failed) {
    return {
                    error: true
                };
              }else if(authorizationResult.returnToPage){
          return {
          returnToPage :true,
          order : order
                    };
                }

            }
          }
        }

    return authorizationResult;
}
```

Update "start" functionto handle payment results

Add below snippet to handle payment different results

[Note: this function contains generic code for all APM's to reduce redundancy: please refer the code below]

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

  }else if(handlePaymentsResult.returnToPage){
    app.getView({
```

```
                    Order : handlePaymentsResult.order
                }).render('checkout/summary/summary');
                    return {};
    }else if(handlePaymentsResult.redirection){
            response.redirect(handlePaymentsResult.redirectionURL);
            return {};
    }else if(handlePaymentsResult.carterror){
            app.getController('Cart').Show();
            return {};
    }else if(handlePaymentsResult.intermediate){
       app.getView({
            alipayReturnUrl : handlePaymentsResult.alipayReturnUrl
                }).render(handlePaymentsResult.renderViewPath);
                    return {};
    } else if(handlePaymentsResult.intermediateSA){
            app.getView({
                    Data:handlePaymentsResult.data, FormAction:handlePaymentsResult.formAction
                }).render(handlePaymentsResult.renderViewPath);
                    return {};
    }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.process3DRedirection) {
            return handlePaymentsResult;
    } else if (handlePaymentsResult.review) {
            ReviewOrder({Order:order});
            return {};
    } else if (handlePaymentsResult.pending) {
             ReviewOrder({Order:order});
             return {};
    }
var orderPlacementStatus = Order.submit(order);
    if (!orderPlacementStatus.error) {
       clearForms();
    }
    return orderPlacementStatus;
}
```

1. Add new method to handle the failed order

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

Add "ReviewOrder" function

Add the review order function with the code snippet below

```
/**
*Leaveorderincreatedstateindemandwareandsendorderconfirmationemail
*@paramargs
*/
function ReviewOrder(args) {
        var Email = app.getModel('Email');
        var Resource = require('dw/web/Resource');
        var order = args.Order;
        // Send order confirmation and clear used forms within the checkout process.
        Email.get('mail/orderconfirmation', order.getCustomerEmail())
          .setSubject((Resource.msg('order.orderconfirmation-email.001', 'order', null) + ' ' +
order.getOrderNo()).toString())
          .send({
            Order: order
          });
        // Clears all forms used in the checkout process.
        clearForms();
        app.getController('COSummary').ShowConfirmation(order);
```

```
        return;
}
```

Add "submitOrder" function

Add the submit order function with the code snippet below

```
/**
*Submittheorderandsendorderconfirmationemail
*@paramargs
*/
function SubmitOrder(args) {
var orderPlacementStatus = Order.submit(args.Order);
if (!orderPlacementStatus.error) {
      clearForms();
      app.getController('COSummary').ShowConfirmation(args.Order);
return;
   }

   app.getController('COSummary').Start();
      }
```

Update "submit" function

Replace the submit function with the code snippet below

```
/*
 * Asynchronous Callbacks for SiteGenesis.
 * Identifies if an order exists, submits the order, and shows a confirmation message.
 */
function submit(args) {
        var Provider = require('int_cybersource_controllers/cartridge/scripts/Provider');
        var providerParam = request.httpParameterMap.provider.stringValue;
        if(!empty(providerParam)) {
                var providerResult = Provider.Check(args);
                if(!empty(providerResult)){
                        if(providerResult.pending){
                                ReviewOrder({Order:providerResult.Order});
                                return;
                        }else if(providerResult.load3DRequest){
                                app.getView().render('cart/payerauthenticationredirect');
                                return;
                        } else if(providerResult.submit){
                                SubmitOrder({Order:providerResult.Order});
                                return;
                        } else if(providerResult.error){
                                fail({Order:providerResult.Order});
                                return;
                        } else if(providerResult.cancelfail){

        app.getController('COSummary').Start({PlaceOrderError:providerResult.PlaceOrderError});
                                return;
                        } else if(providerResult.carterror){
                                app.getController('Cart').Show();
                                return;
                        } else if(providerResult.redirect){
                                app.getView({Location : providerResult.location}).render(providerResult.render);
```

```
                                        return;
                                    }
                                } else {
                                    return;
                                }
                            }
                            app.getController('Cart').Show();
                            return;
                        }
```

Update Export functions

```
exports.Fail = guard.ensure(['https'], fail);
exports.ReviewOrder = ReviewOrder;
exports.SubmitOrder = SubmitOrder;
exports.FailWeChatOrder = guard.ensure(['https'], failWeChatOrder);
```

Update clearForms function

```
function clearForms() {
   // Clears all forms used in the checkout process.
   session.forms.singleshipping.clearFormElement();
   session.forms.multishipping.clearFormElement();
   session.forms.billing.clearFormElement();
   var privacyObject = session.privacy;
            for (var property in privacyObject){ privacyObject[property] = ""; }

}
```

Add FailWeChatOrder function

```
function failWeChatOrder(args) {
            var PlaceOrderError = args.PlaceOrderError!= null ? args.PlaceOrderError : new
dw.system.Status(dw.system.Status.ERROR, "confirm.error.declined");
            session.custom.SkipTaxCalculation=false;
            app.getController('COSummary').Start({PlaceOrderError : PlaceOrderError});
            return;
}
```

Controller - COBilling.js

Update Export Function

```
exports.ReturnToForm = guard.ensure(['https'], returnToForm);
```

Update "resetPaymentForms()" function

Invoke cybersource cartridge "ResetPaymentForms" function after cart basket retrieved.
Remove BML payment instruments from PayPal and credit card condition
Also remove PayPal payment instrument from BML IF condition at the end

Add if condition after cart object

```
function resetPaymentForms() {

var cart = app.getModel('Cart').get();
if (null != cart && !empty(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID)) {
            var Cybersource = require('int_cybersource_controllers/cartridge/scripts/Cybersource');
            Cybersource.ResetPaymentForms({Basket:cart.object, PaymentType:
app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value});
   }
var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
var status = Transaction.wrap(function () {
if
(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(CybersourceConstants.METHO
D_PAYPAL)) {
        app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();

        cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD_CREDIT_CARD));
      } else if
(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD_
CREDIT_CARD)) {
cart.removePaymentInstruments(cart.getPaymentInstruments(CybersourceConstants.METHOD_PAYPAL));
      } else if
(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD_
BML)) {
        app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();

if (!app.getForm('billing').object.paymentMethods.bml.ssn.valid) {
return false;
        }

        cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD_CREDIT_CARD));
      }
return true;
   });

return status;
}
```

Update the validateBilling() function

Update the if condition of selectedPaymentMethodID by adding highlighted section

```
function validateBilling() {
   if (!app.getForm('billing').object.billingAddress.valid) {
      return false;
   }

   if (!empty(request.httpParameterMap.noPaymentNeeded.value)) {
      return true;
   }

   if (!empty(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value)
```

```
                    &&
app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD_CREDIT_CARD)
                    && empty(app.getForm('billing').object.paymentMethods.creditCard.selectedCardID.value)) {
      if (!app.getForm('billing').object.valid) {
         return false;
      }
   }

   return true;
}
```

Update the validatePayment () function

Add transaction.wrap and move the cart.validatePaymentInstruments if condition inside this.

```
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;
}
```

Update "saveCreditCard" function

Replace the entire function with the below snippet,.

```
function saveCreditCard() {
         var Cybersource = require('int_cybersource_controllers/cartridge/scripts/Cybersource');
         return Cybersource.SaveCreditCard();
}
```

Update "selectCreditCard" function

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

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

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

Update "populateCreditCardForm" function

Add new parameter "selectedPaymentMethod" and add Switch condition to handle different APM's as below:

[Note: All app.js changes are similar to billing.js, please refer the below section for billing.js changes, below method contains generic code used for different payment methods as given below]

```
function populateCreditCardForm(cardID,selectedPaymentMethod) {
        // load card details
        var url = util.appendParamToURL(Urls.billingSelectCC, 'creditCardUUID', cardID);
        ajax.getJson({
                url: url,
                callback: function (data) {
                        if (!data) {
                                window.alert(Resources.CC_LOAD_ERROR);
                                returnfalse;
                        }

                        switch (selectedPaymentMethod) {
                        case "SA_REDIRECT":
                                $('.payment-method-expanded .saCCToken .field-wrapper').val(data.selectedCardID);

$("#dwfrm_billing_paymentMethods_creditCard_selectedCardID").val(data.selectedCardID);
                                break;
                        case "SA_IFRAME":
                                $('.payment-method-expanded .salframeCCToken .field-
wrapper').val(data.selectedCardID);

$("#dwfrm_billing_paymentMethods_creditCard_selectedCardID").val(data.selectedCardID);
                                break;
                        case "CREDIT_CARD":
                                setCCFields(data);
                                break;
                        default:
                                setCCFields(data);
                        }

        }
        });
}
```

Update "#creditCardList" on change function

- **Update the method inside export.init () by adding parameter "selectedPaymentMethod":**

```
// select credit card from list
        $('#creditCardList').on('change', function () {
                var cardUUID = $(this).val();
                if (!cardUUID) {$($checkoutForm).find('input[name$="_selectedCardID"]').val(''); return;}
                populateCreditCardForm(cardUUID,selectedPaymentMethod);

                // remove server side error
                $('.required.error').removeClass('error');
                $('.error-message').remove();
                                                        });
```

Update "setCCFields "function

- **Get selected payment method from input type and set CVN, expiry month and expiry year based on selected payment method[this change will work for Silent Post and credit card]**

```
function setCCFields(data) {
        var $creditCard = $('[data-method="CREDIT_CARD"]');
        $creditCard.find('input[name$="creditCard_owner"]').val(data.holder).trigger('change');
        $creditCard.find('select[name$="_type"]').val(data.type).trigger('change');
        $creditCard.find('input[name*="_creditCard_number"]').val(data.maskedNumber).trigger('change');
        var selectedPaymentMethodID = $('input[name$="_selectedPaymentMethodID"]:checked').val();
        if(selectedPaymentMethodID == 'SA_SILENTPOST'){
                $creditCard.find('[name$="_month"]').val(data.expirationMonth);
                $creditCard.find('[name$="_year"]').val(data.expirationYear);
        }
        else{
                $creditCard.find('[name$="_month"]').val(data.expirationMonth).trigger('change');
                $creditCard.find('[name$="_year"]').val(data.expirationYear).trigger('change');
        }
        $creditCard.find('input[name$="_cvn"]').val('').trigger('change');
        $creditCard.find('[name$="creditCard_selectedCardID"]').val(data.selectedCardID).trigger('change');
        $creditCard.find("input[name$='_cvn']").val('');
}
```

Update "updatePaymentMethod "function

- **Based on payment method Id selected, this method will hide/show the button or checkboxes for different APM to make it visible on billing page.**

[Note: This method contains generic code for different payment methods as given below]

```
function updatePaymentMethod(paymentMethodID) {
        var $paymentMethods = $('.payment-method');
        $paymentMethods.removeClass('payment-method-expanded');
        var dataMethod = paymentMethodID;
        if (paymentMethodID=='SA_SILENTPOST') {
                dataMethod = 'CREDIT_CARD';
        }
        var $selectedPaymentMethod = $paymentMethods.filter('[data-method="' + dataMethod + '"]');
```

```
            if ($selectedPaymentMethod.length === 0) {
                        $selectedPaymentMethod = $('[data-method="Custom"]');
            }
            if (paymentMethodID=="VISA_CHECKOUT") {
                        $(".continue-place-order").hide();
                        $(".visacheckoutbutton").show();
            }
            else if (paymentMethodID=="PAYPAL" || paymentMethodID=="PAYPAL_CREDIT") {
                        $("#billingAgreementCheckbox").attr('checked',false);
                        $(".continue-place-order").hide();
            }
            else {
                        $(".continue-place-order").show();
                        $(".visacheckoutbutton").hide();
            }
            if (paymentMethodID=="CREDIT_CARD" || paymentMethodID=="SA_SILENTPOST") {
                        $(".spsavecard").show();
            } else if ((paymentMethodID=="SA_REDIRECT" || paymentMethodID=="SA_IFRAME") &&
SitePreferences.TOKENIZATION_ENABLED) {
                        $(".spsavecard").show();
            }
            else {
                        $(".spsavecard").hide();
            }


            $selectedPaymentMethod.addClass('payment-method-expanded');

// ensure checkbox of payment method is checked
   $('input[name$="_selectedPaymentMethodID"]').removeAttr('checked');
   $('input[value=' + paymentMethodID + ']').prop('checked', 'checked');

   formPrepare.validateForm();
}
```

Update “exports.init “function

- **Add below code snippetafter formPrepare.init to handle card details on billing page based on APM selected**

```
formPrepare.init({
      formSelector: 'form[id$="billing"]',
      continueSelector: '[name$="billing_save"]'
   });

            var $ccContainer = $($checkoutForm).find(".payment-method").filter(function(){
                        return $(this).data("method")=="CREDIT_CARD";
            });
            $($checkoutForm).find('input[name$="_selectedCardID"]').val('');
            $($checkoutForm).find('input[name*="_number"]').val('');
```

```
        $ccContainer.find('input[name*="_number"]').on('change',function(e){
                $($checkoutForm).find('input[name$="_selectedCardID"]').val('');
        });
        $ccContainer.find('input[name$="_owner"]').on('change',function(e){
                $($checkoutForm).find('input[name$="_selectedCardID"]').val('');
        });
        $ccContainer.find('select[name$="creditCard_type"]').on('change',function(e){
                $($checkoutForm).find('input[name$="_selectedCardID"]').val('');
        });

        $ccContainer.find('select[name*="expiration"]').on('change',function(e){
                $($checkoutForm).find('input[name$="_selectedCardID"]').val('');

                var selectedPaymentMethodID = $('input[name$="_selectedPaymentMethodID"]:checked').val();
                var cardNumber = $($checkoutForm).find('input[name*="_number"]').val();
                if(cardNumber.indexOf('****') != -1 && selectedPaymentMethodID == 'SA_SILENTPOST'){
                        $($checkoutForm).find('input[name*="_number"]').val('');
                }

        });

        var $ccNum = $ccContainer.find("input[name$='_number']");
        // default payment method to 'CREDIT_CARD'
        updatePaymentMethod((selectedPaymentMethod) ? selectedPaymentMethod : 'CREDIT_CARD');
        $selectPaymentMethod.on('click', 'input[type="radio"]', function () {
                updatePaymentMethod($(this).val());
                                                        });
// select credit card from list
   $('#creditCardList').on('change', function () {
```

Update “updatePaymentMethod” function at line 84 and 501

- **Add below highlighted code snippet for bank transfer**

```
 } else if ((paymentMethodID=="SA_REDIRECT" || paymentMethodID=="SA_IFRAME") &&
SitePreferences.TOKENIZATION_ENABLED) {
                $(".spsavecard").show();
        }
        else {
                $(".spsavecard").hide();
        }

        var isBicRequired = $selectedPaymentMethod.data('bicrequired');
        if(isBicRequired){
                $(".bic-section").show();
        }else{
                $(".bic-section").hide();
        }

$selectedPaymentMethod.addClass('payment-method-expanded');
```

Form - customeraddress.xml

Include the following code just above the action events

```
<field formid="phone" label="profile.phone" description="address.phone.example" type="string" mandatory="true" binding="phone" max-length="20"/>
        <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 formId

```
<include formid="address" name="customeraddress"/>
```

Form – creditcard.xml

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

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

- **Add more year options as below:**

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

Template - paymentmethods.isml

1. **Add code to declare CyberSource constant file**

Line 4 to Line 6

```
<iscomment> TEMPLATENAME: paymentmethods.isml </iscomment>
```

```
<isinclude template="util/modules"/>
<isscript>
                                                    var CybersourceConstants =
require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
 </isscript>
<isif condition="${pdict.OrderTotal > 0}">
```

2. **Add conditional statement to show declined message for PayPal, visa checkout and secure acceptance**

```
Line 14 to Line 16
<legend>

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

                                                    <div class="payment-method-options form-
indent">
                                                        <isloop
items="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.options}"
var="paymentMethodType">
```

3. **Remove red highlighted code and replace with a condition to display bank transfer payment methods**

```
 Line 28
<isset name="radioID" value="${paymentMethodType.value}" scope="page"/>
    <div class="field-wrapper">
            <input id="is-${radioID}" type="radio" class="input-radio <isif
    condition="${paymentMethodType.object.paymentProcessor.ID.equalsIgnoreCase("bank_transfer") == true
    }">bank-transfer</isif>"
    name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlName}"
    value="${paymentMethodType.htmlValue}" <isif condition="${paymentMethodType.value ==
    pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlValue}">checked="checked"</isif> />
    <input id="is-${radioID}" type="radio" class="input-radio <isif
condition="${paymentMethodType.object.paymentProcessor.ID.equalsIgnoreCase("bank_transfer
") == true }">bank-transfer</isif>"
```

```
name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlName}"
value="${paymentMethodType.htmlValue}" <isif condition="${paymentMethodType.value ==
pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlValue}">checked="c
hecked"</isif> /> </div>
```

4. **Remove Credit card and bml payment method section as highlighted below**

```
Remove code from line 44 to 94

<iscomment>
    Credit card block
    ------------------------------------
</iscomment>

<div class="form-row required">
    <label>
        <span class="required-indicator">${Resource.msg('billing.requiredIndicator','checkout',null)}</span>
        <span>${Resource.msg('billing.creditcardlisterpdate', 'checkout', null)}</span>
</label>
```

5. **Add include for countries file**

```
Line 88
<isscript>
    var currentCountry = require('~/cartridge/scripts/util/Countries').getCurrent(pdict);
</isscript>
```

6. **Remove code using following reference**

```
Line 46 to Line 90

<isdynamicform
formobject="${pdict.CurrentForms.billing.paymentMethods.creditCard.expiration}"
formdata="${currentCountry.dynamicForms.expirationInfo}"/>
</div>
<isscript>
var help = {

<div class="form-row form-caption">
<isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.bml.termsandconditions}"
type="checkbox"/>
 </div>
</div>
```

7. **Add below code for PayPal changes**

```
<isinclude template="common/paymentmethods"/>
                <iscomment>
                        Custom processor
                        ---------------------------------------------------------------
                </iscomment>

                <div class="payment-method <isif condition="${!empty(pdict.selectedPaymentID) &&
pdict.selectedPaymentID==CybersourceConstants.METHOD_PAYPAL}">payment-method-expanded</isif>" data-
method="PAYPAL">
                        <!-- Your custom payment method implementation goes here. -->
                        <isif
condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('payPalBillingAgreements') &&
!empty(pdict.CurrentCustomer.profile) && !empty(pdict.CurrentCustomer.profile.custom.billingAgreementID)}">
                                <input type="image"
src="https://www.paypal.com/en_US/i/btn/btn_xpressCheckout.gif" alt="PayPal Express"
class="billingAgreementExpressCheckout"/>
                        <iselse>
                                <div id="paypal-button-container"></div>
                        </isif>


                        <isif condition="${pdict.CurrentCustomer.authenticated &&
dw.system.Site.getCurrent().getCustomPreferenceValue('payPalBillingAgreements')}">
                                <isif
condition="${!empty(pdict.CurrentCustomer.profile.custom.billingAgreementID)}">
                                        <input type="text" readonly="readonly" id="billingAgreementID"
value="${pdict.CurrentCustomer.profile.custom.billingAgreementID}"/>
                                <iselse>
                                        <input type="checkbox" name="billingAgreementCheckbox"
id="billingAgreementCheckbox">${Resource.msg('billing.billingagreement','checkout',null)}</input>
                                </isif>
                        </isif>
                </div>
                <div class="payment-method <isif condition="${!empty(pdict.selectedPaymentID) &&
pdict.selectedPaymentID==CybersourceConstants.METHOD_PAYPAL_CREDIT}">payment-method-expanded</isif>" data-
method="PAYPAL_CREDIT">
                        <div id="paypal-credit-container"></div>
                                                        </div>
```

8. **Take the value of selected payment method PayPal from constant file**

Line 147

```
<div class="payment-method <isif condition="${!empty(pdict.selectedPaymentID) &&
pdict.selectedPaymentID==CybersourceConstants.METHOD_PAYPAL}">payment-method-
expanded</isif>" data-method="Custom">
<!-- Your custom payment method implementation goes here. -->
${Resource.msg('billing.custompaymentmethod','checkout',null)}
</div>
```

Template – summary.isml

Below changes are generic for Secure Accpetance/Klarna_credit/Device fingerprint

1. **Set summary page tag for Secure Acceptance Iframe**

```
    <iscontent type="text/html" charset="UTF-8" compact="true"/>
    <isset name="summarypage" value="${true}" scope="page"/>
<isdecorate template="checkout/pt_checkout"/>
```

2. **Add below code above <isreportcheckout checkoutstep="${5}" checkoutname="${'OrderSummary'}"/>**

```
<isscript>
        var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
 </isscript>
<isset name="klarnarequired" value="${false}" scope="page"/>
<isif condition="${!empty(pdict.Basket)}">
<isset name="LineCntr" value="${pdict.Basket}" scope="page"/>
<iselseif condition="${!empty(pdict.Order)}">
<isset name="LineCntr" value="${pdict.Order}" scope="page"/>
</isif>
<isset name="summaryaction" value="${URLUtils.https('COSummary-Submit')}" scope="page" />
<script src="${URLUtils.staticURL('/lib/jquery/jquery-1.11.1.min.js')}" type="text/javascript"></script>
<isset name="paymentMethod" value="${null}" scope="page"/>
<isset name="isIFrame" value="${false}" scope="page" />
<isif condition="${!empty(LineCntr.getPaymentInstruments())}">
        <isloop items="${LineCntr.getPaymentInstruments()}" var="paymentInstr" status="loopstate">
                <isset name="paymentMethod"
value="${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).ID}" scope="page"/>
                <isif
condition="${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).ID==CybersourceConstants.METHOD_SA_IFRAME}">
                        <isset name="summaryaction" value="${URLUtils.https('COSummary-SubmitOrder')}"
scope="page" />
                        <isset name="isIFrame" value="${true}" scope="page" />
                <iselseif
condition="${CybersourceConstants.KLARNA_PAYMENT_METHOD.equals(dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).ID)}" >
                        <isset name="klarnarequired" value="${true}" scope="page"/>
                </isif>
        </isloop>
</isif>
<isif condition="${!empty(LineCntr)}">
                                        <isreportcheckout checkoutstep="${5}"
checkoutname="${'OrderSummary'}"/>
```

3. **Replace pdict.Basket with LineCntr at below places**

```
<isif condition="${!pdict.CurrentForms.multishipping.entered.value}">
                <ischeckoutprogressindicator step="3" multishipping="false"
rendershipping="${LineCntr.productLineItems.size() == 0 ? 'false' : 'true'}"/>
        <iselse/>
                <ischeckoutprogressindicator step="4" multishipping="true"
rendershipping="${LineCntr.productLineItems.size() == 0 ? 'false' : 'true'}"/>
```

```
</isif>
```

4. **Add condition for secure acceptance error by replacing place order error with below code**

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

5. **Replace pdict.Basket with LineCntr at below places**

```
 <iscomment>render each shipment</iscomment>
                        <isset name="shipmentCount" value="${0}" scope="page"/>

                        <isloop items="${LineCntr.shipments}" var="shipment" status="shipmentloopstate">

                                <isif condition="${shipment.productLineItems.size() > 0 ||
 shipment.giftCertificateLineItems.size() > 0}">

                                    <isset name="shipmentCount" value="${shipmentCount+1}" scope="page"/>
                                                                    <isif
condition="${LineCntr.shipments.size() > 1}">


. . . . .
. . . <existing code>. . .
. . . .
<iscomment>RENDER COUPON/ORDER DISCOUNTS</iscomment>
                                                                <isloop
items="${LineCntr.couponLineItems}" var="couponLineItem" status="cliloopstate">
. . . .
. . <existing code>. .
. . . .
<td class="item-total">
                                                <isif
condition="${couponLineItem.applied}">
                                                        <span class="coupon-
applied">${Resource.msg('summary.applied','checkout',null)}</span>
                                                <iselse/>
                                                        <span class="coupon-not-
applied">${Resource.msg('summary.notapplied','checkout',null)}</span>
                                                <isif>
                                            </td>


                                                                    </tr>
</isif>

                        </isloop>
```

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

6. **Update with below section for Klarna/Secure acceptance Iframe and device fingerprint and cardinal script related changes**

```
<div class="order-summary-footer">

                        <div class="place-order-totals">
                                <isordertotals p_lineitemctnr="${LineCntr}" p_showshipmentinfo="${false}"
p_shipmenteditable="${false}" p_totallabel="${Resource.msg('summary.ordertotal','checkout',null)}"/>
                        </div>
                <isif condition="${!empty(klarnarequired) && klarnarequired}" >
                        <div id="klarna_container"></div>
                        <div id="auth_button"></div>
                        <input type="hidden" id="processorToken" name="processorToken"
value=${session.privacy.processorToken}/>
                </isif>
                <isif condition="${!empty(pdict.Basket)}">
                        <form action="${summaryaction}" method="post" class="submit-order"
name="submitOrder">
                                <fieldset>
                                        <div class="form-row">
                                                <a class="back-to-cart <isif condition="${!empty(klarnarequired)
&& klarnarequired}"> hide</isif>" href="${URLUtils.url('Cart-Show')}">
                                                        <isprint
value="${Resource.msg('summary.editcart','checkout',null)}" encoding="off" />
                                                </a>
                                                <isif condition="${!empty(klarnarequired) && klarnarequired}" >
                                                        <input type="hidden" id="klarnaAuthToken"
name="klarnaAuthToken"/>
                                                </isif>
                                                <button class="button-fancy-large <isif
condition="${!empty(klarnarequired) && klarnarequired}"> hide</isif>" type="submit" name="submit"
value="${Resource.msg('global.submitorder','locale',null)}">
                                                        ${Resource.msg('global.submitorder','locale',null)}
                                                </button>
                                        </div>
                                        <input type="hidden" name="${dw.web.CSRFProtection.getTokenName()}"
value="${dw.web.CSRFProtection.generateToken()}"/>
<input type="hidden" id="DFReferenceId" name="DFReferenceId" />
                                </fieldset>
                        </form>
                </isif>
                </div>
<isif condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('CsDeviceFingerprintEnabled')}">
        <isinclude url="${URLUtils.url('CYBCredit-IncludeDigitalFingerprint')}"/>
</isif>
<isif condition="${isIFrame}">
<isinclude template="secureacceptance/secureAcceptanceIframeSummmary"/>
</isif>
<isif condition="${pdict.iscardinal }">
        <isinclude template="cardinal/songbird"/>
```

```
  </isif>
<isif condition="${klarnarequired}">
        <script src="${URLUtils.staticURL('/js/cybersource-custom.js')}"></script>
  </isif>
</isdecorate>
```

Template - cart.isml

1. **Add if condition to handle PlaceOrder error on cart page inside cart-banner**

```
<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>
```

2. **Update below code to apply coupon on cart page inside **

```
<iselseif condition="${pdict.CouponStatus != null && pdict.CouponStatus.error}">
                                        <div class="error">
                                            ${Resource.msgf("cart.APPLIED", "checkout", "",
pdict.CurrentForms.cart.couponCode.htmlValue)}
                                        </div>
                                    </isif>
                                </div>
```

Resources – form.properties

- **Add year values above year year.2022=2022**

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

Controller- common.js

Update validatePaymentInstruments function

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

```
// In case of method CREDIT_CARD, check payment cards
```

```
if (PaymentInstrument.METHOD_CREDIT_CARD.equals(paymentInstrument.paymentMethod)) {
// Gets payment card.
var card = PaymentMgr.getPaymentCard(paymentInstrument.creditCardType);

// Checks whether payment card is still applicable.
if (card && cards.contains(card) && !paymentInstrument.isCreditCardExpired()) {
continue;
            }
        }
```

Merchant Defined Data (MDD) Changes

In order to use Merchant defined data fields, merchant has to customize the below files to send merchant defined data in authorization request.

- CCAuthRequest() method of Cardfacade.ds file
- addCCAuthRequestInfo() method of libCyberSource.ds file

Merchant has to create and populate these objects and include in any of the authorization request. merchantDefinedData_mddField_1 to 100 request fields could be used to pass the information.

Credit Card Auth

Form - creditcard.xml

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

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

2. **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

Update code to mask ccNumber inside if condition, also retrieve subscription token of saved card to be used further:

```
<isscript>
            var ccNumber;
            if('maskedFourDigit' in pdict.SelectedCreditCard.custom &&
!empty(pdict.SelectedCreditCard.custom.maskedFourDigit)){
                        ccNumber = pdict.SelectedCreditCard.custom.maskedFourDigit;
                } else {
                        ccNumber = pdict.SelectedCreditCard.maskedCreditCardNumber;
                }
            var cc = {
                maskedNumber:ccNumber,
                holder:pdict.SelectedCreditCard.creditCardHolder,
                type:pdict.SelectedCreditCard.creditCardType,
```

```
                    expirationMonth:pdict.SelectedCreditCard.creditCardExpirationMonth,
                    expirationYear:pdict.SelectedCreditCard.creditCardExpirationYear,
                    selectedCardID:pdict.SelectedCreditCard.UUID

        }
        var json = JSON.stringify(cc);
</isscript>
```

Template - minicreditcard.isml

Add condition to map credit card number with four digit mask card number

```
<isscript>
        var ccType, ccNumber, ccMonth, ccYear, ccOwner;

        if (pdict.card) {
                ccType = pdict.card.creditCardType;
                if('maskedFourDigit' in pdict.card.custom && !empty(pdict.card.custom.maskedFourDigit)){
                        ccNumber = pdict.card.custom.maskedFourDigit;
                } else {
                        ccNumber = pdict.card.maskedCreditCardNumber;
                }

                ccMonth = pdict.card.creditCardExpirationMonth;
                ccYear = pdict.card.creditCardExpirationYear;
                ccOwner = pdict.card.creditCardHolder;
        }
</isscript>
```

Script - Resource.ds

Update ResourceHelper.getPreferences

```
COOKIE_HINT: (cookieHintAsset && cookieHintAsset.online) || false,
CHECK_TLS: Site.getCurrent().getCustomPreferenceValue('checkTLS'),
TOKENIZATION_ENABLED: (Site.getCurrent().getCustomPreferenceValue('CsTokenizationEnable') == 'YES')? true : false
```

Controller–COBilling.js

Update initCreditCardList function

Update function to migrate old card based on current paymentIntstrument inside customer authenticated if condition

```
if (customer.authenticated) {
var profile = app.getModel('Profile').get();

        var migrateCard = require('int_cybersource/cartridge/scripts/helper/migrateOldCardToken');
        migrateCard.MigrateOldCardToken(customer.profile.wallet.paymentInstruments);

if (profile) {
        applicableCreditCards = profile.validateWalletPaymentInstruments(countryCode,
paymentAmount.getValue()).ValidPaymentInstruments;
    }
  }
```

Controller - Hooks.json

Replace hook entry for CYBERSOURCE_CREDIT
[Note: Please delete CYBERSOURCE_CREDIT.js from <storefront controller cartridge>/cartridge/script/payment/processor to process the file present in CyberSource cartridge]

```
{
                    "name": "app.payment.processor.CYBERSOURCE_CREDIT",
                    "script":
"./../../../int_cybersource_controllers/cartridge/scripts/payment/processor/CYBERSOURCE_C
REDIT"
            },
```

Tax Service

Script - calculate.js

Call calculateTaxes function of cybersource by adding below line after basket.updateTotals()

```
calculateProductPrices(basket);

    // ===================================================
    // =====          CALCULATE TAX                  =====
    // ===================================================
    if (dw.order.TaxMgr.taxationPolicy == dw.order.TaxMgr.TAX_POLICY_NET
       && dw.system.Site.getCurrent().getCustomPreferenceValue('CsEnableTaxation')) {

       require('int_cybersource/cartridge/scripts/tax/adaptor/TaxAdaptor').CalculateTaxes(
basket);
       basket.updateTotals();
    }else{
       calculateTax(basket);
         basket.updateTotals();
    }
    // ===================================================
    // =====             DONE                        =====
    // ===================================================
```

Controller Cartridge – Script OrderModel.js

Update placeOrder function

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

```
function placeOrder(order) {
var placeOrderStatus = OrderMgr.placeOrder(order);
if (placeOrderStatus === Status.ERROR) {
            session.custom.SkipTaxCalculation=false;
      OrderMgr.failOrder(order);
throw new Error('Failed to place order.');
  }
  order.setConfirmationStatus(Order.CONFIRMATION_STATUS_CONFIRMED);
  order.setExportStatus(Order.EXPORT_STATUS_READY);
```

```
}
```

Controller -Cart.js

Update submitForm function

UpdatedeleteProductsection by clear cartstatestring

```
'deleteProduct': function (formgroup) {
        Transaction.wrap(function () {
            cart.removeProductLineItem(formgroup.getTriggeredAction().object);
session.custom.cartStateString = null;
        });

return {
            cart: cart
        };
      },
```

Controller – COShipping.js

Update "updateShippingMethodList"function

1. **Set session variable session.custom.SkipTaxCalculation=true; inside for loop and before cart.Calculate()**

```
            applicableShippingMethods = cart.getApplicableShippingMethods(address);
            shippingCosts = new HashMap();
            currentShippingMethod = cart.getDefaultShipment().getShippingMethod() ||
        ShippingMgr.getDefaultShippingMethod();

        // Transaction controls are for fine tuning the performance of the data base interactions when calculating
        shipping methods
            Transaction.begin();

        for (i = 0; i < applicableShippingMethods.length; i++) {
            method = applicableShippingMethods[i];

            cart.updateShipmentShippingMethod(cart.getDefaultShipment().getID(), method.getID(), method,
        applicableShippingMethods);
        session.custom.SkipTaxCalculation=true;
            cart.calculate();
            shippingCosts.put(method.getID(), cart.preCalculateShipping(method));
    }
```

Controller – COPlaceOrder.js

Update "clearforms" function

Add below snippet at end of function

```
session.custom.cartStateString=null;
```

Update "start" function

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

session.custom.SkipTaxCalculation=false;

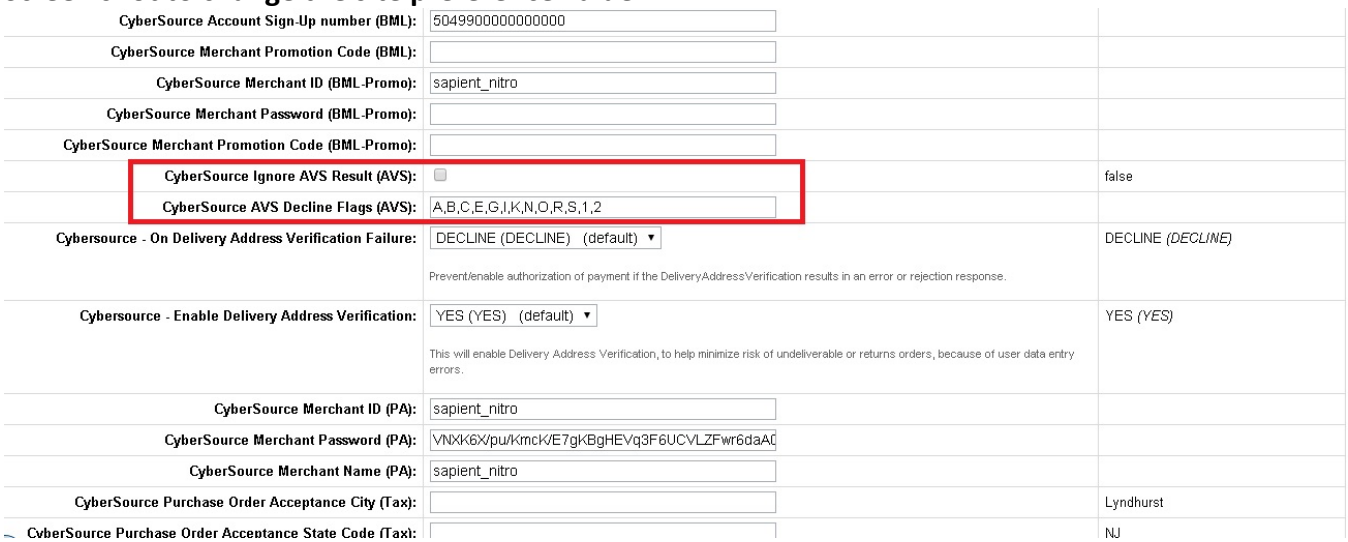
```
if (handlePaymentsResult.error) {
    session.custom.SkipTaxCalculation=false;
    return Transaction.wrap(function () {
        OrderMgr.failOrder(order);
        return {
            error: true,
            PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
        };
    });

}else if(handlePaymentsResult.returnToPage){
    app.getView({
            Order : handlePaymentsResult.order
        }).render('checkout/summary/summary');
    return {};
 }else if(handlePaymentsResult.redirection){
     response.redirect(handlePaymentsResult.redirectionURL);
     return {};
 }else if(handlePaymentsResult.carterror){
     app.getController('Cart').Show();
     return {};
 }else if(handlePaymentsResult.intermediate){
     app.getView({
         alipayReturnUrl : handlePaymentsResult.alipayReturnUrl
        }).render(handlePaymentsResult.renderViewPath);
    return {};
 } else if(handlePaymentsResult.intermediateSA){
     app.getView({
         Data:handlePaymentsResult.data, FormAction:handlePaymentsResult.formAction
        }).render(handlePaymentsResult.renderViewPath);
    return {};
 }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 {
```

Address Verification Service

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

User can change the site preference value by following Merchant Tools > Site Preferences > Custom

Site Preferences > 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.
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:



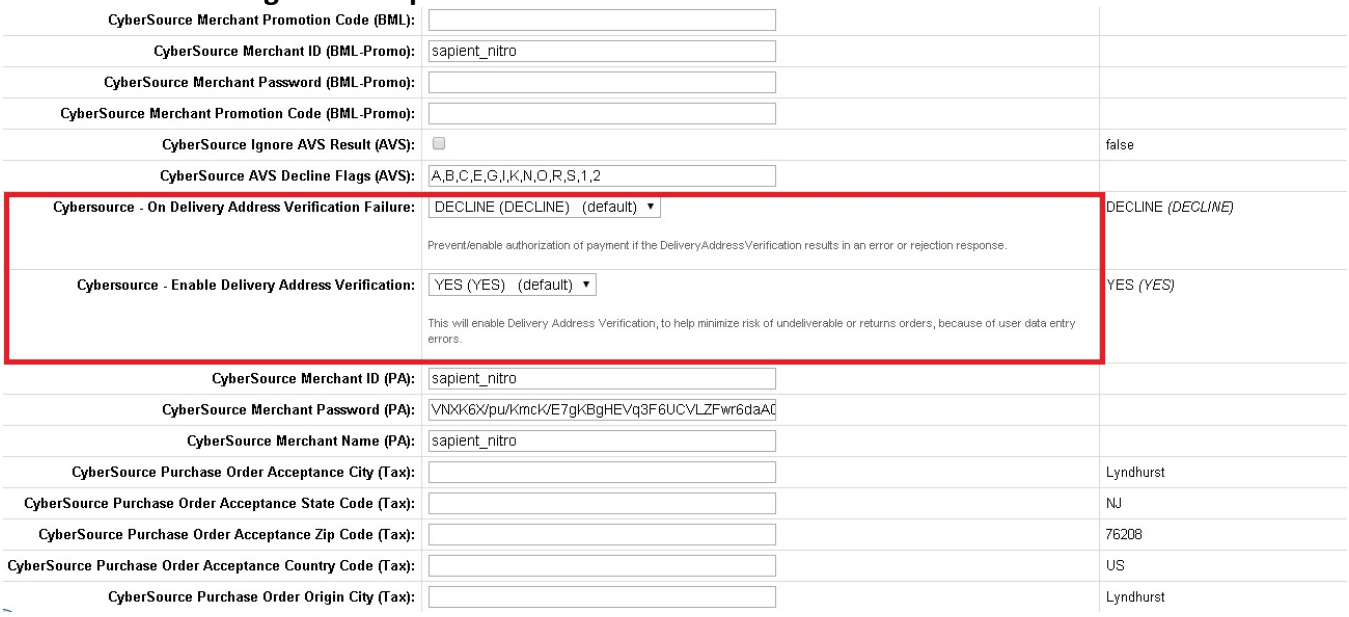
Delivery Address Validation Service

Provide Site Preference values for 2 DAV-related business rules:
User can change the site preference value by following Merchant Tools > Site Preferences > Custom Site Preferences > 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.

Screen shot to change the site preference value:



Payer Authentication Service

Controller - COSummary.js

Update submit Function

Updte function to handle Payer auth redirection

```
function submit() {
    // Calls the COPlaceOrder controller that does the place order action and any payment authorization.
    // COPlaceOrder returns a JSON object with an order_created key and a boolean value if the order was created
successfully.
    // If the order creation failed, it returns a JSON object with an error key and a boolean value.
    var cart = Cart.get();
var DFReferenceID = request.httpParameterMap.DFReferenceId.stringValue;
            session.privacy.DFReferenceID = DFReferenceID;
    var placeOrderResult = app.getController('COPlaceOrder').Start();
    if (placeOrderResult.error) {
        start({
            PlaceOrderError: placeOrderResult.PlaceOrderError
        });
    } else if (placeOrderResult.order_created) {
        showConfirmation(placeOrderResult.Order);
```

```
    }else if(placeOrderResult.process3DRedirection){
            var jwtUtil = require('int_cybersource/cartridge/scripts/cardinal/JWTBuilder');
             var cardinalUtil = require('int_cybersource/cartridge/scripts/cardinal/CardinalUtils');
            var creditCardForm = app.getForm('billing.paymentMethods.creditCard');
             var OrderObject = cardinalUtil.getOrderObject(cart,creditCardForm);
var orderdetailsObject =
cardinalUtil.getOrderDetailsObject(placeOrderResult.Order,placeOrderResult.authenticationTransactionID);
              OrderObject.setOrderDetails(orderdetailsObject);
        var jwtToken = jwtUtil.generateTokenWithKey(OrderObject);

        var orderstring = JSON.stringify(OrderObject);

            app.getView({Order: placeOrderResult.Order,
                        AcsURL:placeOrderResult.AcsURL,
                        PAReq:placeOrderResult.PAReq,
                        PAXID: placeOrderResult.PAXID,
                        authenticationTransactionID : placeOrderResult.authenticationTransactionID,
                        jwtToken:jwtToken,
                        orderstring :orderstring
            }).render('cart/cardinalpayerauthentication');
  }
}
```

Update start Function

Update start function to send jwt and order object

```
function start(context) {
    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');
    var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
    if (!COBilling.ValidatePayment(cart)) {
        COBilling.Start();
        return;
    } else {
        Transaction.wrap(function () {
            cart.calculate();
        });
         Transaction.wrap(function () {
           if (!cart.calculatePaymentTransactionTotal()) {
                COBilling.Start();
             }
         });
```

```
            COBilling.Start();
        }
    });
var order = "";
    var jwtToken ="";
    var iscardinal = false;
var selectedPaymentMethod = cart.getPaymentInstruments()[0].paymentMethod;
if(selectedPaymentMethod.equals(CybersourceConstants.METHOD_CREDIT_CARD) ||
selectedPaymentMethod.equals(CybersourceConstants.METHOD_SA_SILENTPOST)
                    || selectedPaymentMethod.equals(CybersourceConstants.METHOD_VISA_CHECKOUT))
{
            var jwtUtil = require('int_cybersource/cartridge/scripts/cardinal/JWTBuilder');
        var cardinalUtil = require('int_cybersource/cartridge/scripts/cardinal/CardinalUtils');
        jwtToken = jwtUtil.generateTokenWithKey();
        var creditCardForm = app.getForm('billing.paymentMethods.creditCard');
            var OrderObject = cardinalUtil.getOrderObject(cart,creditCardForm);
        order = JSON.stringify(OrderObject);
        iscardinal = true;
    }

    var pageMeta = require('~/cartridge/scripts/meta');
    var viewContext = require('app_storefront_core/cartridge/scripts/common/extend').immutable(context, {
        Basket: cart.object,
        jwtToken: jwtToken,
        order : order,
        iscardinal : iscardinal
    });
    pageMeta.update({pageTitle: Resource.msg('summary.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')});
    app.getView(viewContext).render('checkout/summary/summary');}}
```

And update base cartridge with below code.

Update the base cartridge file name app_storefront_core/cartridge/js/pages/checkout/billing.js

```
cctoken : cctoken,
ccnumber : ccnumber,
```

```
cvn : cvn,
month : month,
expyear : expyear,
cctype : cctype,
format : 'ajax'
```

Update the base cartridge file name: app_storefront_core/cartridge/static/default/js/app.js

```
cctoken : cctoken,
ccnumber : ccnumber,
cvn : cvn,
month : month,
expyear : expyear,
cctype : cctype,
format : 'ajax'
```

Update base cartridge file
app_storefront_core/cartridge/templates/default/account/payment/paymentinstrumentdetails.isml

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

Payment Tokenization Service

My Account - Template - paymentinstrumentdetails.isml

1. **Include the following code block just after the <h1> tag to display the Subscription Error Message message**

```
<h1>${Resource.msg('account.paymentinstrumentlist.addcard', 'account', null)}</h1>
<isif condition="${pdict.SubscriptionError != null}">
        <div class="error-form">
            ${Resource.msg('account.subscription','cybersource',null)}
        </div>
</isif>
```

2. **Include the below code**

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

3. **Include the below code right after <isdynamicform> form object to add Billing Address Fields**

```
<isdynamicform formobject="${pdict.CurrentForms.paymentinstruments.creditcards.newcreditcard.expiration}"
formdata="${currentCountry.dynamicForms.expirationInfo}"/>

                        <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

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

```
 <h1>${Resource.msg('account.paymentinstrumentlist.header','account',null)}</h1>
<isif condition="${pdict.SubscriptionError != null}">
        <div class="error-form">
           ${Resource.msg('paymentinstrumentlist.deletesubscription','cybersource',null)}
        </div>
</isif>
```

Rate Limit on My Account Page

Add below lines in paymentinstrumentlist.isml - Template

```
<isif condition="${pdict.savedCardlimitReached}">
    <div class="error-form">
        ${Resource.msg('error.message.addcard.fail','cybersource',null)}
    </div>
    </isif>
```

My Account - Controller - PaymentInstruments.js

Update "list" function

Update as below to handle Subscription Errorand migrate card token

```
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;
  }

var migrateCard = require('int_cybersource/cartridge/scripts/helper/migrateOldCardToken');
  migrateCard.MigrateOldCardToken(paymentInstruments);

  app.getView({
PaymentInstruments: paymentInstruments,
    SubscriptionError : SubscriptionError
  }).render('account/payment/paymentinstrumentlist');
return;
}
```

Update "Add" function

Update as below to handle SubscriptionError

```
function add(clearForm, subscriptionError) {
var paymentForm = app.getForm('paymentinstruments');

if (clearForm !== false) {
    paymentForm.clear();
  }
  paymentForm.get('creditcards.newcreditcard.type').setOptions(dw.order.PaymentMgr
    .getPaymentMethod(dw.order.PaymentInstrument.METHOD_CREDIT_CARD).activePaymentCards.iterator());

  app.getView({
ContinueURL: URLUtils.https('PaymentInstruments-PaymentForm'),
    SubscriptionError: subscriptionError
  }).render('account/payment/paymentinstrumentdetails');
}
```

Update "handlePaymentForm" function

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

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

Update "create" function

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

```
function create() {
            var SubscriptionError;
if (!verifyCreditCard()) {
            return {
                        error: true,
          SubscriptionError: SubscriptionError
            };
   }
var subscriptionID;
var enableTokenization : String =
dw.system.Site.getCurrent().getCustomPreferenceValue("CsTokenizationEnable").value;
            if (enableTokenization.equals('YES')) {
            var Cybersource_Subscription = require('int_cybersource_controllers/cartridge/scripts/Cybersource');
                        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").value;
                                    if (enableTokenization.equals('YES') && !empty(action.object.UUID)) {
                        var Cybersource_Subscription = require('int_cybersource_controllers/cartridge/scripts/Cybersource')
                        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'));
   }
}
```

Rate Limit on My Account Page

Replace handlePaymentForm() in PaymentInstruments.js – Controller

```
function handlePaymentForm() {
    var paymentForm = app.getForm('paymentinstruments');
    var profile = customer.getProfile();
    var currentTime= new Date();
    paymentForm.handleAction({
        create: function () {
            var Site = require('dw/system/Site');
            var Logger = require('dw/system/Logger');
            var cyberSourceHelper =
require('int_cybersource/cartridge/scripts/cybersource/libCybersource').getCybersourceHelp
er();
```

```
            var LimitSavedCardRateEnabled =
!empty(cyberSourceHelper.getLimitSavedCardRate())?
cyberSourceHelper.getLimitSavedCardRate() : false;

            if (!LimitSavedCardRateEnabled) {
                cardCreate();
            } else {
                    try{
                        var SavedCardLimitTimeFrame =
!empty(cyberSourceHelper.getSavedCardLimitTimeFrame())?
cyberSourceHelper.getSavedCardLimitTimeFrame() : 0;
                        var SavedCardLimitCount =
!empty(cyberSourceHelper.getSavedCardLimitFrame())?
cyberSourceHelper.getSavedCardLimitFrame() : 0;

                        if ('savedCCRateLookBack' in profile.custom &&
!empty(profile.custom.savedCCRateLookBack)) {
                            var customerTime =  profile.custom.savedCCRateLookBack;
                            var currentTime = new Date();
                            var difference = new Date().setTime(Math.abs(currentTime-
customerTime));
                            var differenceInSec = Math.floor(difference/1000);
                            if ( differenceInSec <  SavedCardLimitTimeFrame * 60 * 60) {
                                if ('savedCCRateCount' in profile.custom &&
(profile.custom.savedCCRateCount < SavedCardLimitCount)) {
                                    cardCreate();
                                    Transaction.wrap(function() {
                                        profile.custom.savedCCRateCount =
profile.custom.savedCCRateCount + 1;
                                    });
                                } else {
                                    response.redirect(URLUtils.https('PaymentInstruments-
List' , 'carderror' , true));
                                }

                            } else {
                                cardCreate();
                                resetLookBackDateandCount(profile);
                            }
                        } else {
                            cardCreate();
                            resetLookBackDateandCount(profile);
                        }
                    } catch(e) {
                        Logger.error("Error Processed while adding card: ",e.message);
                    }
            }
```

```
        },
        error: function () {
            add(false);
        }
    });
}


function cardCreate() {
    var createResult = create();
    if (createResult.error) {
           add(false, createResult.SubscriptionError);
           return;
    } else {
        response.redirect(URLUtils.https('PaymentInstruments-List'));
    }
}
```

Klarna

Controller - hooks.json

- **Add a hook for payment processor as KLARNA_CREDIT at the end of hooks.json in cartridge app_storefront_controllers**

```
        ,
                {
                        "name": "app.payment.processor.KLARNA_CREDIT",
                        "script":
"./../../../int_cybersource_controllers/cartridge/scripts/payment/processor/KLARNA_CREDIT"
}
```

COBilling.js

- Update save function inside billing function to handle the error returned by Klarna session service.

```
save: function () {
var cart = app.getModel('Cart').get();
var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
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.equals(CybersourceConstants.KLARNA_PAYMENT_METHOD)){
            returnToForm(cart,{KlarnaSessionError: handlePaymentSelection(cart).KlarnaSessionError});
            } else {
            returnToForm(cart);
            }
        } else {

if (customer.authenticated && app.getForm('billing').object.billingAddress.addToAddressBook.value) {
```

```
            app.getModel('Profile').get(customer.profile).addAddressToAddressBook(cart.getBillingAddress());
        }

// Mark step as fulfilled
        app.getForm('billing').object.fulfilled.value = true;

// A successful billing page will jump to the next checkout step.
        app.getController('COSummary').Start();
return;
    }
```

billing.isml

- Add a condition to handle error returned by session service

```
<iscomment>
    This template visualizes the billing step of both checkout scenarios.
    It provides selecting a payment method, entering gift certificates and
specifying a separate billing address.
    Depending on the checkout scenario (single or multi shipping) it is
either the second or third checkout step.
</iscomment>
        <isif condition="${!empty(pdict.KlarnaSessionError)}">
                <div class="error-form">${Resource.msg(pdict.KlarnaSessionError.code,'checkout',null)}</div>
        </isif>
<iscomment>Report this checkout step</iscomment>
<isreportcheckoutcheckoutstep="4"checkoutname="${'Billing'}"/>
```

htmlhead.isml

- Add a place holder to load Klarna JS

Line 9 - Line 20

```
<iscomment>See https://github.com/h5bp/html5-boilerplate/blob/5.2.0/dist/doc/html.md#x-ua-
compatible</iscomment>
<meta http-equiv="x-ua-compatible" content="ie=edge">

<iscomment>See https://github.com/h5bp/html5-boilerplate/blob/5.2.0/dist/doc/html.md#mobile-
viewport</iscomment>
<meta name="viewport" content="width=device-width, initial-scale=1">
<isscript>
        var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
</isscript>
<script type="text/javascript">
 WebFontConfig = {
  google: { families: [ 'Lato:100,300,700,100italic,300italic:latin', 'Crete+Round:400,400italic:latin' ] }
 };
 (function() {
```

Line 78 – Line 83

```
<iscomment>Visa Checkout clickjacking prevention</iscomment>
<isinclude template="visacheckout/clickjackingPrevent.isml" />
<isif condition="${'klarnaJSAPIPath' in dw.system.Site.current.preferences.custom &&
!empty(dw.system.Site.current.preferences.custom.klarnaJSAPIPath)
```

```
                                        &&
dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.KLARNA_PAYMENT_METHOD).isActive()}">
        <script src="${dw.system.Site.current.preferences.custom.klarnaJSAPIPath}" async></script>
</isif>
```

summary.isml

- Changes have been made in this file to load Klarna widget on summary page and other conditions to display Place Order and Edit button for other payment methods except Klarna.

Please refer to the changes mentioned under custom code – generic section- > summary.isml

Resource.ds

Include the following coloured line changes in the file.

```
rateLimiterReset        : URLUtils.url('RateLimiter-HideCaptcha').toString(),
    csrffailed          : URLUtils.url('CSRF-Failed').toString(),
    silentpost          : URLUtils.https('CYBSecureAcceptance-GetRequestDataForSilentPost').toString(),
    klarnaupdate        : URLUtils.https('CYBKlarna-UpdateSession').toString()
```

Bank Transfer

Controller - hooks.json

- **Add a hook for payment processor as KLARNA_CREDIT at the end of hooks.json in cartridge app_storefront_controllers**

```
        ,
                {
                        "name": "app.payment.processor.BANK_TRANSFER",
                        "script":
"./../../../int_cybersource_controllers/cartridge/scripts/payment/processor/BANK_TRANSFER"
                }
```

billing.xml

- Add form fields for BIC and Bank List

```
<group formid="paymentMethods">

        <!--
                the selected payment method, e.g. "CREDIT_CARD" or "PayPal", this field is
                used to transport the payment method selection; validations then can be
                made on the proper form group which defines the actual payment method attributes
        -->


        <field formid="bankListSelection" label="payment.bankselection" type="string" mandatory="false"
                        missing-error="payment.bankselectionerorr" value-
error="payment.bankselectionerorr" />

        <field formid="bicNumber" label="payment.bicnumber" type="string" mandatory="false"
                        missing-error="payment.bicnumbererror" value-error="payment.bicnumbererror" />

        <field formid="selectedPaymentMethodID" type="string" default-value="CREDIT_CARD">
                <options optionid-binding="ID" value-binding="ID" label-binding="name"/>
        </field>
```

```
        <!-- list of available credit cards to select from -->
        <list formid="creditCardList">

            <!-- action for actually selecting the credit card -->
            <action formid="useThisCreditCard" valid-form="false"/>

        </list>


        <!-- fields for CreditCard selection -->
        <include formid="creditCard" name="creditcard"/>

        <!-- fields for BML selection -->
        <include formid="bml" name="bml"/>

    </group>
```

paymentmethods.isml

- Add condition to handle bank transfer payment method on billing page present at checkout\billing\ path

Changes are aleady covered under custom code > generic section-> paymentmethods.isml

forms.properties

- Add resource bundle value

```
payment.bankselection=Select Bank
payment.bankselectionerorr=Please Select Bank
payment.bicnumber=BIC Number
payment.bicnumbererror=Please Enter BIC number
```

Alipay Authorization

ValidatePaymentInstruments.ds

- **Replace the GIFT_CERTIFICATE payment instrument check**

Add import

```
importPackage( dw.web );

// ignore gift certificate payment instruments
if(PaymentInstrument.METHOD_GIFT_CERTIFICATE.equals(pi.paymentMethod) ||
Resource.msg("paymentmethodname.alipay", "cybersource", null).equals(pi.paymentMethod))
    {
```

Controller – Hooks.json

- **Add a hook for payment processor as CYBERSOURCE_ALIPAY at the end of hooks.json in cartridge app_storefront_controllers**

```
        ,
```

```
                    {
                        "name": "app.payment.processor.CYBERSOURCE_ALIPAY",
                        "script":
            "./../../../int_cybersource_controllers/cartridge/scripts/payment/processor/CYBERSOURCE_ALIPAY"
}
```

COPlaceOrder.js

- [Note: Below snipped is for reference purpose only, changes are aleady covered under custom code > generic section ->COPlaceOrder.js].
- Note : If Alipay payment fails due to one of the fields(alipayReturnUrl) in the authorization request is invalid, then one should check should check for length of the alipayReturnUrl field . It should not be more than 200 characters. To maintain url length less than 200, site url excluding controller name and method should be less than 120 characters.

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

}else if(handlePaymentsResult.redirection){
             response.redirect(handlePaymentsResult.redirectionURL);
            return {};
    }
else if(handlePaymentsResult.intermediate){
        app.getView({
             alipayReturnUrl : handlePaymentsResult.alipayReturnUrl
                 }).render(handlePaymentsResult.renderViewPath);
                      return {};
    }
```

WeChat Pay

COBilling.js

- Update save function inside billing function to handle the Wechat Pay session.

```
save: function () {
var cart = app.getModel('Cart').get();
var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
var
CybersourceHelper=require('int_cybersource/cartridge/scripts/cybersource/libCybersource').getCybersourceHelper();

var handlePaymentSelectionResult = handlePaymentSelection(cart);
if (!resetPaymentForms() || !validateBilling() || !handleBillingAddress(cart) || // Performs validation steps, based upon
the entered billing address
// and address options.
handlePaymentSelectionResult.error) {// Performs payment method specific checks, such as credit card verification.
if(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(CybersourceConstants.KLARN
A_PAYMENT_METHOD)){
            returnToForm(cart,{KlarnaSessionError: handlePaymentSelectionResult.KlarnaSessionError});
            } else {
            returnToForm(cart);
            }
        } else {
```

```
if (customer.authenticated && app.getForm('billing').object.billingAddress.addToAddressBook.value) {
            app.getModel('Profile').get(customer.profile).addAddressToAddressBook(cart.getBillingAddress());
        }

// Mark step as fulfilled
        app.getForm('billing').object.fulfilled.value = true;
// Redirecting to Wechat condition
            if
(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(CybersourceConstants.WECHA
T_PAYMENT_METHOD)
                                && !empty(handlePaymentSelectionResult.WechatMerchantURL)) {
                    app.getView({
                                formAction : handlePaymentSelectionResult.WechatMerchantURL,
                                noOfCalls : CybersourceHelper.getNumofCheckStatusCalls()!=null ?
CybersourceHelper.getNumofCheckStatusCalls() : 5 ,
                                serviceCallInterval : CybersourceHelper.getServiceCallInterval()!=null ?
CybersourceHelper.getServiceCallInterval() : 10
                    }).render('wechat/wechatRedirect');
                    return;
                }
// A successful billing page will jump to the next checkout step.
        app.getController('COSummary').Start();
return;
    }
```

Controller – hooks.json

- **Add a hook for payment processor as CYBERSOURCE_WECHAT at the end of hooks.json in cartridge app_storefront_controllers**

```
                {
                        "name": "app.payment.processor.CYBERSOURCE_WECHAT",
                        "script":
"./../../../int_cybersource_controllers/cartridge/scripts/payment/processor/CYBERSOURCE_WECHAT"
                }
```

PayPal Express & PayPal Billing Agreement

footer_ui.isml

Place below lines of code in footer_ui.isml at end of file

```
<isscript>
        var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
</isscript>
```

```
<isif condition="${dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.METHOD_PAYPAL).isActive() &&
dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}">
        <scriptsrc="https://www.paypalobjects.com/api/checkout.js"></script>
</isif>
<scriptsrc="${URLUtils.staticURL('/js/cybersource-custom.js')}"></script>
```

Minicart.isml

Include script module after util/module

```
<isincludetemplate="util/modules"/>
<isscript>
        var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
</isscript>
```

Add below code after class="button mini-cart-link-cart" anchor tag

```
<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="${URLUtils.https('CYBPaypal-
SessionCallback')}" method="post" name="${pdict.CurrentForms.cart.dynamicHtmlName}" id="checkout-form">
                            <fieldset>
                                <isif
condition="${dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.METHOD_PAYPAL).isActive() &&
dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}">
                                    <isif
condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('payPalBillingAgreements') &&
!empty(pdict.CurrentCustomer.profile) && !empty(pdict.CurrentCustomer.profile.custom.billingAgreementID)}">
                                        <input type="image"
src="https://www.paypal.com/en_US/i/btn/btn_xpressCheckout.gif" alt="Paypal Express" />
                                    <iselse>
                                        <div class="paypal-button-container-mini"></div>
                                    </isif>
                                </isif>
                            </fieldset>
                        </form>
```

Cart.isml

Add cubersource constants after API include section

```
<isincludetemplate="util/reporting/ReportBasket.isml"/>
<isscript>
        var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
</isscript>
```

Add below lines of inside <div class="cart-actions > and

```
<div class="cart-actions cart-actions-top">
```

```
                    <iscomment>continue shop url is a non-secure but checkout needs a secure and that is why
separate forms!</iscomment>
                    <form class="cart-action-
checkout"action="${URLUtils.continueURL()}"method="post"name="${pdict.CurrentForms.cart.dynamicHtmlName}"id="c
heckout-form">
                        <fieldset>
                            <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="${dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.METHOD_PAYPAL).isActive() &&
dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}">
                                    <isif
condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('payPalBillingAgreements') &&
!empty(pdict.CurrentCustomer.profile) && !empty(pdict.CurrentCustomer.profile.custom.billingAgreementID)}">
                                        <input type="image"
src="https://www.paypal.com/en_US/i/btn/btn_xpressCheckout.gif" alt="Paypal Express"
class="billingAgreementExpressCheckout"/>
                                    <iselse>
                                        <div class="paypal-button-container-
cart2"></div>
                                    </isif>
                                                                                    </isif>
```

```
<div class="cart-actions">

                    <iscomment>continue shop url is a non-secure but checkout needs a secure and that is why
separate forms!</iscomment>
                    <form class="cart-action-checkout"  action="${URLUtils.continueURL()}" method="post"
name="${pdict.CurrentForms.cart.dynamicHtmlName}" id="checkout-form">
                        <fieldset>
                            <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="${dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.METHOD_PAYPAL).isActive() &&
dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}">
                                    <isif
condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('payPalBillingAgreements') &&
!empty(pdict.CurrentCustomer.profile) && !empty(pdict.CurrentCustomer.profile.custom.billingAgreementID)}">
                                        <input type="image"
src="https://www.paypal.com/en_US/i/btn/btn_xpressCheckout.gif" alt="Paypal Express"
class="billingAgreementExpressCheckout"/>
                                    <iselse>
                                        <div class="paypal-button-container-
cart1"></div>
```

```
                                                    </isif>

                                                                                    </isif>
```

Resources.ds

Add below urls in urls json object under ResourceHelper.getUrls method.

```
,
paypalinitsession : URLUtils.url('CYBPaypal-InitiatePaypalExpress').toString(),
paypalcallback    : URLUtils.https('CYBPaypal-SessionCallback').toString(),
billingagreement :  URLUtils.https('CYBPaypal-BillingAgreement').toString(),
orderreview       : URLUtils.https('COSummary-Start').toString()
```

Add below preference in json object under ResourceHelper.getPreferences method

```
,
ISPAYPALENABLED : (dw.order.PaymentMgr.getPaymentMethod('PAYPAL').isActive()
&&Site.getCurrent().getCustomPreferenceValue('CsEnableExpressPaypal')?true:false)
```

Checkout.properties

Add billingagreement message for PayPal.

```
billing.selectcreditcard=SelectCreditCard
billing.billingagreement=CreateBillingAgreement
```

Paymentmethods.isml

Include cubersource constant at API include section

Changes are aleady covered under custom code > generic section-> paymentmethods.isml

PAYPAL_EXPRESS.js

Include Cybersource constants at API include section

```
var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
var CommonHelper = require(CybersourceConstants.CS_CORE_SCRIPT+'helper/CommonHelper');
```

Replace below code with the code in Handle method

```
function Handle(args) {
var cart = Cart.get(args.Basket);

   Transaction.wrap(function () {
CommonHelper.removeExistingPaymentInstruments(cart);
    var paymentInstrument = cart.createPaymentInstrument(CybersourceConstants.METHOD_PAYPAL,
cart.getNonGiftCertificateAmount());
});

return {success: true};
```

```
}
```

Replace below code with the code in Authorize method

```
function Authorize(args) {
        var paymentInstrument = args.PaymentInstrument;
var paymentProcessor =
PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();
var adapter = require(CybersourceConstants.PAYPAL_ADAPTOR);
  //Logic to determine if this is standard/custom Paypal order
         Transaction.wrap(function () {
             paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;
         });
        var paymentResponse = adapter.PaymentService(args.Order,paymentInstrument);

  if(paymentResponse.authorized)
        {
        return {authorized: true};
        }else if(paymentResponse.pending){
                return {review: true};
        }
        else{
                return {error: true};
                                                        }}
```

ValidatePaymentInstruments.ds

Add another condition in if statement at line 51

```
Add import
importPackage( dw.web );

if(PaymentInstrument.METHOD_GIFT_CERTIFICATE.equals(pi.paymentMethod) ||
Resource.msg("paymentmethodname.paypal", "cybersource", null).equals(pi.paymentMethod))
{
        continue;
}
```

PayPal Credit

Controller – PAYPAL_CREDIT.js

Include API at the top of file as below:

```
/* API Includes */
var Cart = require('~/cartridge/scripts/models/CartModel');
var PaymentMgr = require('dw/order/PaymentMgr');
```

```
var Transaction = require('dw/system/Transaction');
var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
var CommonHelper = require(CybersourceConstants.CS_CORE_SCRIPT+'helper/CommonHelper');
```

Update handle function for PayPal credit payment instrument

```
function Handle(args) {
        var cart = Cart.get(args.Basket);

        Transaction.wrap(function () {
                CommonHelper.removeExistingPaymentInstruments(cart);
          var paymentInstrument = cart.createPaymentInstrument(CybersourceConstants.METHOD_PAYPAL_CREDIT,
cart.getNonGiftCertificateAmount());
        });

        return {success: true};
}
```

Update authorize function for PayPal credit payment instrument

```
function Authorize(args) {
var paymentInstrument = args.PaymentInstrument;
        var paymentProcessor =
PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();
        var adapter = require(CybersourceConstants.PAYPAL_ADAPTOR);
        //Logic to determine if this is standard/custom Paypal order
        Transaction.wrap(function () {
            paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;
        });
        var paymentResponse = adapter.PaymentService(args.Order,paymentInstrument);

        if(paymentResponse.authorized)
        {
                return {authorized: true};
        }else if(paymentResponse.pending){
                return {review: true};
        }
        else{
                return {error: true};
    }
}
```

Paymentmethods.isml

Above mentioned steps for PayPal Express is also required for PayPal credit except minicart.isml and cart.isml. Only additional div for payment-method need to add in paymentmethods.isml

Changes are aleady covered under custom code > generic section-> paymentmethods.isml

Retail POS

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

```
/* API Includes */
var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
var guard = require(CybersourceConstants.GUARD);
/**
 * Authorizes a payment using a credit card. The payment is authorized by using the POS specific 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 AuthorizePOS(args) {
    var POSAdaptor = require(CybersourceConstants.CS_CORE_SCRIPT+'pos/adaptor/POSAdaptor');
    var result = POSAdaptor.InitiatePOSAuthRequest(args);
    if (result.success){
        return {authorized:true, posAuthResponse:result.serviceResponse};
    }
    return {error:true};
}
```

The above controller method AuthorizePOS should be integrated at EACreditCard-Authorize controller of DSS app. The track data, expiration date or account number should not be encrypted and may need to be decrypted prior to calling CYBPos -AuthorizePOS depending on the payment terminal used.

To make call for POS authentication, InitiatePOSAuthRequest method is defined where POS related data are being passed as input parameters..
The args contains variables based on POS terminal entry mode. Below are the use and description of variables. Assuming that args object contains the required values as follows:

POS terminal entry mode can be set in int_ocapi_ext/cartridge/scripts/actions/CaptureCreditCardDetails.ds as Shown below.

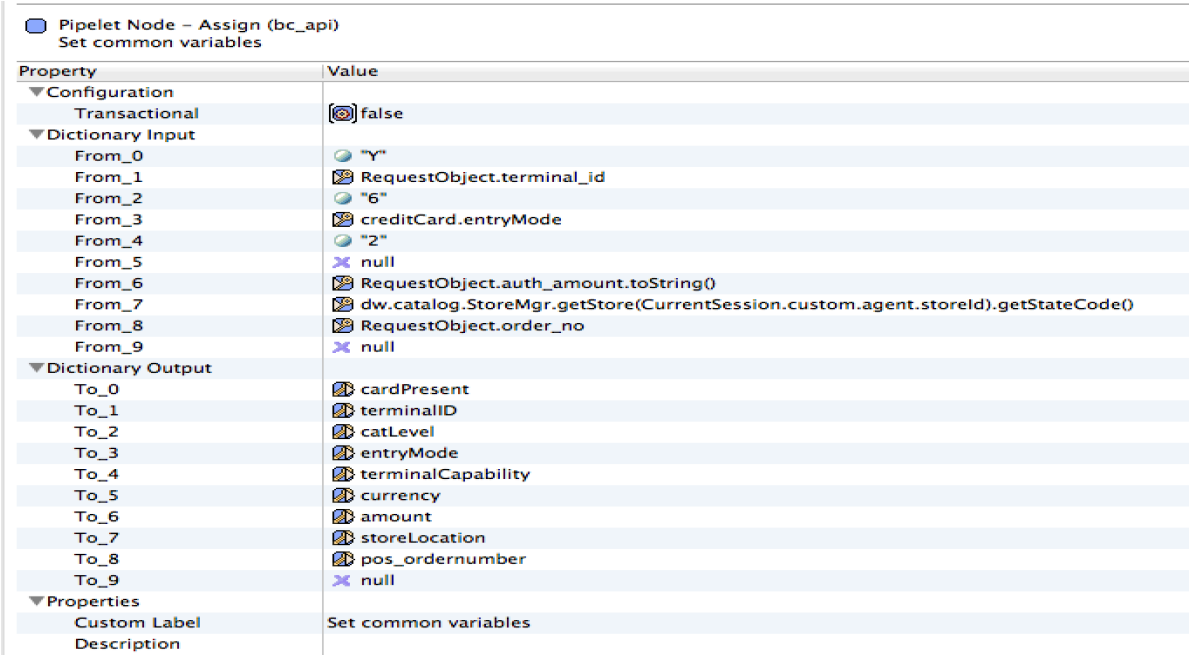
```
var expMonth : Number = 0;
var expYr : Number = 0;
var ccType : String = "";
var mode : String = "swiped";

if( empty(track1) && empty(track2) ) {
    name = dw.web.Resource.msg('carddetails.name', 'capturecreditcarddetails',
    ccNumber = args.AccountNumber;
    expMonth = args.ExpirationDate.substring(0,2);
    expYr = args.ExpirationDate.substring(3);
    mode = "keyed";
} else {
    //Begin
    if (empty(track1)) {
        track1 = "";
    }
    name = track1.substr(track1.indexOf("^", 0)+1, track1.indexOf("^", track1.i
    name = name.replace("/", " ");
    ccNumber= track2.substr(1, track2.indexOf("=", 0)-1);
    expMonth = new Number(track2.substr(track2.indexOf("=", 0)+3, 2));
    expYr = new Number(track2.substr(track2.indexOf("=", 0)+1, 2));
}

ccType = "MasterCard";
if (ccNumber.substring(0, 1) == checkForVisaStartNumber && ccNumber.length == v
    ccType = "Visa";
} else if (ccNumber.substring(0, 1) == checkForMasterCardStartNumber && ccNumbe
    ccType = "MasterCard";
} else if (ccNumber.substring(0, 1) == checkForAmexStartNumber && (ccNumber.len
    ccType = "Amex";
}

//Create Xredit Card data object
args.creditCard = {
    owner : name,
    num : ccNumber,
    type : ccType,
    month : expMonth,
    year : expYr,
    entryMode: mode
};
```

Below input fields will set the common variables for the transaction irrespective of entry mode used.

- **args.cardPresent**
- **args.entryMode**
- **args.catLevel**
- **args.terminalCapability**
- **args.terminalID**
- **args.amount**
- **args.currency**

- **args.storeLocation**
- **pos_ordernumber**

Example input variables from DSS:



Pipelet Node – Assign (bc_api)
Set common variables

Property	Value
Configuration	
Transactional	false
Dictionary Input	
From_0	"Y"
From_1	RequestObject.terminal_id
From_2	"6"
From_3	creditCard.entryMode
From_4	"2"
From_5	null
From_6	RequestObject.auth_amount.toString()
From_7	dw.catalog.StoreMgr.getStore(CurrentSession.custom.agent.storeId).getStateCode()
From_8	RequestObject.order_no
From_9	null
Dictionary Output	
To_0	cardPresent
To_1	terminalID
To_2	catLevel
To_3	entryMode
To_4	terminalCapability
To_5	currency
To_6	amount
To_7	storeLocation
To_8	pos_ordernumber
To_9	null
Properties	
Custom Label	Set common variables
Description	

If "keyed entry" mode is used on the POS terminal device.Below Input variable need to set:

- **args.accountNumber**
- **args.cardType**
- **args.cvnNumber**
- **args.expiryMonth**
- **args.expiryYear**

Example input variables from DSS:

Pipelet Node – Assign (bc_api)
Set keyed variables

Property	Value
Configuration	
Transactional	false
Dictionary Input	
From_0	creditCard.num
From_1	creditCard.month
From_2	creditCard.year
From_3	null
From_4	null
From_5	null
From_6	null
From_7	null
From_8	null
From_9	null
Dictionary Output	
To_0	accountNumber
To_1	expiryMonth
To_2	expiryYear
To_3	cardType
To_4	cvnNumber
To_5	null
To_6	null
To_7	null
To_8	null
To_9	null
Properties	
Custom Label	Set keyed variables
Description	

If "swiped entry" mode is used on the POS terminal device.Below Input variable need to set:

- **args.trackData:**

Pipelet Node – Assign (bc_api)
Set swiped variables

Property	Value
▼Configuration	
Transactional	false
▼Dictionary Input	
From_0	creditCard.track1 + creditCard.track2
From_1	null
From_2	null
From_3	null
From_4	null
From_5	null
From_6	null
From_7	null
From_8	null
From_9	null
▼Dictionary Output	
To_0	trackData
To_1	null
To_2	null
To_3	null
To_4	null
To_5	null
To_6	null
To_7	null
To_8	null
To_9	null
▼Properties	
Custom Label	Set swiped variables
Description	

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 imported). 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	Required if entryMode=swiped.

		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^2012101976110000868000000?;4111111111111111=20121019761186800000? Track 1 – the track 1 data precedes the semicolon (;) Track 2 – the track 2 data follows the semicolon (;)	
7	currency	Currency used for order. For possible values refer ISO Standard Currency Codes	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

Apple Pay REST Interface Integration ways with Device/APP

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

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

Interface AS Service

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

Interface Direct Functions [when basket or order available]

a. This integration way is recommended when hook script has order or basket available along with other service required inputs. Also merchant site enabled “Limit Storefront Order” setting
b. The Hook script file having OCAPI hook defined call below functions directly and before calling also validate inputs are valid.
c. The function “**MobilePaymentAuthRequest**” is called when Payload is available
MobilePaymentFacade.MobilePaymentAuthRequest(jsonParam)
JsonParam will containlineItemCtnr : dw.order.LineItemCtnr, orderNo : String, IPAddress : String, encryptedPaymentData.

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

d. The function “**MobilePaymentAuthRequest**” is called when network token is available
MobilePaymentFacade.MobilePaymentAuthRequest(jsonParam)
jsonParam will contain lineItemCtnr : dw.order.LineItemCtnr, orderNo : String, IPAddress : String, cryptogram, networkToken, tokenExpirationMonth, tokenExpirationYear, cardType.

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

cardType	String

e. This function called to update the payment instrument with the service response PaymentInstrumentUtils.UpdatePaymentTransactionCardAuthorize(paymentInstrument, ServiceResponseObject: Object)

Parameter	Type
paymentInstrument	dw.order.PaymentInstrument
ServiceResponseObject	Object

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

Interface Functions [when required service request objects available]

a. This integration way is recommended when hook script has order or basket available in for of JSON instead of object along with other service required inputs. Also merchant site enabled "Limit Storefront Order" setting
b. Hook script to prepare CyberSource service related objects like billto, shipto, purchaseTotal etc.
c. The Hook script file having OCAPI hook defined call below functions and before calling also validate inputs are valid.
d. The function "**MobilePaymentAuthRequest**" is called when Payload is available
MobilePaymentFacade.MobilePaymentAuthRequest(jsonParam)
jsonParam will containbillTo, shipTo, purchaseObject, items, orderNo : String, IPAddress : String, encryptedPaymentData.

Parameter	Type
billTo	Cybersource_BillTo_Object
shipTo	Cybersource_ShipTo_Object
purchaseObject	Cybersource_PurchaseTotals_Object
Items	Cybersource_Item_Object
orderNo	String
IPAddress	String
encryptedPaymentData	String

e. The function "**MobilePaymentAuthRequest**" is called when Network Token is available
MobilePaymentFacade.MobilePaymentAuthRequest(jsonParam)
jsonParam will containbillTo, shipTo, purchaseObject, items, orderNo : String, IPAddress : String, cryptogram, networkToken, tokenExpirationMonth, tokenExpirationYear, cardType.

Parameter	Type
billTo	Cybersource_BillTo_Object
shipTo	Cybersource_ShipTo_Object
purchaseObject	Cybersource_PurchaseTotals_Object
Items	Cybersource_Item_Object

orderNo	String
IPAddress	String
Cryptogram	String
networkToken	String
tokenExpiration Month	String
tokenExpiration Year	String
cardType	String

f. This function called to update the payment instrument with the service response PaymentInstrumentUtils.UpdatePaymentTransactionCardAuthorize(paymentInstrument, ServiceResponseObject: Object)

Parameter	Type
paymentInstrument	dw.order.PaymentInstrument
ServiceResponse Object	Object

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

Android Pay REST Interface Integration ways with Device/APP

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

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

Interface AS Service

a. Using “Interface AS Service” has limitation that merchant site MUST disable “Limit Storefront Order” setting
b. Register interface in service initialization script file say “SoapServiceInit.ds”
c. Define above service end point as merchant site URL for “CYBAndroidPay -Authorize” in BM service configurations
d. Define user/password to be picked from site preferences “cybAndroidPayInterfaceUser”, “cybAndroidPayInterfacePassword” in service initialization script file say “SoapServiceInit.ds”

e. The Hook script file having OCAPI hook defined invoke service endpoint by passing required JSON input. (The JSON Input format defined in appropriate REST Interface section above in the document.)
f. Interpret the response received and display thank you page on success and order failure page on failure

Interface Direct Functions [when basket or order available]

g. This integration way is recommended when hook script has order or basket available along with other service required inputs. Also merchant site enabled "Limit Storefront Order" setting
h. The Hook script file having OCAPI hook defined call below functions directly and before calling also validate inputs are valid.
i. The function "**MobilePaymentAuthRequest**" is called when Payload is available **MobilePaymentFacade. MobilePaymentAuthRequest** (JSONParams).
JSONParam will contains dw.order.LineItemCtnr, orderNo , IPAddress, encryptedPaymentData

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

The function "**MobilePaymentAuthRequest**" is called when network token is available **MobilePaymentFacade. MobilePaymentAuthRequest** (**MobilePaymentAuthRequest** (JSONParams).
j. JSONParam will contains lineItemCtnr : dw.order.LineItemCtnr, orderNo : String, IPAddress : String, cryptogram, networkToken, tokenExpirationMonth, tokenExpirationYear, cardType

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

k. This function called to update the payment instrument with the service response PaymentInstrumentUtils.UpdatePaymentTransactionCardAuthorize(paymentInstrument, ServiceResponseObject: Object)

Parameter	Type
paymentInstrument	dw.order.PaymentInstrument
ServiceResponseObject	Object

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

Interface Functions [when required service request objects available]

h. This integration way is recommended when hook script has order or basket available in for of JSON instead of object along with other service required inputs. Also merchant site enabled "Limit Storefront Order" setting
i. Hook script to prepare CyberSource service related objects like billto, shipto, purchaseTotal etc.
j. The Hook script file having OCAPI hook defined call below functions and before calling also validate inputs are valid.
k. The function "**MobilePaymentFacade.MobilePaymentAuthReques**t" is called when Payload is available
MobilePaymentFacade.MobilePaymentAuthRequest (paymentAPIRequestParams)
paymentAPIRequestParamswill contain billTo, shipTo, purchaseObject, items, orderNo : String, IPAddress : String, encryptedPaymentData.

Parameter	Type
billTo	Cybersource_BillTo_Object
shipTo	Cybersource_ShipTo_Object
purchaseObject	Cybersource_PurchaseTotals_Object
Items	Cybersource_Item_Object
orderNo	String
IPAddress	String
encryptedPaymentData	String

l. The function "**MobilePaymentAuthReques**t" is called when Network Token is available
MobilePaymentFacade.MobilePaymentAuthRequest(paymentAPIRequestParams)
paymentAPIRequestParamswill contain billTo, shipTo, purchaseObject, items, orderNo : String, IPAddress : String, cryptogram, networkToken, tokenExpirationMonth, tokenExpirationYear, cardType.

Parameter	Type
billTo	Cybersource_BillTo_Object
shipTo	Cybersource_ShipTo_Object
purchaseObject	Cybersource_PurchaseTotals_Object
Items	Cybersource_Item_Object
orderNo	String
IPAddress	String
Cryptogram	String
networkToken	String
tokenExpirationMonth	String
tokenExpirationYear	String

cardType	String

m. This function called to update the payment instrument with the service response **PaymentInstrumentUtils.UpdatePaymentTransactionCardAuthorize**(paymentInstrument, ServiceResponseObject: Object)

Parameter	Type
paymentInstrument	dw.order.PaymentInstrument
ServiceResponseObject	Object

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

Visa Checkout

billing.js

1.) Update updatePaymentmethod function

Add a condition just afte selectedPaymentMethod condition to display or hide visa checkout button on selected payment methods as 'VISA_CHECKOUT'.

Sample code:

[Note: Below changes are covered in custom code > Generic section > billing.js, defined here for reference only]

```
if (paymentMethodID=="VISA_CHECKOUT") {
            $(".continue-place-order").hide();
            $(".visacheckoutbutton").show();
      }
      else {
            $(".continue-place-order").show();
            $(".visacheckoutbutton").hide();
     }
```

[note: this section updated in custom code section as to reduce redundancy]

```
function updatePaymentMethod(paymentMethodID) {
    var $paymentMethods = $('.payment-method');
    $paymentMethods.removeClass('payment-method-expanded');

    var $selectedPaymentMethod = $paymentMethods.filter('[data-method="' + paymentMethodID + '"]');
    if ($selectedPaymentMethod.length === 0) {
        $selectedPaymentMethod = $('[data-method="Custom"]');
    }
    if (paymentMethodID=="VISA_CHECKOUT") {
        $(".continue-place-order").hide();
        $(".visacheckoutbutton").show();
    }
    else {
        $(".continue-place-order").show();
        $(".visacheckoutbutton").hide();
    }
```

paymentmethods.isml

1.) **Add ondition for Visa Checkout error handling just after closing of </legend> block.**

[Note: Below snippet is for reference purpose only, changes are aleady covered under custom code > generic section ->COPlaceOrder.js]

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

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

billing.isml

- **Add class on "continue to place order" button that would be used in billing.js to hide or show button based on payment method selection below isbonusdiscountlineitem tag and set type as button**

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

- **Include Visa Checkout Button**

Add following div section after the form ends

```
<div class="visacheckoutbutton hide" style="text-align: center;">
<isinclude url="${URLUtils.url('CYBVisaCheckout-Button')}"/>
</div>
```

cart.isml

1.) Include Visa checkout button:
Add following lines above cart-recommendations div

```
<!--  BEGIN Visa Checkout code -->
                        <isif condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('cybVisaButtonOnCart')}">
                                <isif condition="${pdict.CurrentHttpParameterMap.visacheckout.value}">
                                        <isinclude url="${URLUtils.url('CYBVisaCheckout-
Button','visacheckout','pdict.CurrentHttpParameterMap.visacheckout.value')}"/>
                                <iselse>
                                        <isinclude url="${URLUtils.url('CYBVisaCheckout-Button')}"/>
                                </isif>
                        </isif>
<!--  END Visa Checkout code -->
```

minicart.isml

1.) Add following line in after </isapplepay> tag anddiv having id
before checkout button

```
        <!--  BEGIN Visa Checkout code -->
                                <isif
condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('cybVisaButtonOnCart')}">
                                        <isif condition="${empty(pdict.CurrentHttpParameterMap.visacheckout.value) ||
!pdict.CurrentHttpParameterMap.visacheckout.value}">
                                                <isinclude url="${URLUtils.url('CYBVisaCheckout-
Button','buttonsource','minicart')}"/>
                                        <iselse>
                                                <isinclude url="${URLUtils.url('CYBVisaCheckout-Button')}"/>
                                        </isif>
                                </isif>
                                <!--  END Visa Checkout code -->

                                <a class="mini-cart-link-checkout" href="${URLUtils.https('COCustomer-Start')}"
title="${Resource.msg('minicart.directcheckout','checkout',null)}">${Resource.msg('minicart.directcheckout','checkout',nu
ll)} &raquo;</a>
                        </div>
                </div>
```

footer_UI.isml

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

```
<iscomment>Visa Checkout launch</iscomment>
<isincludetemplate="visacheckout/launch.isml"/>
```

header.isml

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

```
<iscomment>INCLUDE: Mini-cart, do not cache</iscomment>
        <div id="mini-cart">
```

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

htmlhead.isml

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

```
<iscomment>Visa Checkout clickjacking prevention</iscomment>
<isinclude template="visacheckout/clickjackingPrevent.isml"/>
```

Controller - Cart.js

Update the Show() method with Visa checkout changes.

1.) Add following hightlighted line of code to show() function as shown in the snippet:

```
function show() {
var cartForm = app.getForm('cart');
  app.getForm('login').invalidate();

  cartForm.get('shipments').invalidate();
var VisaCheckout = require('int_cybersource/cartridge/scripts/visacheckout/helper/VisaCheckoutHelper');
var VInitFormattedString='';
var signature='';
    var result = VisaCheckout.Initialize();
if (result.success) {
    VInitFormattedString = result.VInitFormattedString;
    signature= result.signature;
  }
// TO handle the visa checkout click even on cart and billing page from mini cart
  session.custom.cyb_CurrentPage = "CybCart";

  app.getView('Cart', {
    cart: app.getModel('Cart').get(),
RegistrationStatus: false,
    VInitFormattedString:VInitFormattedString,
    Signature:signature

  }).render('checkout/cart/cart');

}
```

Controller - COBilling.js

Update start function to make Visa checkout button non clickable on billing and cart page

```
/**
*Updatescartcalculationandpageinformationandrendersthebillingpage.
*@transactional
```

```
*@param{module:models/CartModel~CartModel}cart-ACartModelwrappingthecurrentBasket.
*@param{object}params-(optional)ifpassed,addedtoviewpropertiessotheycanbeaccessedinthetemplate.
*/
function start(cart, params) {

   app.getController('COShipping').PrepareShipments();

// TO handle the visa checkout click even on cart and billing page from mini cart
   session.custom.cyb_CurrentPage = "CybBilling";

   Transaction.wrap(function () {
      cart.calculate();
   });

var pageMeta = require('~/cartridge/scripts/meta');
   pageMeta.update({
      pageTitle: Resource.msg('billing.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')
   });
   returnToForm(cart, params);
}
```

Update the returnToForm() method

```
function returnToForm(cart, params) {
var pageMeta = require('~/cartridge/scripts/meta');

// if the payment method is set to gift certificate get the gift certificate code from the form
if (!empty(cart.getPaymentInstrument()) && cart.getPaymentInstrument().getPaymentMethod() ===
PaymentInstrument.METHOD_GIFT_CERTIFICATE) {
      app.getForm('billing').copyFrom({
         giftCertCode: cart.getPaymentInstrument().getGiftCertificateCode()
      });
   }

var VisaCheckout = require('int_cybersource/cartridge/scripts/visacheckout/helper/VisaCheckoutHelper');
var VInitFormattedString='',signature='';
            var result = VisaCheckout.Initialize(false);//no delivery address in lightbox
if (result.success) {
            VInitFormattedString = result.VInitFormattedString;
            signature = result.signature;
   }

   pageMeta.update({
      pageTitle: Resource.msg('billing.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')
   });

if (params) {
      app.getView(require('~/cartridge/scripts/object').extend(params, {
            VInitFormattedString:VInitFormattedString,
         Basket: cart.object,
Signature:signature,
```

```
        ContinueURL: URLUtils.https('COBilling-Billing')
      })).render('checkout/billing/billing');
    } else {
      app.getView({
        Basket: cart.object,
VInitFormattedString:VInitFormattedString,
        Signature:signature,
        ContinueURL: URLUtils.https('COBilling-Billing')
      }).render('checkout/billing/billing');
    }
}
```

Secure Acceptance

Generic Section

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

Update "export.init "function

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

[Note: Below changes are covered in custom code > Generic section > billing.js, defined here for reference only]

```
        // Secure Acceptance Redirect or iframe payment method : on selection change of saved credit card
// select credit card from list
    $('#creditCardList').on('change', function () {
      var cardUUID = $(this).val();
      if (!cardUUID) {$($checkoutForm).find('input[name$="_selectedCardID"]').val(''); return;}
                    populateCreditCardForm(cardUUID,selectedPaymentMethod);

      // remove server side error
      $('.required.error').removeClass('error');
      $('.error-message').remove();
    });

$('.creditCardList').on('change', function () {
                    var cardUUID = $(this).val();
                    if (!cardUUID) {return;}

                    var selectedPaymentMethod = $selectPaymentMethod.find(':checked').val();
                    populateCreditCardForm(cardUUID,selectedPaymentMethod);

                    // remove server side error
                    $('.required.error').removeClass('error');
                    $('.error-message').remove();
          });
```

Update "populateCreditCardForm" function

```
function populateCreditCardForm(cardID,selectedPaymentMethod) {
```

```
// load card details
var url = util.appendParamToURL(Urls.billingSelectCC, 'creditCardUUID', cardID);
ajax.getJson({
url: url,
callback: function (data) {
if (!data) {
window.alert(Resources.CC_LOAD_ERROR);
return false;
}
switch (selectedPaymentMethod) {
case "SA_REDIRECT":
$('.payment-method-expanded .saCCToken .field-wrapper').val(data.selectedCardID);

$("#dwfrm_billing_paymentMethods_creditCard_selectedCardID").val(data.selectedCardID);
break;
case "SA_IFRAME":
$('.payment-method-expanded .saIframeCCToken .field-wrapper').val(data.selectedCardID);

$("#dwfrm_billing_paymentMethods_creditCard_selectedCardID").val(data.selectedCardID);
break;
case "CREDIT_CARD":
setCCFields(data);
break;
default:
setCCFields(data);
}
}
});
}
```

Update "updatePaymentMethod" function

```
if (paymentMethodID=='SA_FLEX') {
              dataMethod = 'CREDIT_CARD';
       }
      var $selectedPaymentMethod = $paymentMethods.filter('[data-method="' + dataMethod +
'"]');

    if ($selectedPaymentMethod.length === 0) {
        $selectedPaymentMethod = $('[data-method="Custom"]');
    }

    if (paymentMethodID=="VISA_CHECKOUT") {
       $(".continue-place-order").hide();
       $(".visacheckoutbutton").show();
    }
    else if (paymentMethodID=="PAYPAL" || paymentMethodID=="PAYPAL_CREDIT") {
       $("#billingAgreementCheckbox").attr('checked',false);
```

```
        $(".continue-place-order").hide();
    }
    else {
        $(".continue-place-order").show();
        $(".visacheckoutbutton").hide();
    }

    if (paymentMethodID=="CREDIT_CARD" || paymentMethodID=="SA_SILENTPOST") {
        $(".spsavecard").show();
        $(".saflexshow").addClass('hide');
        $(".saflexhide").removeClass('hide');
    } else if ((paymentMethodID=="SA_REDIRECT" || paymentMethodID=="SA_IFRAME") &&
SitePreferences.TOKENIZATION_ENABLED) {
        $(".spsavecard").show();
    } else if (paymentMethodID=="SA_FLEX") {
        $(".saflexshow").removeClass('hide');
        $(".saflexhide").addClass('hide');
        $(".spsavecard").show();
    }
    else {
        $(".spsavecard").hide();
    }
```

Template – Cart.isml

- **Add below error condition just after cart-banner slot**

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

Template – Summary.isml

- **Secure acceptance error handling changes are done in summay.isml**

Please refer to the changes mentioned under custom code – generic section- > summary.isml

Template – Billing.isml

- **Add below error condition just after checkout progress indicator**

```
<isif condition="${!pdict.CurrentForms.multishipping.entered.value}">
<ischeckoutprogressindicator step="2" multishipping="false" rendershipping="${pdict.Basket.productLineItems.size() ==
0 ? 'false' : 'true'}"/>
<iselse/>
<ischeckoutprogressindicator step="3" multishipping="true" rendershipping="${pdict.Basket.productLineItems.size() == 0
? 'false' : 'true'}"/>
</isif>

        <isif condition="${pdict.CurrentHttpParameterMap.SecureAcceptanceError != null &&
!empty(pdict.CurrentHttpParameterMap.SecureAcceptanceError.stringValue)}">
                    <div class="error-form">${Resource.msg('sa.billing.payment.error.declined','cybersource',null)}</div>
        </isif>

<form action="${URLUtils.continueURL()}" method="post"
id="${pdict.CurrentForms.billing.htmlName}" class="checkout-billing address form-horizontal">
```

Template – paymentmethods.isml

- **Add below code snippet to handle secure acceptance error after closing on </legend> tag**

Changes are aleady covered under custom code > generic section-> paymentmethods.isml

Secure Acceptance Redirect Section

Controller - COPlaceOrder.js
Update "Start" function
[Note: Below changes are covered in custom code > Generic section > COPlaceOrder.js, defined here for reference only]

```
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.intermediateSA){
            app.getView({
                    Data:handlePaymentsResult.data, FormAction:handlePaymentsResult.formAction
                }).render(handlePaymentsResult.renderViewPath);
                    return {};
    } else if (handlePaymentsResult.missingPaymentInfo) {
```

Secure Acceptance Iframe Section

Controller - COPlaceOrder.js
Update "Start" function
[Note: Below changes are covered in custom code > Generic section > COPlaceOrder.js, defined here for reference only]

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

```
                }).render('checkout/summary/summary');
                    return {};
    }else if(handlePaymentsResult.intermediateSA){
            app.getView({
                    Data:handlePaymentsResult.data, FormAction:handlePaymentsResult.formAction
                }).render(handlePaymentsResult.renderViewPath);
                    return {};
    }else if (handlePaymentsResult.missingPaymentInfo) {
```

Controller - COSummary.js

Create new "SubmitOrder" function

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

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

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

Template changes

Update "summary.isml"

Secure acceptance Iframe related changes are done in summary.isml

Please refer to the changes mentioned under custom code – generic section- > summary.isml

Update "miniBillingInfo.isml"

Replace the line

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

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

```
</isif>
<isif condition="${!empty(billingAddress)}">
```

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

```
<div class="mini-billing-address  order-component-block">

<h3 class="section-header">
<isif condition="${!empty(pdict.Basket)}"><a href="${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>
```

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

```
 <isloop items="${paymentInstruments}" var="paymentInstr" status="loopstate">
     <div class="mini-payment-instrument  order-component-block <isif condition="${loopstate.first}"> first <iselseif
condition="${loopstate.last}"> last</isif>">
<h3 class="section-header">
                    <isif condition="${!empty(pdict.Basket)}"><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>
```

Update “miniSummary.isml”

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

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

<isif condition="${!empty(pdict.checkoutstep)}">
```

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

```
<isif condition="${checkoutstep <= 6}">
```

- **Replace pdict.Basket with lineCtnr at below places**

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

- Replace the line with below ${Resource.msg('summary.title','checkout',null)} ${Resource.msg('global.edit','locale',null)}

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

- Update the DIV "checkout-mini-cart" with below code

```
<div class="checkout-mini-cart">
<isif condition="${checkoutstep  != 5 && checkoutstep  != 6}">
<isminilineitems p_lineitemctnr="${lineCtnr}"/>
</isif>
</div>
```

- Update the DIV "checkout-order-totals" with below code

```
<div class=" checkout-order-totals">
<isif condition="${checkoutstep == 6}">
<isordertotals p_lineitemctnr="${lineCtnr}" p_showshipmentinfo="${true}" p_shipmenteditable="${false}" p_totallabel="${Resource.msg('global.ordertotal','locale',null)}"/>
<iselseif condition="${checkoutstep > 3}">
<isordertotals p_lineitemctnr="${lineCtnr}" p_showshipmentinfo="${true}" p_shipmenteditable="${true}" p_totallabel="${Resource.msg('global.ordertotal','locale',null)}"/>
<iselse/>
<isordertotals p_lineitemctnr="${lineCtnr}" p_showshipmentinfo="${false}" p_shipmenteditable="${false}"
p_totallabel="${Resource.msg('global.estimatedtotal','locale',null)}"/>
                                                    </isif>
</div>
```

Update "minshipments.isml"

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

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

- Replace pdict.Basket with lineCtnr at below places

```
<isif condition="${shipment.productLineItems.length <= 0 || shipment.custom.shipmentType == null && shipment.UUID==lineCtnr.defaultShipment.UUID && !empty(shipment.shippingAddress) && empty(shipment.shippingAddress.address1)}">
```

```
<isif condition="${Shipments.size() > 1 &&lineCtnr.productLineItems.size() > 0}"><div class="name">${Resource.msgf('multishippingshipments.shipment','checkout',null, shipmentCount)}</div></isif>
```

- Replace the line with below <a href="${editUrl}" class="section-header-

note">${Resource.msg('global.edit','locale',null)} twice in a file

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

- Replace pdict.Basket with lineCtnr at below line

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

Update "ReportCheckout.isml"

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

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

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

```
'BasketID', null != LineCntr ? LineCntr.UUID:null,
```

Core - scss changes

Update "_checkout.scss"

- Add below code snippet at the end of file

```
.SecureAcceptance_IFRAMEiframe{
                height:600px !important;
}

@mediascreen and ( max-width:1024px){
.SecureAcceptance_IFRAMEiframe{
                height:650px !important;
}
}


@mediascreen and ( max-width:767px){
.SecureAcceptance_IFRAMEiframe{
                height:670px !important;
}
}
```

Secure Acceptance Silent Post Section

Template - billing.isml

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

```
</form>
<div id="secureAcceptancePost">
</div>
```

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

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

Core – footer_UI.isml

Include scriptjquery.payment.js of cybersource cartridge

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

Core – Resource.ds

- **Add two new Resource in ResourceHelper.getResources**

```
TLS_WARNING                 : Resource.msg('global.browsertoolscheck.tls', 'locale', null),
    INVALID_SERVICE             : Resource.msg('checkout.getsignature.service.problem', 'cybersource', null),
                                                                INVALID_CREDITCARD          :
Resource.msg('checkout.invalid.credit.card.info', 'cybersource', null),
```

- Add below line under ResourceHelper.getUrls

```
paypalcallback : URLUtils.https('CYBPaypal-SessionCallback').toString(),
billingagreement :  URLUtils.https('CYBPaypal-BillingAgreement').toString(),
orderreview : URLUtils.https('COSummary-Start').toString(),
    silentpost              : URLUtils.https('CYBSecureAcceptance-GetRequestDataForSilentPost').toString(),
```

Core - billing.js

Create new “secureacceptance” on Click function

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

```
        $('.secureacceptance').on('click', function (e) {
                var $selectPaymentMethod = $('.payment-method-options');
                var selectedPaymentMethod = $selectPaymentMethod.find(':checked').val();
                if ('SA_SILENTPOST' == selectedPaymentMethod) {
                        var $checkoutForm = $('.checkout-billing');
                        var ccnumber = $($checkoutForm).find('input[name$="_creditCard_number"]').val();
                        var cvn =  $($checkoutForm).find('input[name$="_creditCard_cvn"]').val();
                        var month = $('.payment-method-expanded .month select').val();
                        var expyear = $('.payment-method-expanded .year select').val();
                        var dwcctype = $('.payment-method-expanded .cctype select').val();
                        var savecc = $($checkoutForm).find('input[name$="_creditCard_saveCard"]').is(':checked');
```

```
                                var customerEmail = $("#dwfrm_billing_billingAddress_email_emailAddress").val();
                                var cardmap= {'Visa': '001','Amex': '003','MasterCard': '002','Discover': '004','Maestro':'042'};
                                if(month.length == 1) {
                                        month = "0"+month;
                                }
                                var cctype  = cardmap[dwcctype];
                                var firstname =
encodeRequestFieldValue($($checkoutForm).find('input[name$="_addressFields_firstName"]').val());
                                var lastname =
encodeRequestFieldValue($($checkoutForm).find('input[name$="_addressFields_lastName"]').val());
                                var address1 =
encodeRequestFieldValue($($checkoutForm).find('input[name$="_addressFields_address1"]').val());
                                var address2 =
encodeRequestFieldValue($($checkoutForm).find('input[name$="_addressFields_address2"]').val());
                                var city =
                encodeRequestFieldValue($($checkoutForm).find('input[name$="_addressFields_city"]').val());
                                var zipcode =
encodeRequestFieldValue($($checkoutForm).find('input[name$="_addressFields_postal"]').val());
                                var country =
encodeRequestFieldValue($($checkoutForm).find('select[name$="_addressFields_country"]').val());
                                var state = $($checkoutForm).find('select[name$="_addressFields_states_state"]').val();
                                if (state===undefined) {
                                        state = $($checkoutForm).find('input[name$="_addressFields_states_state"]').val();
                                 }
                                 state = encodeRequestFieldValue(state);
                                var phoneno =
encodeRequestFieldValue($($checkoutForm).find('input[name$="_addressFields_phone"]').val());
                                var cctoken = encodeRequestFieldValue($('[data-
method="CREDIT_CARD"]').find('[name$="creditCard_selectedCardID"]').val());

                                var validCardType = dwcctype.toLowerCase();
                                var validCardNumber = $.payment.validateCardNumber(ccnumber);
                                var validCardCvv= $.payment.validateCardCVC(cvn,validCardType);
                                var validCardExp = $.payment.validateCardExpiry(month, expyear);

                                if(cctoken) {
                                        validCardNumber = true;
                                }

                                $($checkoutForm).find('input[name$="_creditCard_number"]').val("");
                                $($checkoutForm).find('input[name$="_creditCard_cvn"]').val("");
                                $($checkoutForm).find('input[name$="_creditCard_expiration_month"]').val("");
                                $($checkoutForm).find('input[name$="_creditCard_expiration_year"]').val("");
                                $($checkoutForm).find('input[name$="_creditCard_type"]').val("");

                                if(validCardCvv && validCardExp && validCardNumber) {
                                        var         data = {
                                                                custemail : customerEmail,
                                                                savecc : savecc,
                                                                firstname : firstname,
                                                                lastname : lastname,
                                                                address1 : address1,
```

```
                address2 : address2,
                city : city,
                zipcode : zipcode,
                country : country,
                state : state,
                phone : phoneno,
                cctoken : cctoken,
                format : 'ajax'
            };
            $.ajax({
                url: Urls.silentpost,
                type: "POST",
                data: data,
                success: function(xhr,data) {

                    if(xhr) {
                        if(xhr.error == true) {
                            $("#saspCardError").html(xhr.errorMsg);
                            $("#saspCardError").addClass('error');
                        }
                        else {
$("#secureAcceptancePost").html(xhr);

        $("#card_expiry_date").val(month +'-'+expyear);

        $("#card_type").val(cctype);

        $("#card_cvn").val(cvn);
                                                                    if(cctoken == null ||
cctoken == '') {

        $('#silentPostFetchToken').append('<input type="hidden" id="card_number" name="card_number" />');

        $("#card_number").val(ccnumber);
                                                                    }

        $("#silentPostFetchToken").submit();
                                                }
                                            }
                                            else {
                                        $("#saspCardError").html(Resources.INVALID_SERVICE);
                                                $("#saspCardError").addClass('error');
                                    }
                                },
                            error: function () {

$("#saspCardError").html(Resources.INVALID_SERVICE).addClass('error');
                                                }
                        });
                    }
                else{
```

```
                                        $("#saspCardError").html(Resources.INVALID_CREDITCARD);
                                        $("#saspCardError").addClass('error');

                                        returnfalse;
                                    }
                                }
                                else{
                                        $('.secureacceptance').prop("type", "submit").submit();
                                        returntrue;
                                }
                                                                        });
```

Create new "encodeRequestFieldValue" function

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

```
/**
*@function
*@descriptionfunctiontoconverthtmltagtoltorgt;
*@param{fieldValue}valueofthefield
*/
functionencodeRequestFieldValue(fieldValue) {

        return fieldValue.replace(/</g, "&lt;").replace(/>/g, "&gt;")
}
```

Update "updatePaymentMethod "function

[Note: Below changes are covered in custom code > Generic section > billing.js, defined here for reference only]

- **Update the function:**

```
function updatePaymentMethod(paymentMethodID) {
        var $paymentMethods = $('.payment-method');
        $paymentMethods.removeClass('payment-method-expanded');
        var dataMethod = paymentMethodID;
        if (paymentMethodID=='SA_SILENTPOST') {
                dataMethod = 'CREDIT_CARD';
        }
        var $selectedPaymentMethod = $paymentMethods.filter('[data-method="' + dataMethod + '"]');
        if ($selectedPaymentMethod.length === 0) {
                $selectedPaymentMethod = $('[data-method="Custom"]');
        }
        if (paymentMethodID=="VISA_CHECKOUT") {
                $(".continue-place-order").hide();
                $(".visacheckoutbutton").show();
        }
        else if (paymentMethodID=="PAYPAL" || paymentMethodID=="PAYPAL_CREDIT") {
                $("#billingAgreementCheckbox").attr('checked',false);
                $(".continue-place-order").hide();
        }
        else {
                $(".continue-place-order").show();
                $(".visacheckoutbutton").hide();
        }
        if (paymentMethodID=="CREDIT_CARD" || paymentMethodID=="SA_SILENTPOST") {
                $(".spsavecard").show();
```

```
        } else if ((paymentMethodID=="SA_REDIRECT" || paymentMethodID=="SA_IFRAME") &&
SitePreferences.TOKENIZATION_ENABLED) {
                $(".spsavecard").show();
        }
        else {
                $(".spsavecard").hide();
        }


        $selectedPaymentMethod.addClass('payment-method-expanded');

// ensure checkbox of payment method is checked
  $('input[name$="_selectedPaymentMethodID"]').removeAttr('checked');
  $('input[value=' + paymentMethodID + ']').prop('checked', 'checked');

  formPrepare.validateForm();
}
```

Secure Acceptance Flex Microform Section

Controller - COBilling.js

```
function validateBilling() {
    if (!app.getForm('billing').object.billingAddress.valid) {
        return false;
    }

    if (!empty(request.httpParameterMap.noPaymentNeeded.value)) {
        return true;
    }

    if (!empty(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value)
            &&
app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentIn
strument.METHOD_CREDIT_CARD)
            &&
empty(app.getForm('billing').object.paymentMethods.creditCard.selectedCardID.value)) {
       if (!app.getForm('billing').object.valid &&
empty(app.getForm('billing').object.paymentMethods.creditCard.flexresponse.value)) {
            return false;
       }
    }

    return true;
}
```

XML - creditcard.xml

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

       <field formid="selectedCardID" type="string" />

       <field formid="flexresponse" type="string" />
```

```
<!-- confirm action to save the card details -->
                                    <action formid="confirm" valid-form="true" />
```

Script - Resource.ds

Update ResourceHelper.getUrls

```
orderreview          : URLUtils.https('COSummary-Start').toString(),
orderSubmit          : URLUtils.https('COSummary-Submit').toString(),
WeChatCheckStatus    : URLUtils.https('CYBWeChat-CheckStatusService').toString(),
failWechatOrder      : URLUtils.https('COPlaceOrder-FailWeChatOrder').toString()
```

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 and summary.isml** page (Recommended: at bottom of page to have no visual impacts)

   [Note: summary .isml device fingerprint changes are covered in custom code-generic section-summary.isml]

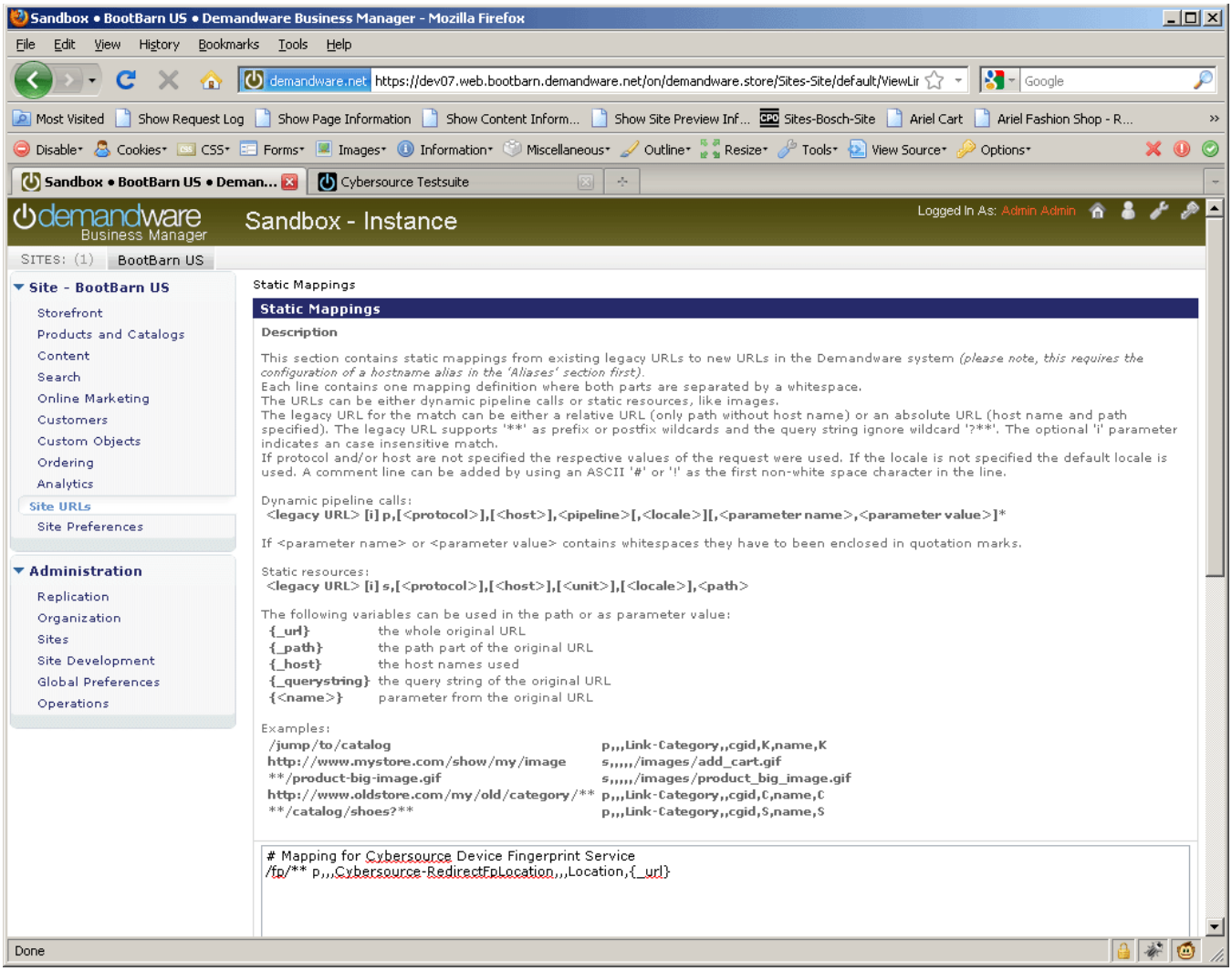
```
    <script>window.Countries = <isprint value="${json}" encoding="off"/></script>
<isif condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('CsDeviceFingerprintEnabled')}">
        <isinclude url="${URLUtils.url('CYBCredit-IncludeDigitalFingerprint')}"/>
    </isif>
</isdecorate>
```

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

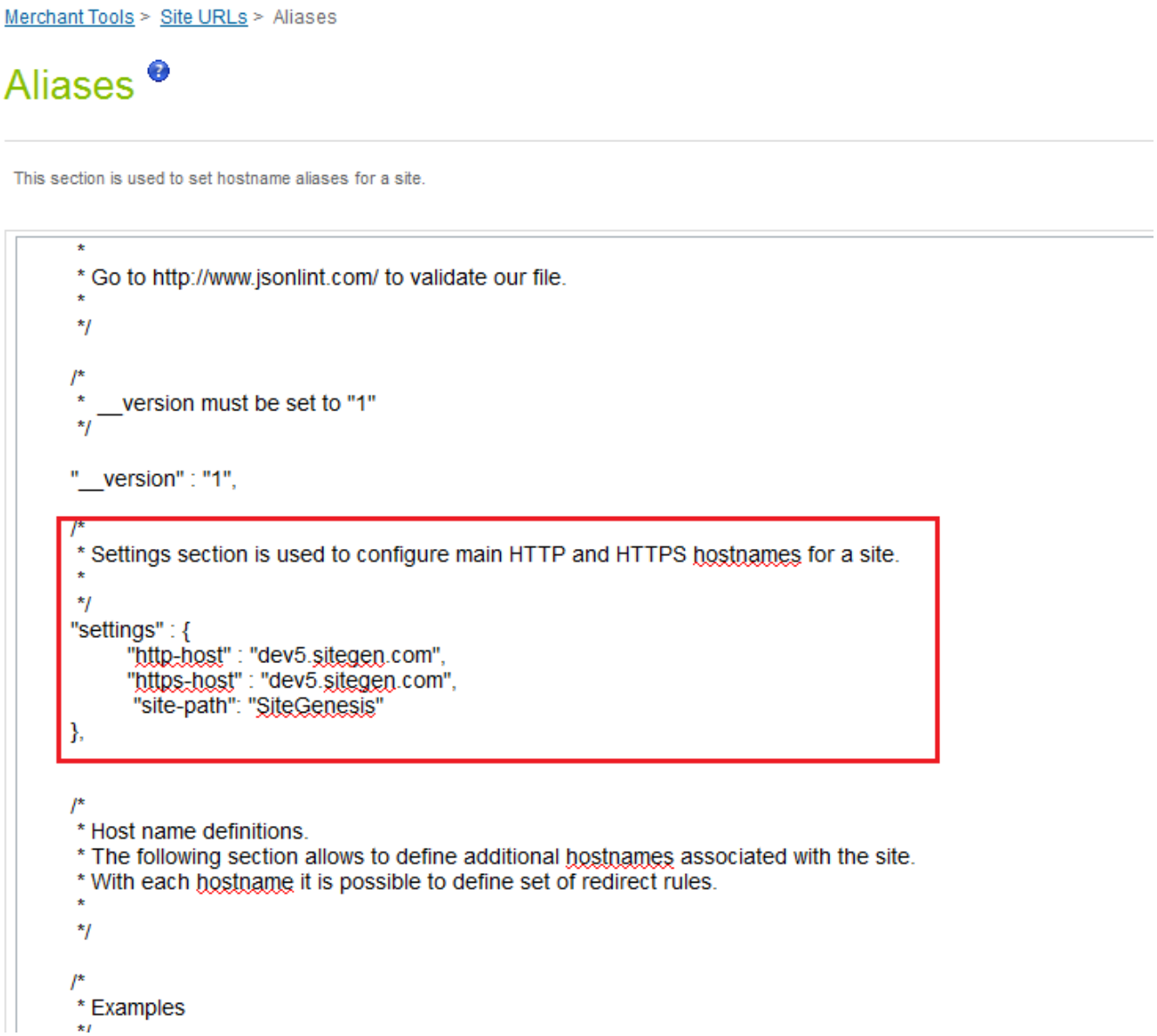
Device Fingerprint: submitted

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 a119emandware controller call. The controller call will redirect to the CsJetmetrixLocation.
Dynamic *If set to dynamic, you have to specify a mapping rule at SiteUrls->Static Mappings.*
All URLs matching the pattern will be redirected by the Demandware Server.



Example for a matching mapping rule for the device fingerprint redirection
Make an Alias entry in Business manager to execute Device finger print with " Dynamic" redirection Type
Go to Site > Site URLs > Aliases and add an Alias for your domain like below:



Site Configuration

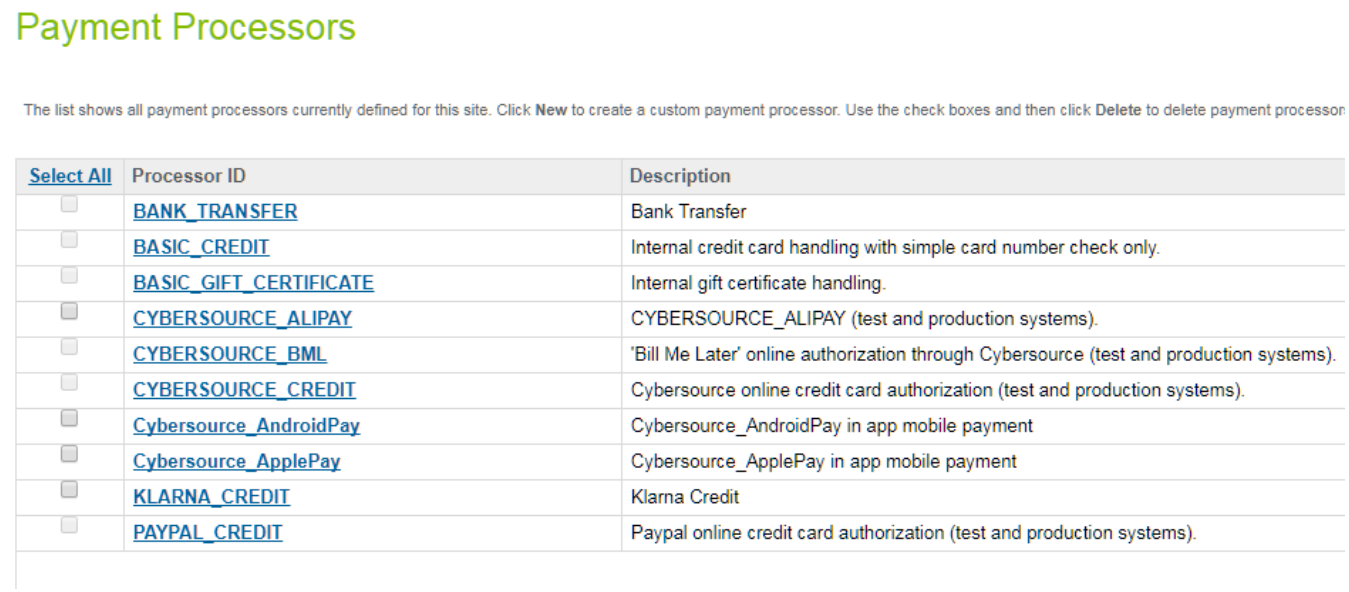
Configure Payment Processor

Steps to Create payment processor

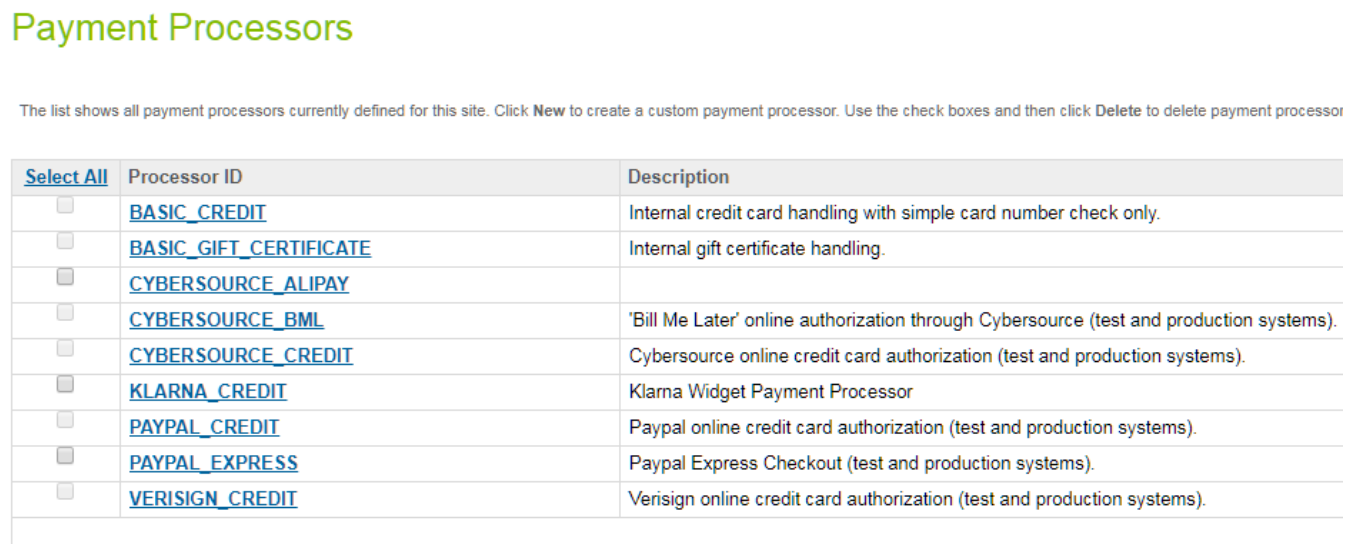
Go to Site -> Ordering -> Payment Processors; add a new payment processor with ID and description as given in below table

Processor ID	Description
BASIC_CREDIT	Internal credit card handling with simple card number checks only.
BASIC_GIFT_CERTIFICATE	Internal gift certificate handling.
CYBERSOURCE_ALIPAY	CYBERSOURCE_ALIPAY (test and production systems).
CYBERSOURCE_CREDIT	Cybersource online credit card authorization and visa checkout (test and production systems).
BANK_TRANSFER	Bank Transfer
Cybersource_AndroidPay	Cybersource_AndroidPay in app mobile payment
Cybersource_ApplePay	Cybersource_ApplePay in app mobile payment
KLARNA_CREDIT	Klarna
PAYPAL_CREDIT	PayPal online credit card authorization (test and production systems).
PAYPAL_EXPRESS	Pay Pal

[Payment Processors on Site genesis global]



[Payment Processors on Site genesis]



Import Meta Data

Import following site configuration meta-data through Business Manager:

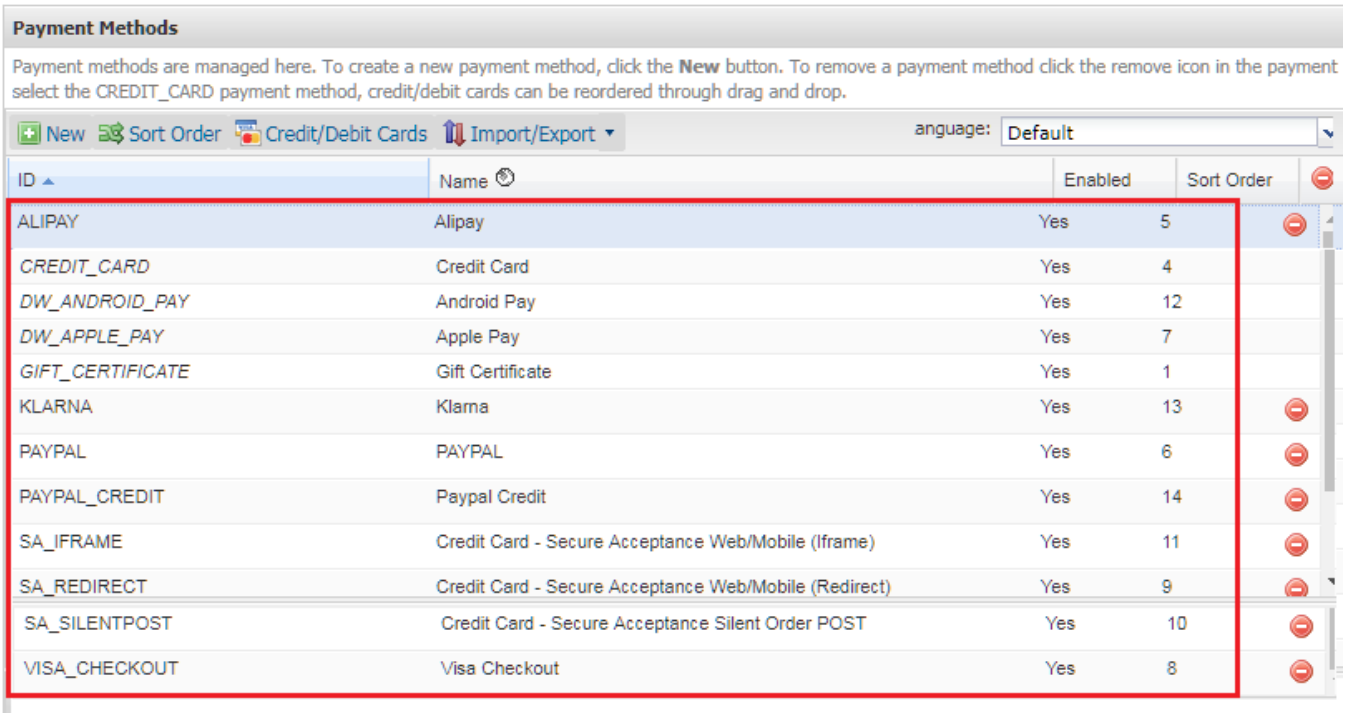
1. Go to **Cybersource/metadata/site_genesis_meta/sites/** folder.
2. Rename **yourSiteID** folder name with your site ID in Business Manager (this can be found by looking up **Administration->Sites->Manage Sites**)
3. Zip **site_genesis_meta** folder.
4. Go to **Administration->Site Development->Site Import & Export** and upload **site_genesis_meta.zip** file.
5. Import the uploaded zip file.

Import Payment Methods

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

- /CyberSource/metadata/site_genesis_meta/sites/yourSiteIID/payment-methods.xml
- Merchant can enable/disable any of the payment method listed below:

Payment Method ID	Payment Method Name
ALIPAY	Alipay
BANCONTACT	BANCONTACT
CREDIT_CARD	Credit Card
DW_ANDROID_PAY	Android Pay
DW_APPLE_PAY	Apple Pay
EPS	EPS
GIROPAY	GIROPAY
IDEAL	IDEAL Bank Transfer
KLARNA	Klarna
PAYPAL	Pay Pal
PAYPAL_CREDIT	PayPal Credit
SA_IFRAME	Credit Card - Secure Acceptance Web/Mobile (Iframe)
SA_REDIRECT	Credit Card - Secure Acceptance Web/Mobile (Redirect)
SA_SILENTPOST	Credit Card - Secure Acceptance Silent Order POST
SA_FLEX	Credit Card - Secure Acceptance Flex Microform
SOFORT	SOFORT
VISA_CHECKOUT	Visa Checkout
WECHAT	WeChat Pay



[Note:] Each APM defined above is tightly coupled with specific Merchant Id Configured in Custom preferences i.e. some APM are mapped with one merchant ID and some with other sas per merchant need.

Thus to execute a particular APM on SFCC, merchant should ensure that the respective APM is mapped with correct Merchant ID and password.

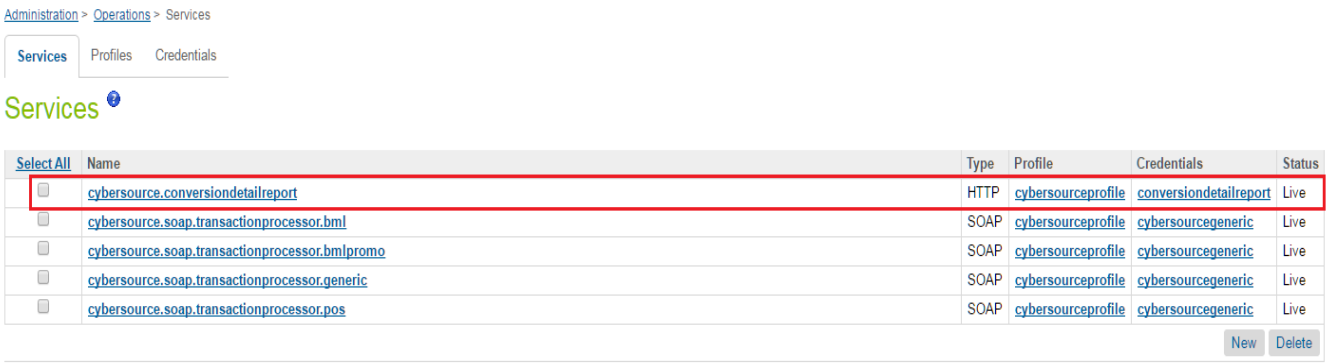
Configure Services

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

- /CyberSource/metadata/site_genesis_meta/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
  a. Cybersource.soap.transactionprocessor.generic
  b. Cybersource.soap.transactionprocessor.pos
  c. Cybersource.conversiondetailreport

The profile names cybersource profile, 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 SFCC with below separate Credentials as follows:

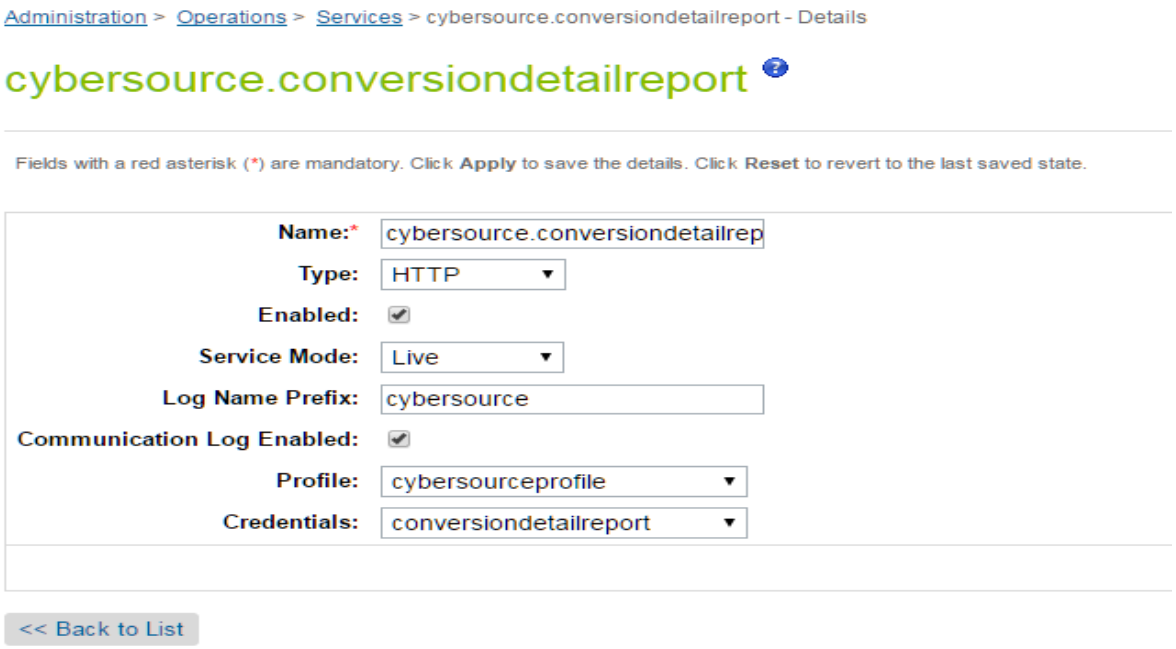
a. URL: Specify below report location along with the requested parameter ,the parameter values are replaced at runtime by the JOB code
   - Test environment URL is “https://apitest.cybersource.com/reporting/v3/conversion-details"
   - Production environment URL is "https://api.cybersource.com/reporting/v3/conversion-details"

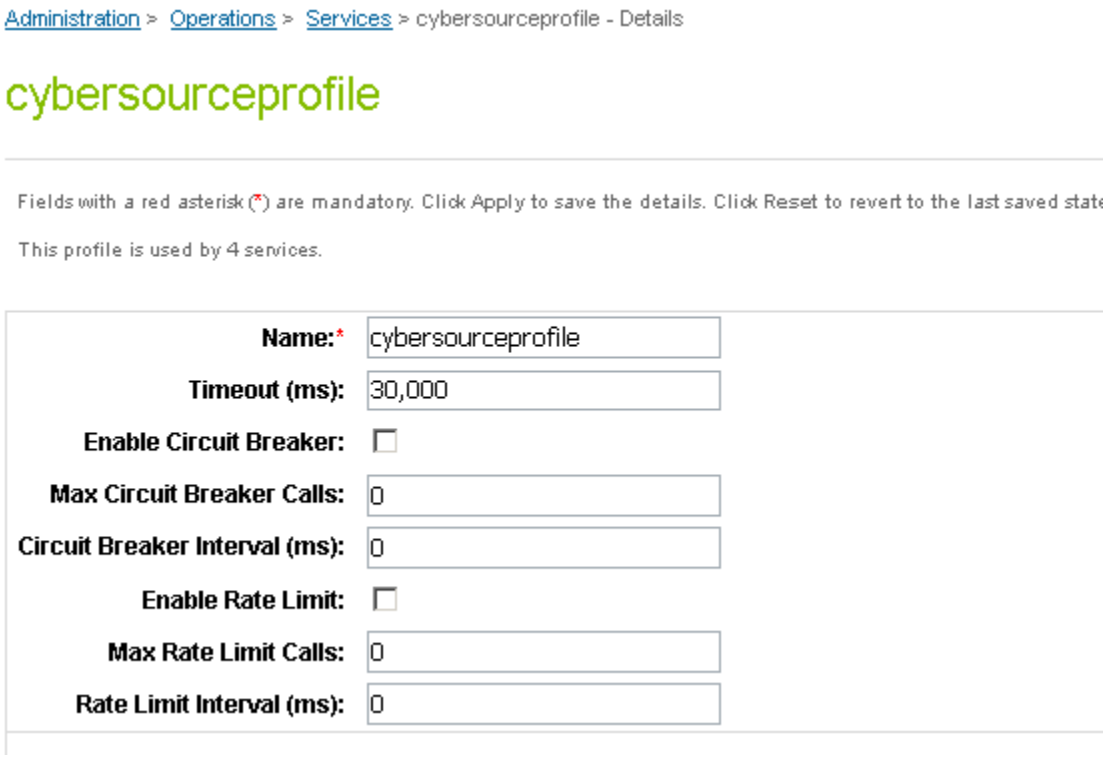
b. User: Merchant specific username [Represents user having report downloader role in cybersource console]

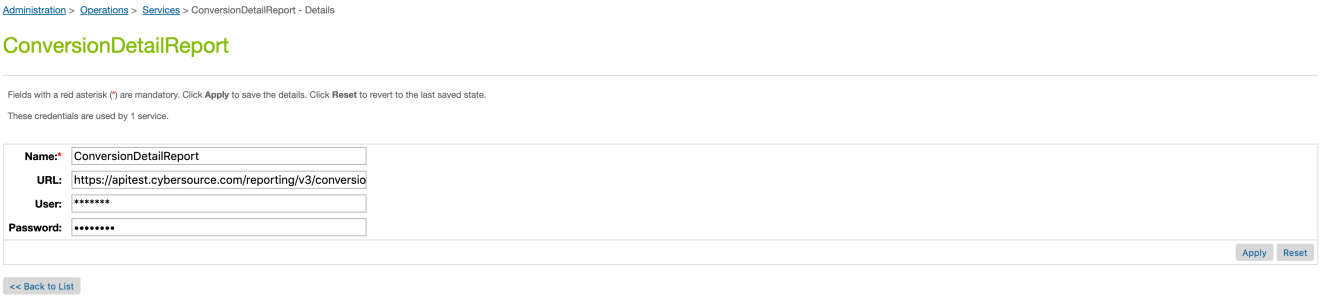
c. Password: Merchant specific password

- 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

CyberSourceSite Preference

Site Preferences Attribute

Site Preferences	Description
CyberSource Merchant Id(CsMerchantId)	CyberSource Merchant ID **Note: Merchant Key is defined at site preference level due to its length which could not be stored at DW service level configurations.**
CyberSource Merchant Key(CsSecurityKey)	CyberSource Security Key **Note:** Merchant Key is defined at site preference level due to its length which could not be stored at DW service level configurations.
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(CsShipFromStateCode)	Ship to data if fixed for the site
CyberSourceShipFromZipCode(CsShipFromZipCode)	Ship to data if fixed for the site
CyberSourceShipFrom Country Code(CsShipFromCountryCode)	Ship to data if fixed for the site
CyberSource Ignore AVS Result(CsAvsIgnoreResult)	AVS ignore results
CyberSource AVS Decline Flags(CsAvsDeclineFlags)	
CyberSource – On Delivery Address Verification Failure(CsDavOnAddressVerificationFailure)	
CyberSource – Enable Delivery Address Verification(CsDavEnable)	This will enable Delivery Address Verification, to help minimize risk of undeliverable or returns orders, because of user data entry errors.
CyberSource Merchant ID(PA)(CsPaMerchantId)	Payer Auth merchant ID
CyberSource Merchant Password(PA)(CsPaMerchantPassword)	Payer Auth Merchant Key
CyberSource Merchant Name(PA)(CsPaMerchantName)	Name
CyberSource Purchase Order Acceptance City(Tax)(CsPoaCity)	CyberSource purchase order acceptance data – used by Tax
CyberSource Purchase Order Acceptance State Code(Tax)(CsPoaStateCode)	CyberSource purchase order acceptance data – used by Tax
CyberSource Purchase Order Acceptance Zip Code(Tax)(CsPoaZipCode)	CyberSource purchase order acceptance data – used by Tax

CyberSource Purchase Order Acceptance Country Code(Tax)(CsPoaCountryCode)	CyberSource purchase order acceptance data – used by Tax
CyberSource Purchase Order Origin City((Tax)CsPooCity)	CyberSource purchase order origin data – used by Tax
CyberSource Purchase Order Origin StateCode(Tax)(CsPooStateCode)	CyberSource purchase order origin data – used by Tax
CyberSource Purchase Order Origin ZipCode(Tax)(CsPooZipCode)	CyberSource purchase order origin data – used by Tax
CyberSource Purchase Order Origin Country Code(Tax)(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(PA)(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 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 None Preauthorization: 0 passed in request Final authorization: 1 passed in request Undefined authorization:omit authIndicator field from the request message
CsDeveloperID	Merchant developer Id, mandatory for Cybersource configuration (max limit- String 8 char)
CyberSource Enable taxation (CsEnableTaxation)	Enable/Disable CyberSource Taxation service
LimitSavedCardRate	Limit number of credit card save attempts
SavedCardLimitFrame	Limit of card count in a certain time period
SavedCardLimitTimeFrame	Number of hours that saved credit card attempts are counted

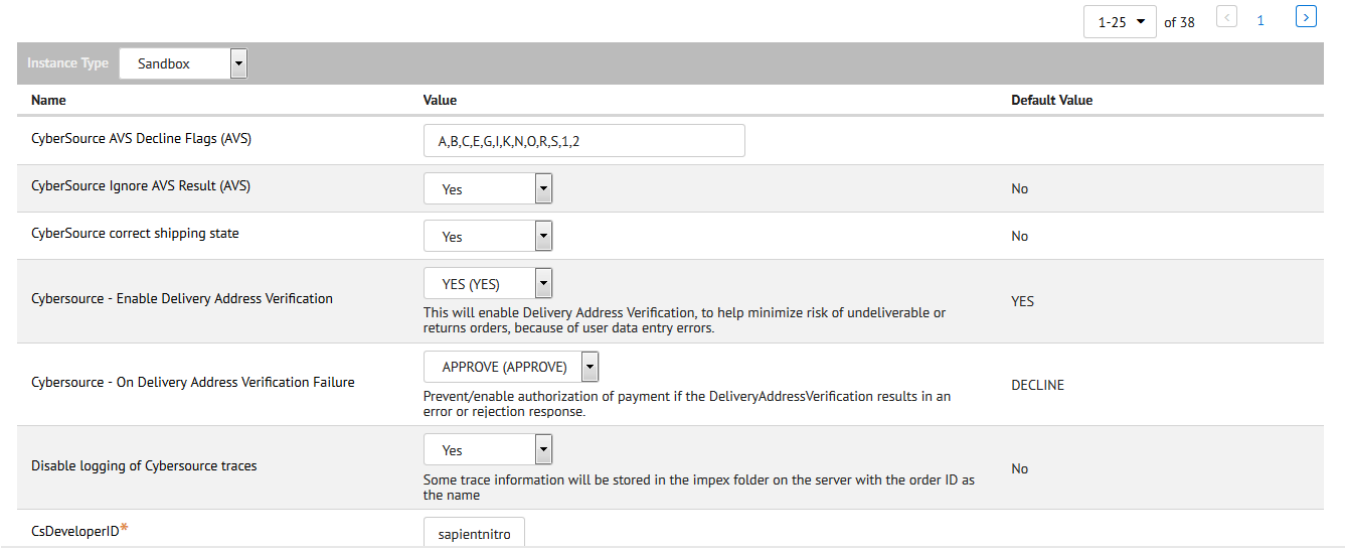
Note:

- **CyberSource Merchant Key (CsSecurityKey) - Security key is maintained at site preference level due to the bigger length of the Key which cannot be stored at service level**
- **Please contact Cybersource support for acquiring the Key**

Site preference data

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

The screen shot below depicts the site preferences configuration:



Cybersource - On Delivery Address Verification Failure:	DECLINE (DECLINE) (default) Prevent/enable authorization of payment if the DeliveryAddressVerificat	DECLINE *(DECLINE)*
Cybersource - Enable Delivery Address Verification:	YES (YES) (default) This will enable Delivery Address Verification, to help minimize risk of u	YES *(YES)*
CyberSource Merchant ID (PA):	sapient_nitro	
CyberSource Merchant Password (PA):	Wz/S0jZi10wezu7TJuXAixkAjmaqnlD1LxGfDnQaLq2LB9z6	
CyberSource Merchant Name (PA):	Merchant_ID	
CyberSource Purchase Order Acceptance City (Tax):		Lyndhurst
CyberSource Purchase Order Acceptance State Code (Tax):		NJ
CyberSource Purchase Order Acceptance Zip Code (Tax):		76208
CyberSource Purchase Order Acceptance Country Code (Tax):		US
CyberSource Purchase Order Origin City (Tax):		Lyndhurst
CyberSource Purchase Order Origin StateCode (Tax):		NJ
CyberSource Purchase Order Origin ZipCode (Tax):		76208
CyberSource Purchase Order Origin Country Code (Tax):		US
CyberSource Nexus States List:	Add Another Value	
CyberSource No Nexus States List:	Add Another Value	
Disable logging of Cybersource traces:	Some trace information will be stored in the impex folder on the server	false
Jetmetrix Location:	https://h.online-metrix.net Jetmetrix Location for Device fingerprint	
CsDeviceFingerprintOrgId:	1snn5n9w	

CsDeviceFingerprintOrgId:	1snn5n9w Organization ID for the device fingerprint check.	
Cybersource Device Fingeprint enabled:	☑ If enabled, a unique ID (fingerprint) of the customer's e	false
Device Fingerprint Redirection:	dynamic 'none' for no redirection, 'static' for pipeline redirection mappings Site Urls->Static mapping: # Mapping for Cy	none *(none)*
Cybersource - Enable Tokenization:	YES (YES) (default) This will enable Payment Tokenization.	YES *(YES)*
CyberSource Save Proof.xml (PA):	☑	false
CyberSource Save ParesStatus (PA):	☑ Save ParesStatus received as response from Pa Auther merchant account settings.	false
Decision Manager Enable for Card:	☑	true
Master Card Auth Indicator:	1 (Final authorization) Preauthorization: 0 passed in request Final authorizati	NONE *(Undefined authorization)*
CyberSource correct shipping state:	☑	false

Alipay Site Preference

Verify Alipay Site Preferences in already existing custom preferences group "CyberSource".

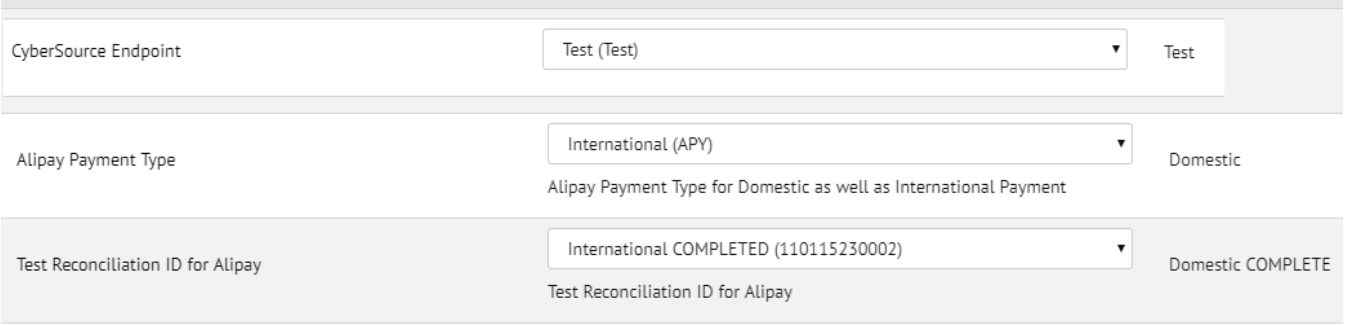
Site Preferences Attribute

Site Preferences	Description
CyberSource Endpoint (CsEndpoint)	CyberSource Alipay endpoint on different environments
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

Site preference data

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

The screen shot below depicts the site preferences configuration:



CyberSource Apply Pay Site Preference

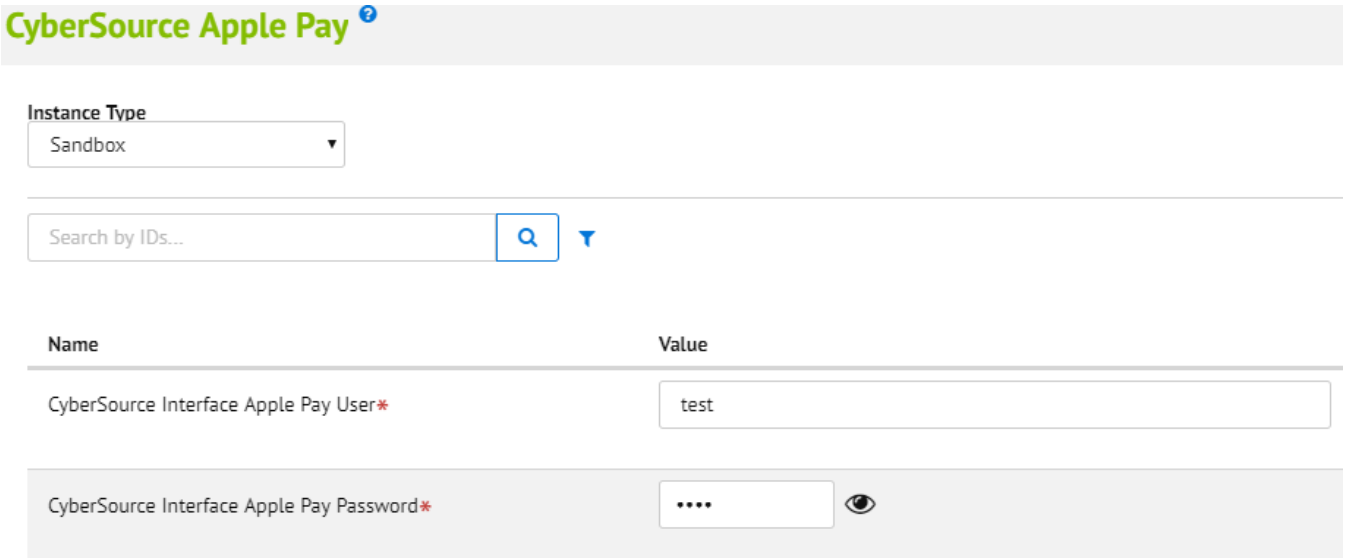
Site Preferences Attribute

Site Preferences	Description
CyberSource Interface Apple Pay User(CsApplePayUser)	CyberSource REST Interface Header Authentication User, need to be configured to authenticate the REST Interface for valid access.
CyberSource Interface Apple Pay Password (CsApplePayPassword)	CyberSource REST Interface Header Authentication Password, need to be configured to authenticate the REST Interface for valid access.

Site Preference data

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

The screen shot below depicts the site preferences configuration:



CyberSource_paypal Site Preference

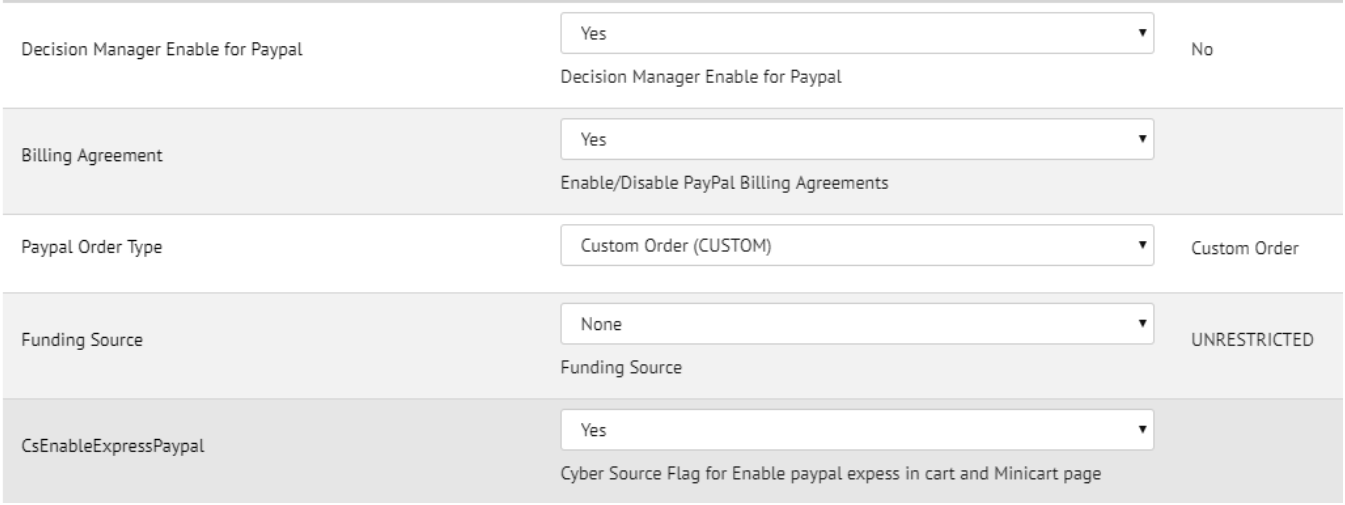
Site Preferences Attribute

Site Preferences	Description
Decision Manager Enable for PayPal(isDecisionManagerEnable)	Decision Manager Enable for PayPal
Billing Agreement(payPalBillingAgreements)	Enable/Disable PayPal Billing Agreements
PayPal Order Type(CsPaypalOrderType)	PayPal Order type
Funding Source (CsFundingSource)	Funding Source
CsEnableExpressPaypal	Cyber Source Flag for Enable PayPal expess in cart and Minicart page

Site preference data

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

The screen shot below depicts the site preferences configuration:



CyberSource Android Pay Site Preference

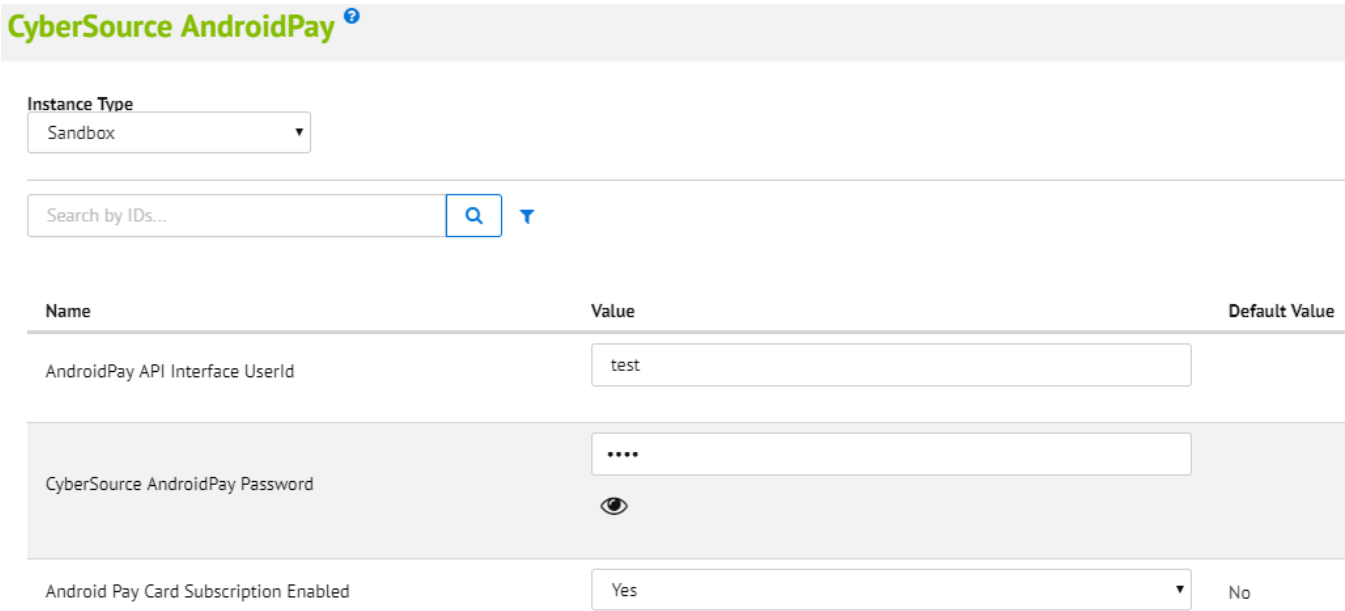
Site Preferences Attribute

Site Preferences	Description
AndroidPay API Interface UserId(cybAndroidPayInterfaceUser)	CyberSource REST Interface Header Authentication User, need to be configured to authenticate the REST Interface for valid access.
CyberSource AndroidPay Password(cybAndroidPayInterfacePassword)	CyberSource REST Interface Header Authentication Password, need to be configured to authenticate the REST Interface for valid access.
Android Pay Card Subscription Enabled(CsAndoridPayTokenizationEnabled)	Enable/disable subscription during autherisation, subscription is stored at order level attributes only

Site preference data

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

The screen shot below depicts the site preferences configuration:



CyberSource WeChat Pay Site Preference

Site Preferences Attribute

Site Preferences	Description
Test Reconciliation ID for WeChat Pay	Sets the status of the AP SALE such as settled, pending, abandoned, or failed
WeChatPayTransactionTimeout	Transaction Timeout for QR Code in WeChat Pay in seconds
CheckStatusServiceInterval	Interval in seconds before checking status of AP sale
NumofCheckStatusCall	Max number of calls to check status for each AP sale

Site preference data

CyberSource WeChat Pay site preference through Business Manager >StoreFront Site> Site Preferences > Custom Preferences.-CyberSource WeChat Pay

CyberSource Klarna Site Preference

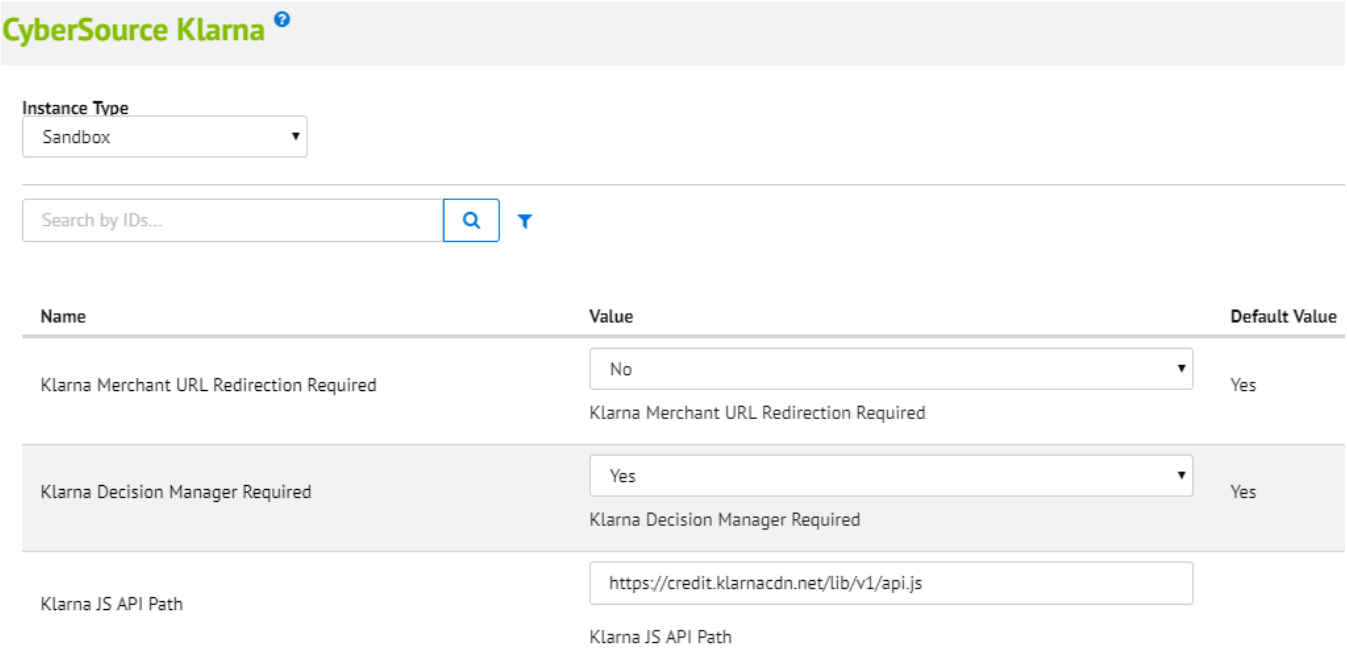
Site Preferences Attribute

Site Preferences	Description
Klarna Merchant URL Redirection Required(isKlarnaRedirectionRequired)	Enable/disable Klarna Merchant URL Redirection if Required
Klarna Decision Manager Required(isKlarnaDecisionManagerRequired)	Enable/disable Klarna Decision Manager if Required
Klarna JS API Path (klarnaJSAPIPath)	Klarna JS API Path

Site preference data

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

The screen shot below depicts the site preferences configuration



CyberSource Bank Transfer APM's Site Preference

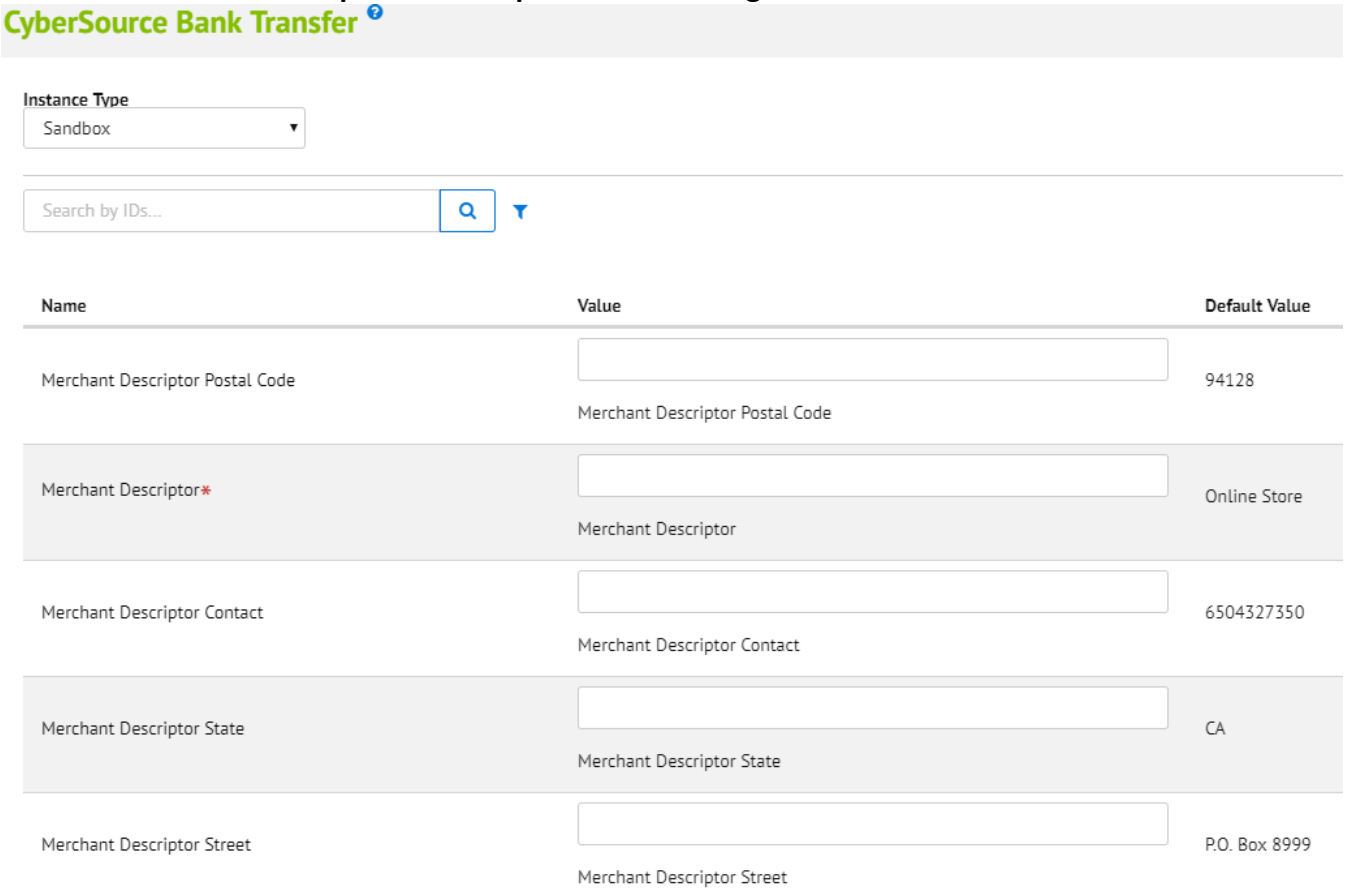
Site Preferences Attribute

Site Preferences	Description
Merchant Descriptor Postal Code(merchantDescriptorPostalCode)	Merchant Descriptor Postal Code
Merchant Descriptor(merchantDescriptor)	Merchant Descriptor
Merchant Descriptor Contact(merchantDescriptorContact)	Merchant Descriptor Contact
Merchant Descriptor State(merchantDescriptorState)	Merchant Descriptor State
Merchant Descriptor Street(merchantDescriptorStreet)	Merchant Descriptor Street
Merchant Descriptor City(merchantDescriptorCity)	Merchant Descriptor City
Merchant Descriptor Country(merchantDescriptorCountry)	Merchant Descriptor Country

Site preference data

CyberSource Bank Transfer site preference through Business Manager >StoreFront Site> Site Preferences > Custom Preferences.-CyberSource Bank Transfer

The screen shot below depicts the site preferences configuration



CyberSource Visa Checkout Site Preference

Site Preferences Attribute

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

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

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

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

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

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

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

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

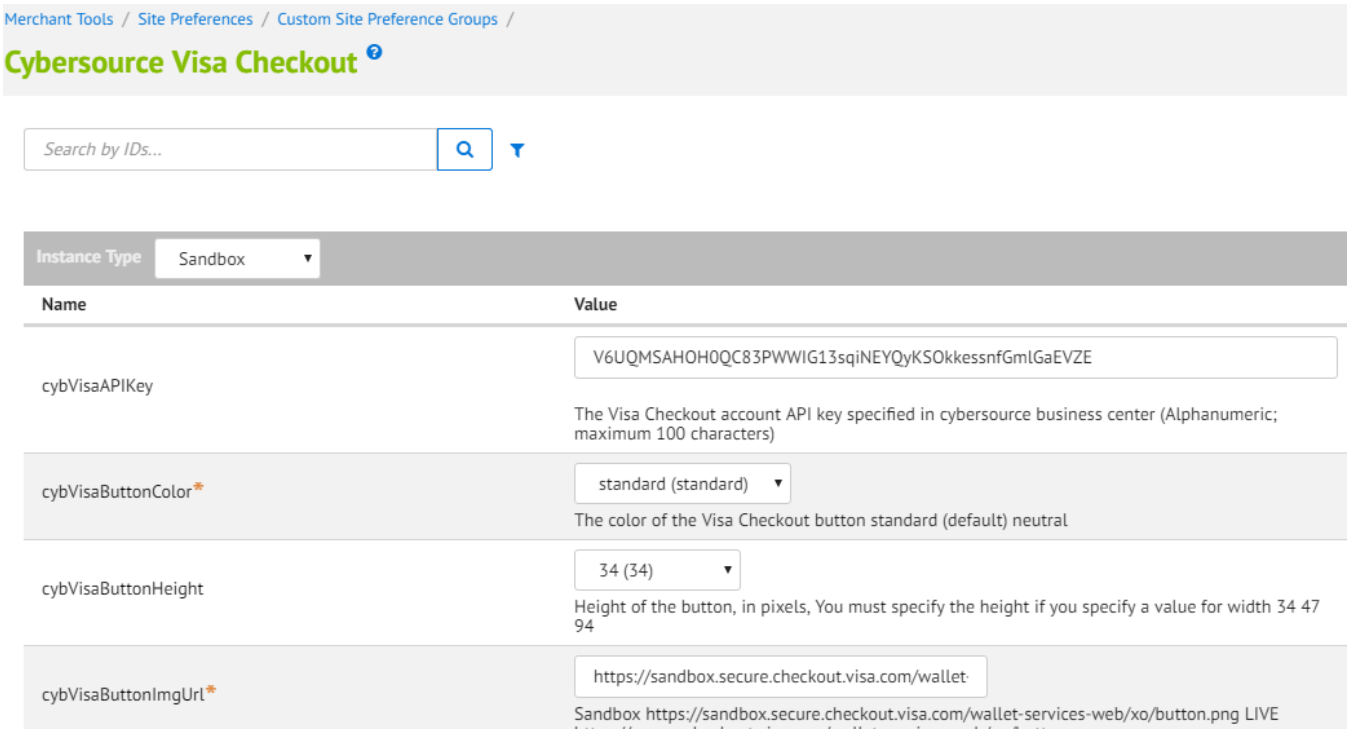
Field	Description	Possible Values
cybVisaButtonSize	You can either specify size to display a standard size button, or you can specify height and width to specify a custom size. If you do not specify size or both height and width, the button size is 213 pixels. If you specify height or width, the value of size is ignored	154 - small 213 - medium (default) 425 - High resolution or large
cybVisaButtonHeight	Height of the button, in pixels, You must specify the height if you specify a value for width	34 47 94
cybVisaButtonWidth	Width of the button, in pixels, You must specify the width if you specify a value for height	-less than 477 if height is 34; default value is 154 -greater than 212 and less than 659 if height is 47; default value is 213

		-greater than 424 and less than 1317 if height is 94; default value is 425
cybVisaButtonColor	The color of the Visa Checkout button	standard (default) neutral
cybVisaCardBrands	Override value for brands associated with card art to be displayed. If a brand matching the consumer's preferred card is specified, the card art is displayed on the button; otherwise, a generic button is displayed	Comma Separated list is accepted VISA MASTERCARD AMEX DISCOVER
cybVisaThreeDSActive	Whether Verified by Visa (VbV) is active for this transaction. If Verified by Visa is configured, you can use threeDSActive to deactivate it for the transaction; otherwise, VbV will be active if it has been configured	false (default) true
cybVisaThreeDSSuppressChallenge	Whether a Verified by Visa (VbV) consumer authentication prompt is suppressed for this transaction. If true, VbV authentication is performed only when it is possible to do so without the consumer prompt.	true - Do not display a consumer prompt. false - Allow a consumer prompt
cybVisaTellMeMoreLinkActive	Indicate whether Tell Me More Link to be displayed with VISA button	true (default) false
cybVisaButtonOnCart	Indicate whether Visa checkout button to be displayed on minicart and cart page	true (default) false

Site preference data

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

The screen shot below depicts the site preferences configuration:



CyberSource Secure Acceptance Site Preference

Site Preferences Attribute

Attribute ID	Data Type	Description
CsSAOverrideBillingAddress	Boolean	Cybersource Secure Acceptance Override Billing Address
CsSAOverrideShippingAddress	Boolean	Cybersource Secure Acceptance Override Shipping Address
CsCvnDeclineFlags	Boolean	CyberSource Ignore CVN Result (CVN) [should be in sync with CYB profile cvn flag]
SA_Redirect_AccessKey	String	Secure Acceptance Redirect Access Key. **Note:** Contact CyberSource support team for more details.
SA_Redirect_ProfileID	String	Secure Aceeptance Redirect Profile ID **Note:** Contact CyberSource support team for more details.
SA_Redirect_SecretKey	String	Secure Acceptance Redirect Secret Key **Note:** Contact CyberSource support team for more details.
SA_Iframe_AccessKey	String	Secure Acceptance Iframe Access Key **Note:** Contact CyberSource support team for more details.

SA_Iframe_ProfileID	String	Secure Acceptance Iframe Profile ID **Note:** Contact CyberSource support team for more details.
SA_Iframe_SecretKey	String	Secure Acceptance Iframe secret key **Note:** Contact CyberSource support team for more details.
SA_Silent_AccessKey	String	Secure Acceptance Silent Post Access Key **Note:** Contact CyberSource support team for more details.
SA_Silent_ProfileID	String	Secure Aceeptance Silent Post Profile ID **Note:** Contact CyberSource support team for more details.
SA_Silent_SecretKey	String	Secure Acceptance Silent Post Secret Key **Note:** Contact CyberSource support team for more details.
CsSARedirectFormAction	String	Cybersource secure acceptance redirect form action **Note:** Contact CyberSource support team for more details.
CsSAIframetFormAction	String	Cybersource secure acceptance Iframe form action **Note:** Contact CyberSource support team for more details.
Secure_Acceptance_Token_Create_Endpoint	String	Secure Acceptance Token Create Endpoint **Note:** Contact CyberSource support team for more details.
Secure_Acceptance_Token_Update_Endpoint	String	Secure Acceptance Token Update Endpoint **Note:** Contact CyberSource support team for more details.

Site Preferences Data

Name	Value	Default Value
Secure Aceeptance Redirect Profile ID	B6D0DA91-DF4F-4FC8-BFC5-C6851698A452	
Secure Acceptance Redirect Secret Key	023ad2d6731a485d98220f7a1c09a955169a78b	
Secure Acceptance Redirect Access Key	e0ae7f5bf469321892f5549a3f7248dc	
Cybersource Secure Acceptance Override Billing Address	Yes	No
Cybersource Secure Acceptance Override Shipping Address	Yes	No
Cybersource secure acceptance redirect form action	https://testsecureacceptance.cybersource.com/pa	
Secure Aceeptance Silent Post Profile ID	2E714D8B-E7BE-4A02-9A48-6BAA65406D3C	
Secure Acceptance Silent Post Secret Key	a57a860d7f0f450bb1d870ffd3b2f24e73499746	
Secure Acceptance Silent Post Access Key	a8b2c94df04b3e63a1c79919ca0a1c31	

Configure Payment Method

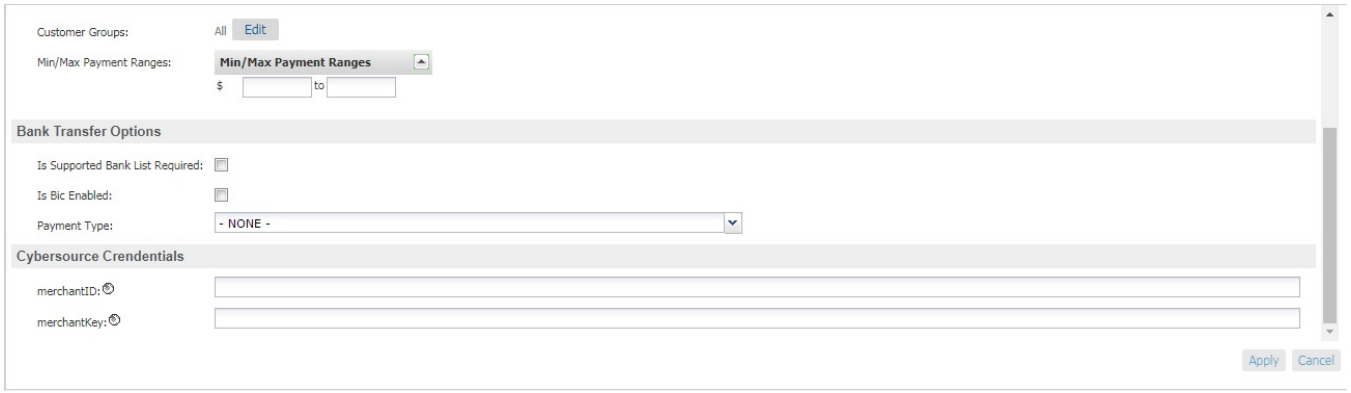
Generic Changes

Attribute ID	Data Type	Localizable	Description
merchantID	String	Yes	Attribute to store merchant id specific to payment method. If configured will be used for service calls else global site preference of Merchant ID will be used
merchantKey	String	Yes	Attribute to store merchant key specific to payment method. If configured will be used for service calls else global site preference of Merchant Key will be used

Bank Transfer APM's

Attribute ID	Data Type	Localizable	Description
isBicEnabled	Boolean	No	Attribute to check if BIC field is required for EPS and GIROPAY to display on billing page
isSupportedBankListRequired	Boolean	No	Attribute to check if bank list is required for IDEAL to display on billing page

paymentType	Enum of Strings	No	Payment type for bank transfer APMs, required to add new value for future bank transfer APM

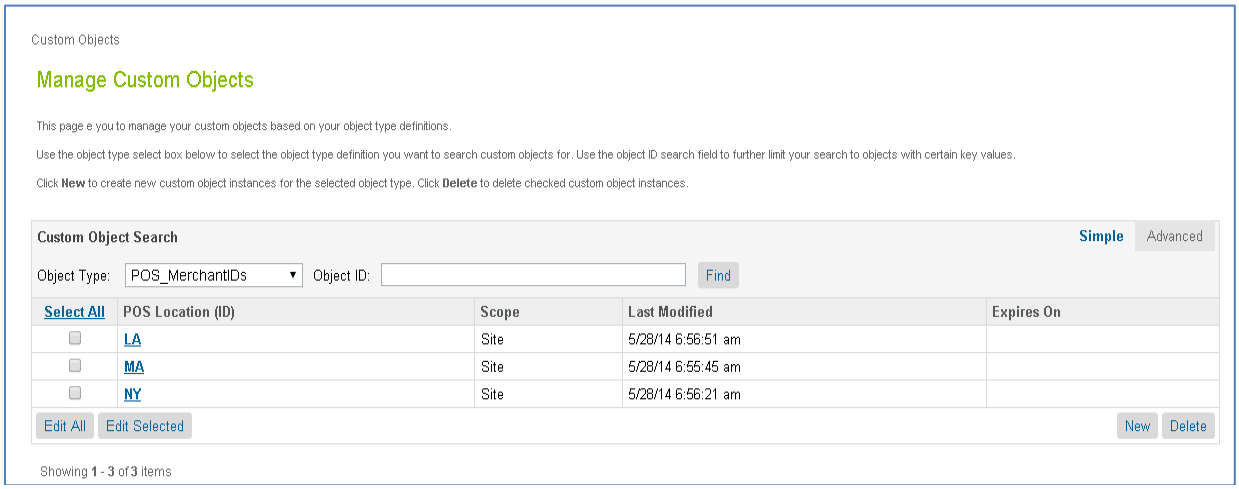


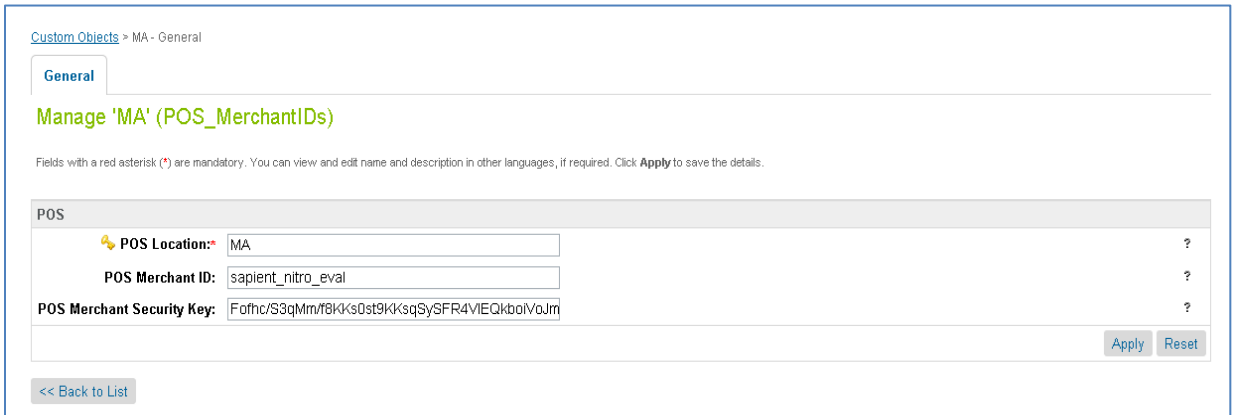
Configure Custom Objects

Retail POS

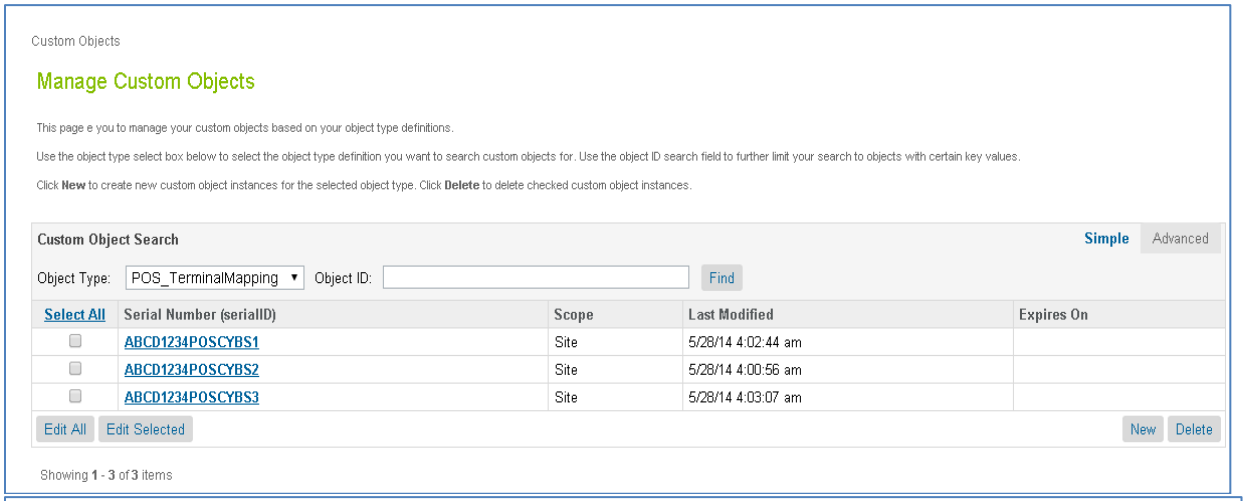
Two custom objects have been added for POS transactions. Ensure to populate these custom objects with merchant specific data. Below are screenshots of sample custom object entry for both custom objects:

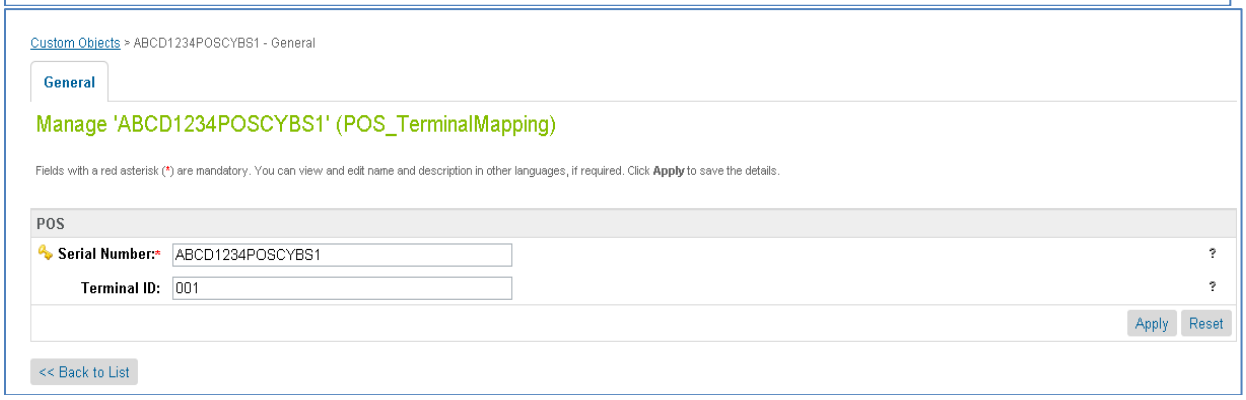
a. POS_MerchantIDs





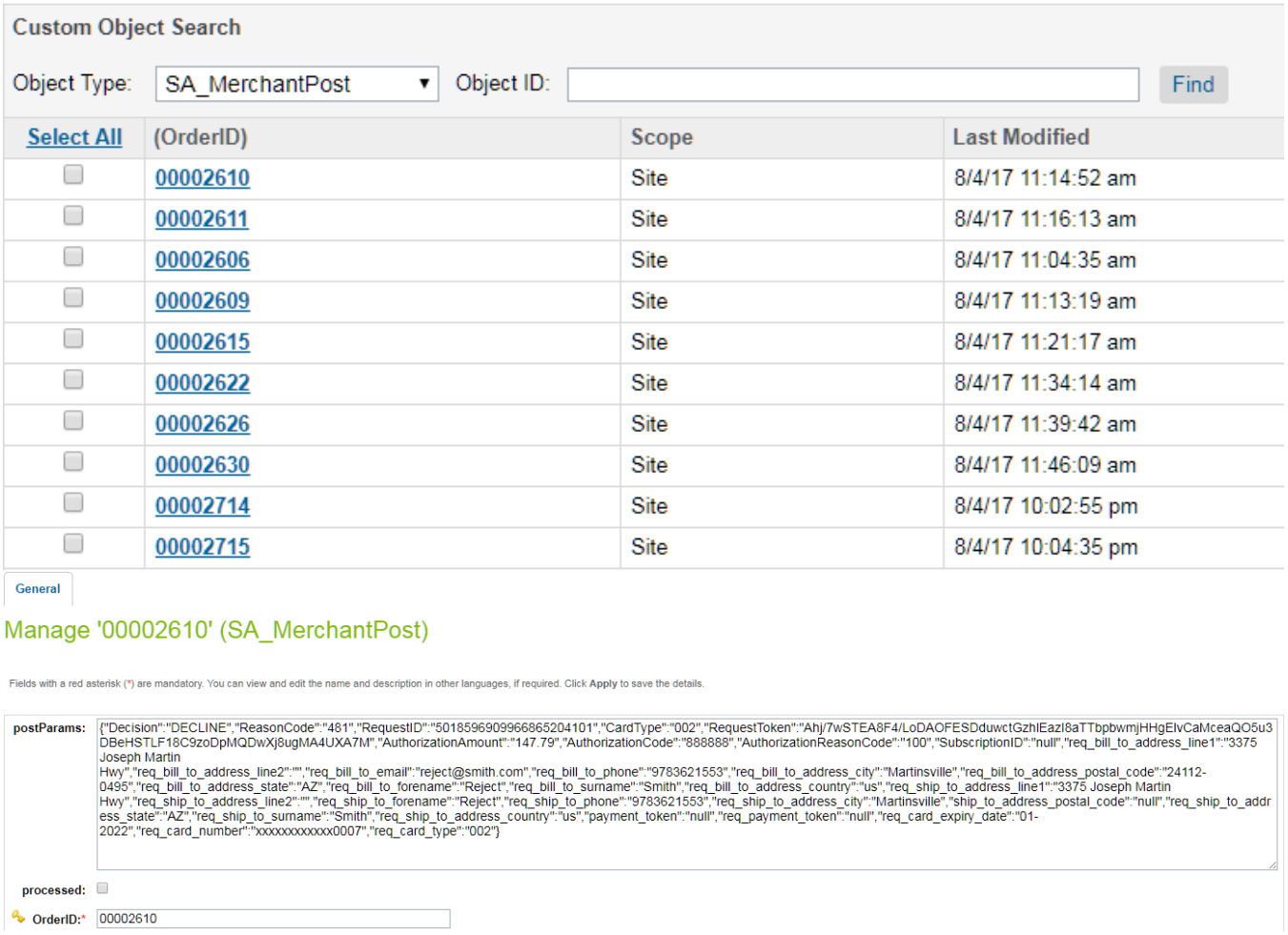
b. POS_TerminalMapping





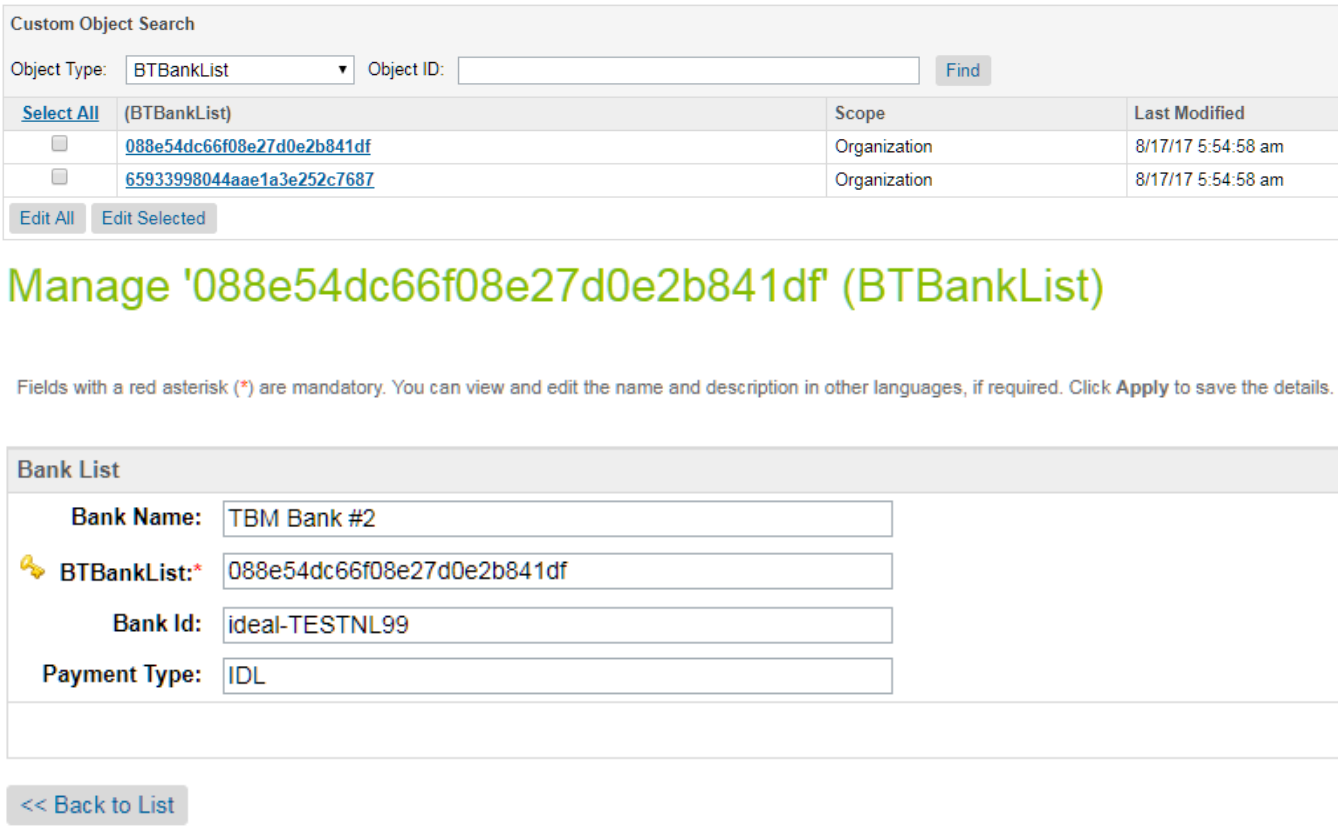
SA Merchant Post Notifications

A new custom object has been added for Secure Acceptance Web/Mobile and Iframe transactions. Ensure to populate these custom objects for every order placed through SA Web Mobile and Iframe. Below are screenshots of sample custom object entry:



Bank Transfer APM's Bank Options List

A new custom object has been added for Bank transfer bank list. Ensure to populate these custom objects with merchant specific data. Below are screenshots of sample custom object entry:



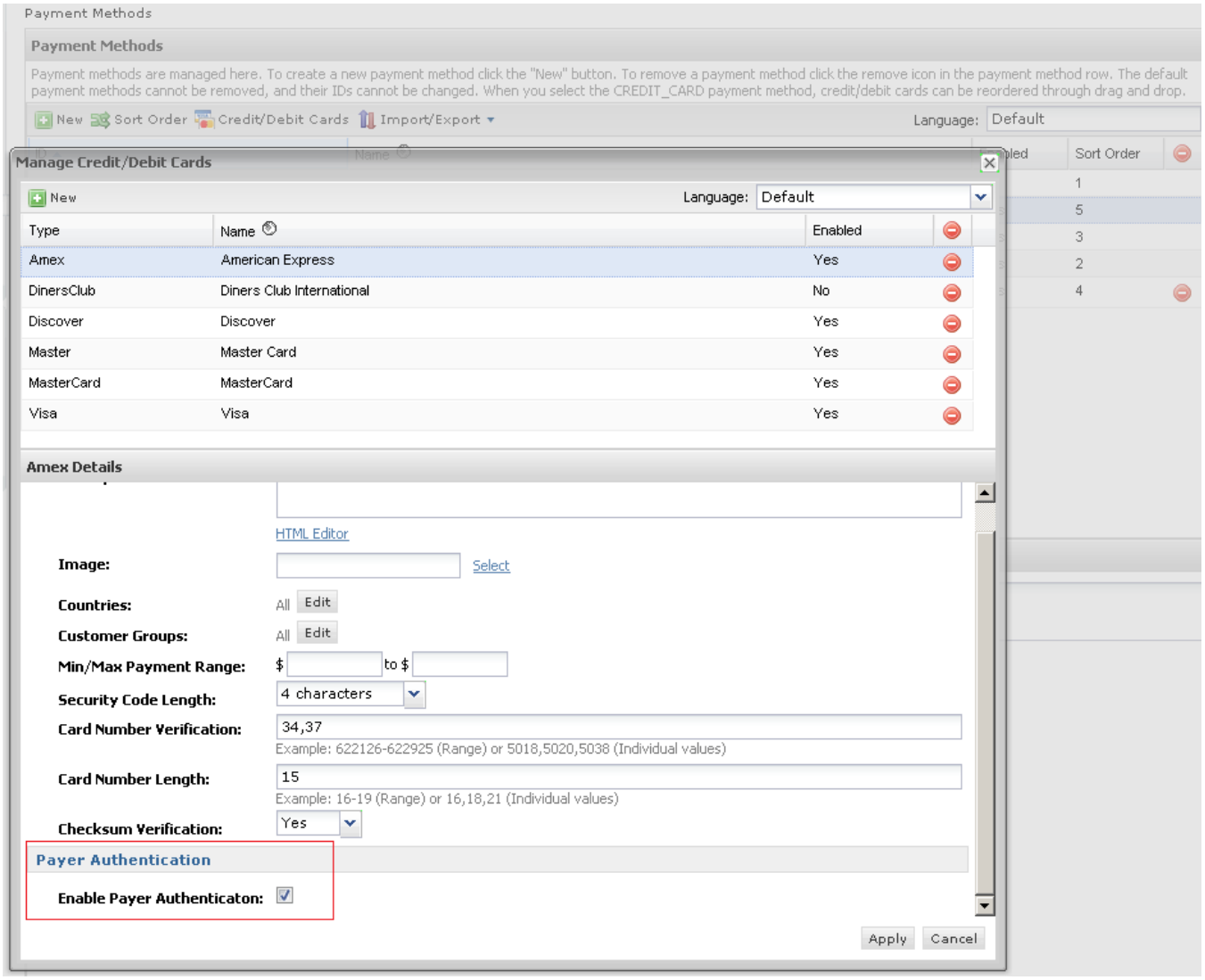
Custom Attribute in Customer Profile

Custom Attribute "billingAgreementID" has been created in Customer's profile system object to store the PayPal Billing Agreement ID. Value of this Custom Attribute will be used to place the order with PayPal payment method.

Attribute ID	Attribute Name	Type
billingAgreementID	PayPal Billing Agreement ID	String

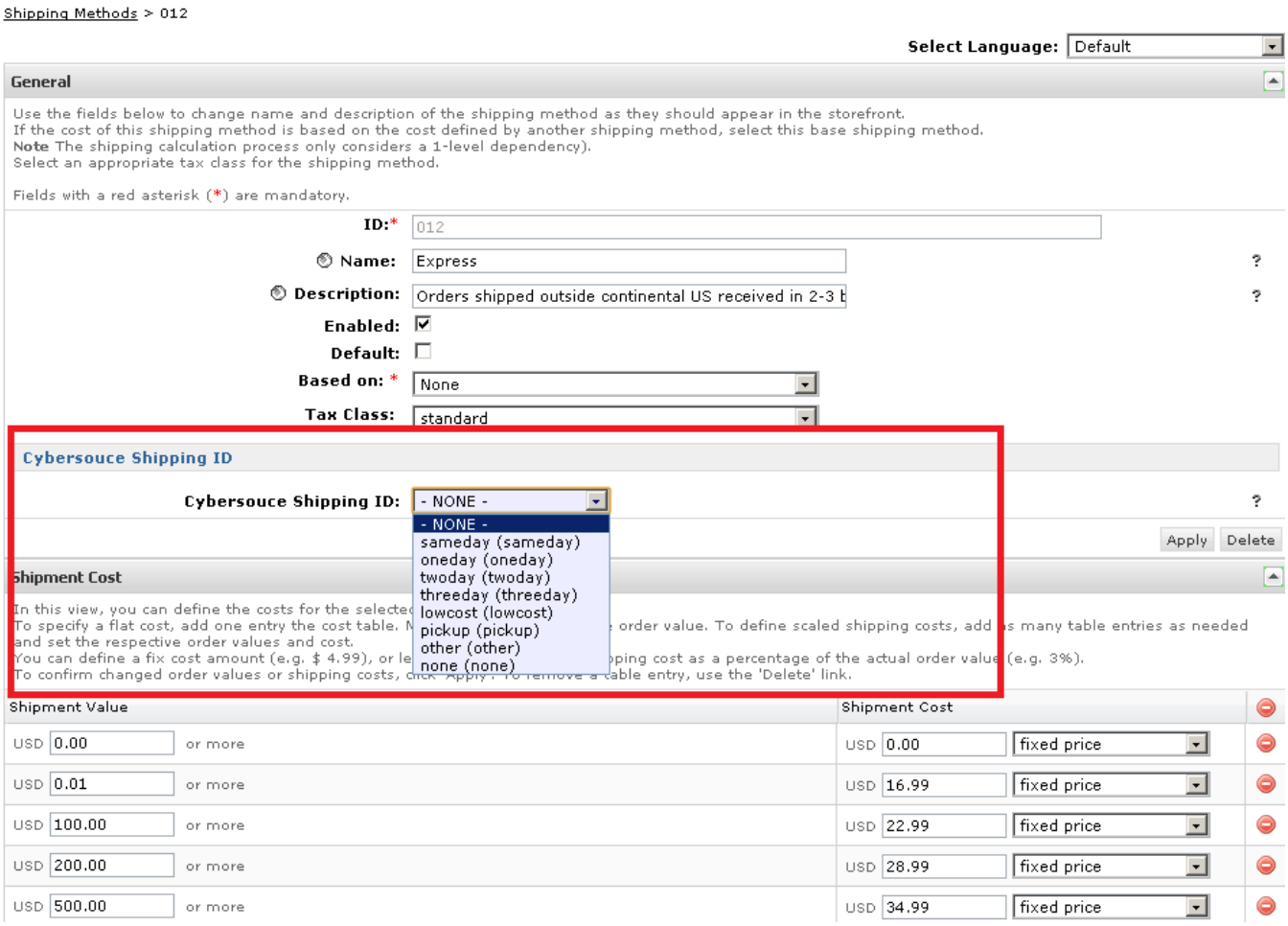
Enable Payer Authentication for cards

Update credit card preference through Business Manager >StoreFront Site> Ordering> Payment Methods> Credit Card/Debit Cards >Choose cardand then modify Enable Payer Authentication checkbox
The screen shot below depicts the site preferences configuration:



Update shipping method preference

Update shipping method preference through Business Manager >StoreFront Site> Ordering> Shipping Methods > Name >CyberSource Shipping ID
The screen shot below depicts the site preferences configuration:

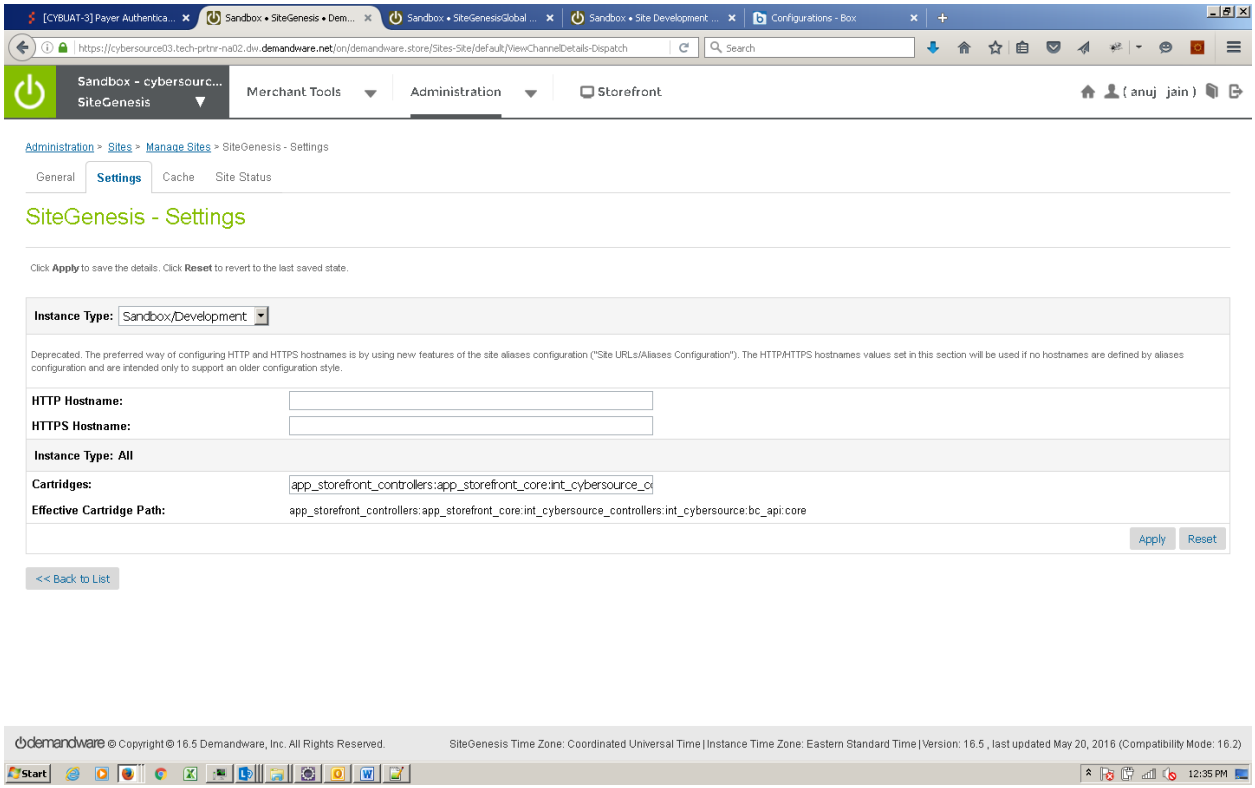


Applying CyberSource Cartridge to the Site

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

Add int_cybersource cartridges to the BM cartridge path.

Add int_cybersource_controllers and int_cybersource cartridges to the cartridge path as depicted in the following screen:



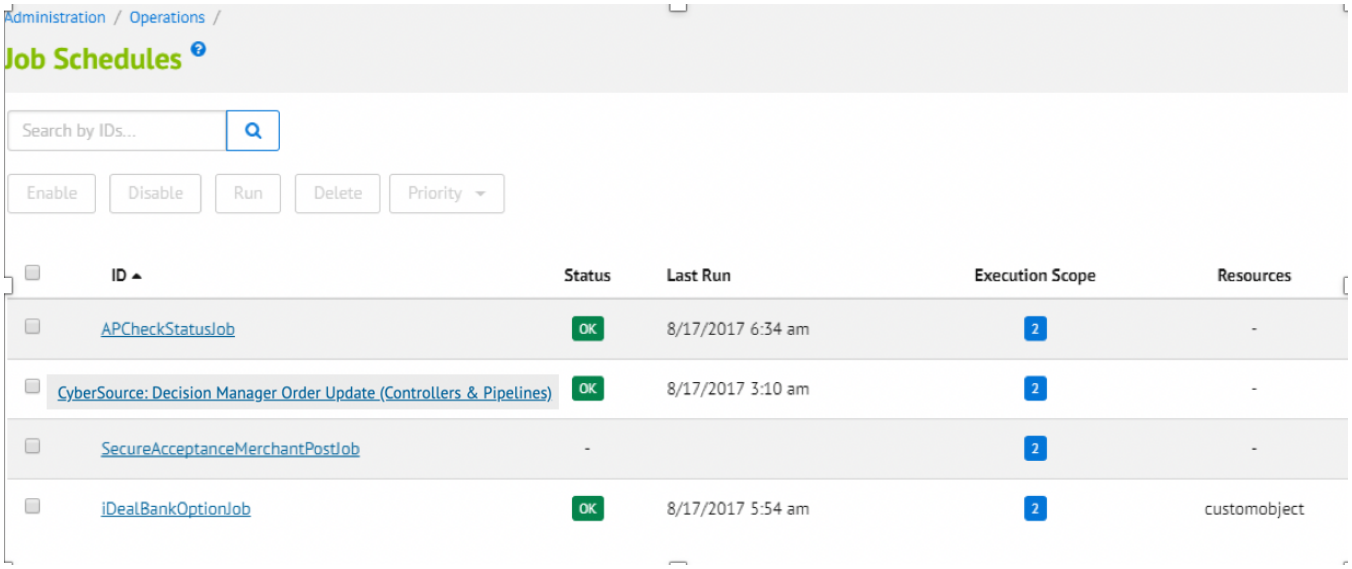
Batch Jobs

Cybersource cartridge has 4 batch jobs created for different functional items and are placed under int_cybersource cartridge:

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

/CyberSource/metadata/site_genesis_meta/jobs.xml – this will add below jobs

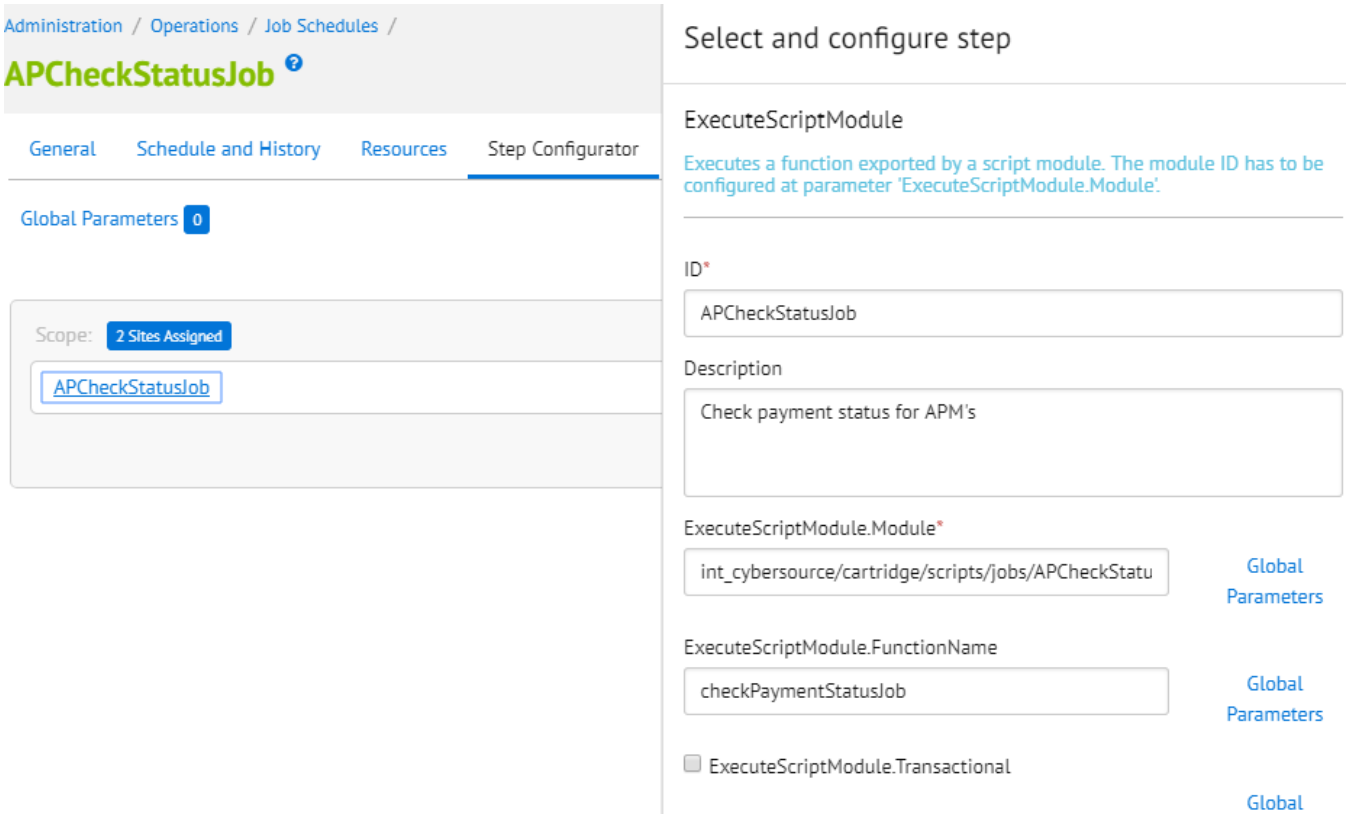
1. **APCheckStatusJob**
2. **CyberSource: Decision Manager Order Update (Controllers & Pipelines)**
3. **SecureAcceptanceMerchantPostJob**
4. **iDealBankOptionJob**



Below steps are used to configured each job in Business manager

Batch Job for AP Check Status

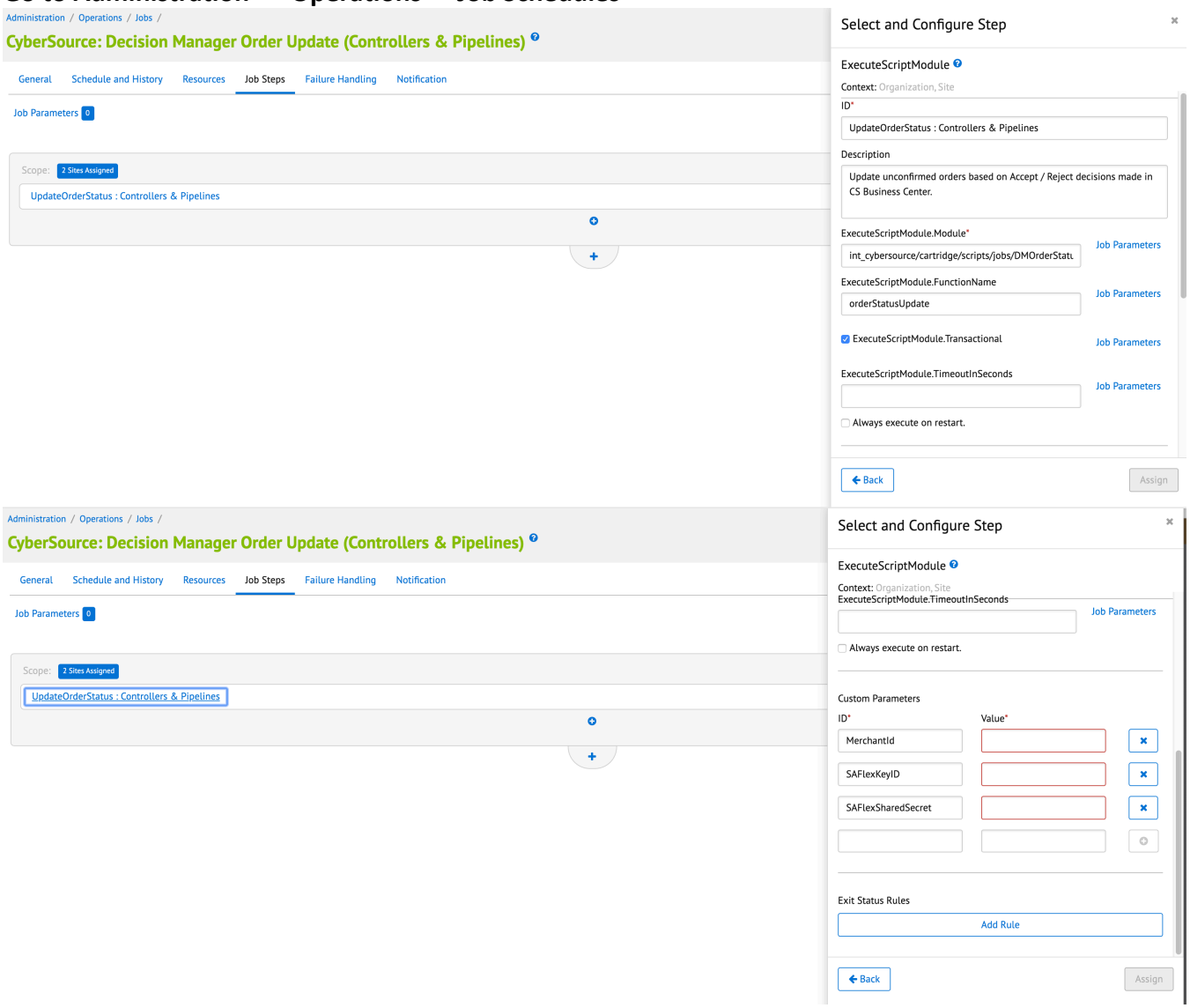
- Add new batch job for AP check status service
- Verify the newly added batch jobs for AP Check Status Service
- **Go to Administration - > Operations -> Job Schedules**



Batch Job for Conversion Detail Report

- Add new batch job to update order status in BM for CyberSource “Accepted” & “Rejected” orders.

Verify the newly added batch jobs for Conversion detail report service.
Go to Administration - > Operations -> Job Schedules

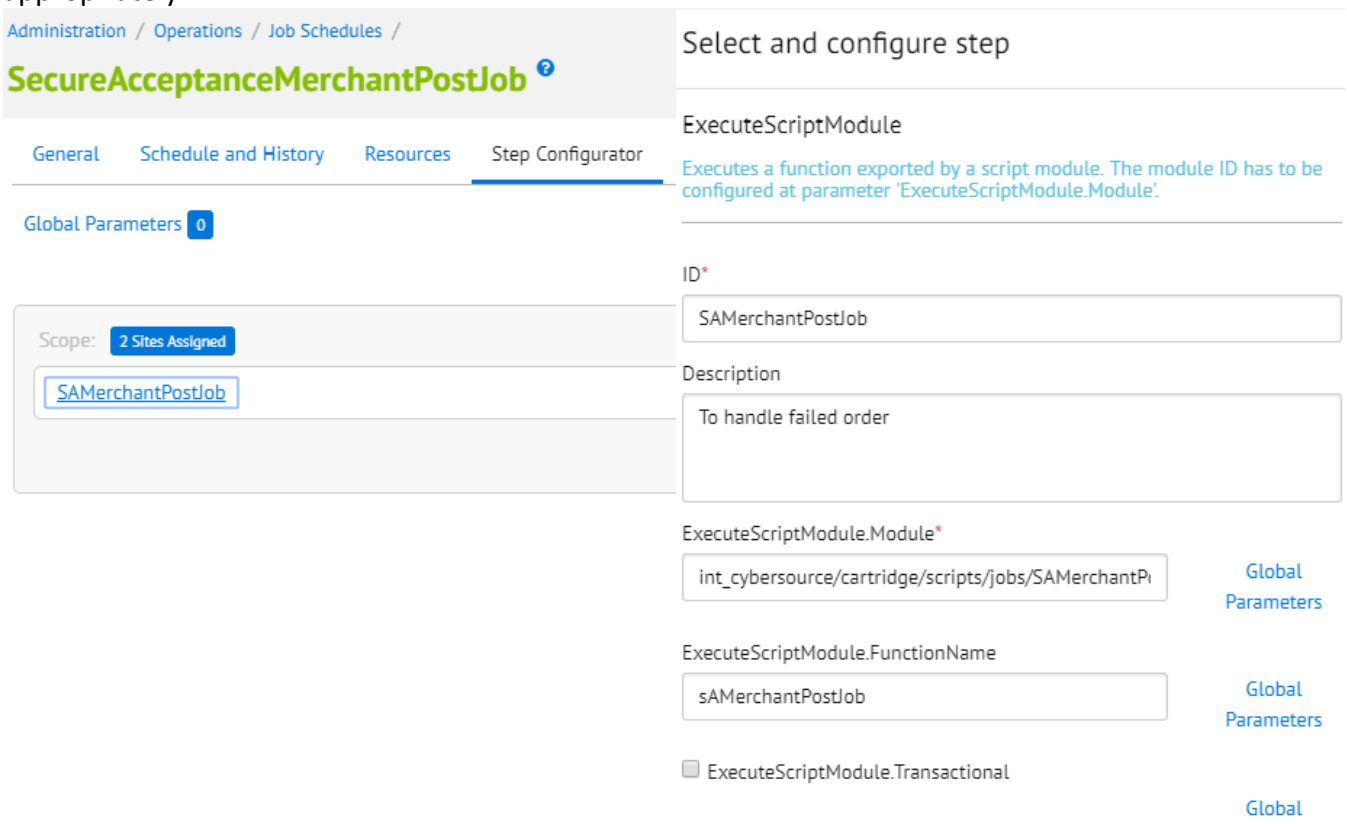


The batch job created for cybersource conversion detail report specified below, it updates the status of order in demandware which are in CREATED state and mark them as “CANCELLED” for rejected order or “NEW” for accepted order. The accepted orders are marked for “READY FOR EXPORT” as well.

Secure Acceptance Merchant Post Batch Job

- Add new Service for secure Acceptance Order update via merchant post notifications

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

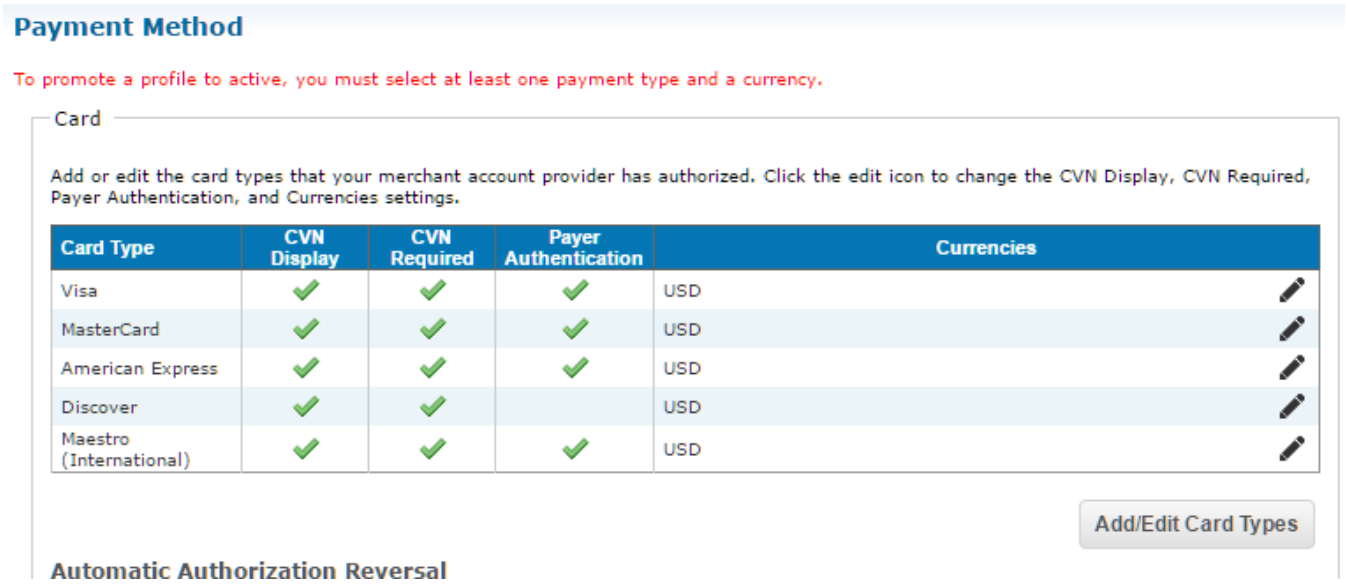


Secure Acceptance Profile Configuration into CyberSource Business Manager

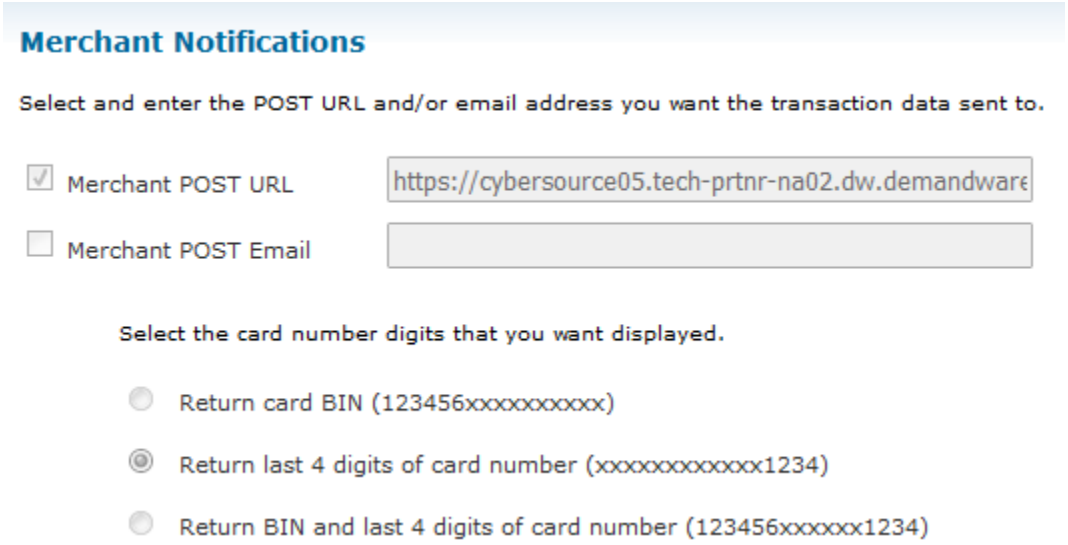
Secure Acceptance profile settings are configured on CyberSource business center console; along with other settings below are key settings which must be configured in cybersource profiles in order to complete the checkout process successfully.

Profile name	Notification Section [Merchant post URL]
SA Redirect	[Merchant specific URL]/SECURE_ACCEPTANCE-MerchantPost
SA Iframe	[Merchant specific URL]/SECURE_ACCEPTANCE-MerchantPost
SA SilentPost	N/A

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



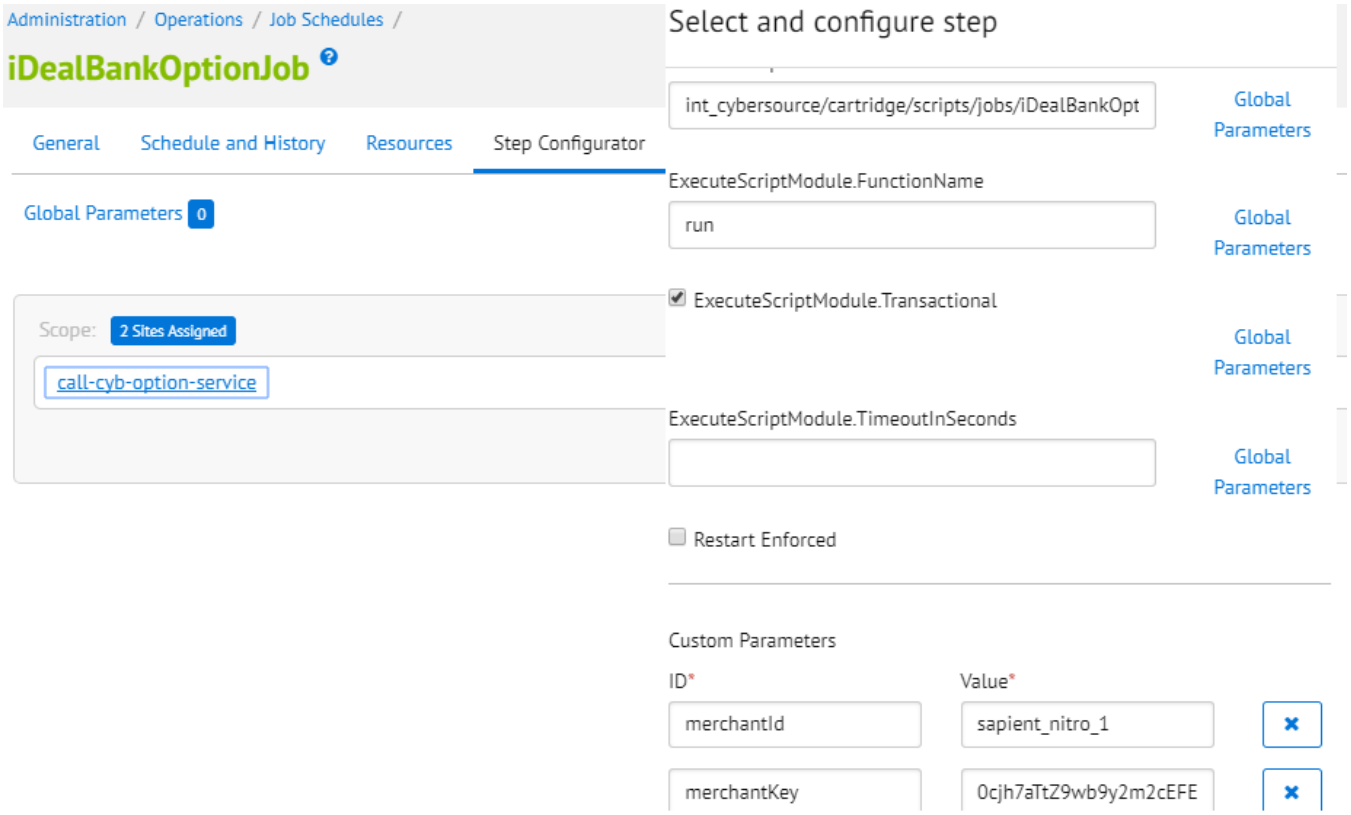
Merchant Notifications POST URL card number digits supported option 2 as shown.



Batch Job for Bank Option List

- Add new batch job to to add merchant defined custom objects for bank options.

Verify the newly added batch jobs for Ideal Bank Option service.
Go to Administration - > Operations -> Job Schedules



Unit Test Services

Use **CYBServicesTesting** controller to test all the services as follows:
CyberSource Services Test Suite is created to gives the facility to the user to execute any of the selected Test Service by providing requested Input and getting the response back on the same interface.
Below is the URL for CyberSource Test Suite:
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-StartServices

Services Test Suite

Test CC Auth
Test Tax
Test Finger Print
Test PA
Test Alipay Initiate Service
Test DAV Check
Test POS
Create Subscription
View Subscription
Update Subscription
Delete Subscription
Test Secure Acceptance Create Token
Test On Demand Payment
Test Sale Service
Test PayPal Authorize Service
Test Refund Service
Test Cancel Service
Test Capture Service
Test Auth Reversal Service
Test Check Status Service

[Note: Modify exports of **Test Services** in CYBServicesTesting.js with https guard before executing the test cases. This activity is common for all test interfaces.]

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

Refer the screen shot below:

```
/*
 * Local methods require guard change below usage
 */

exports.TestCCAuth=guard.ensure(['https'], TestCCAuth);
exports.TestTax=guard.ensure(['https'], TestTax);
exports.TestDAVCheck=guard.ensure(['https'], TestDAVCheck);
exports.TestPA=guard.ensure(['https'], TestPA);
exports.TestFingerprint=guard.ensure(['https'], TestFingerprint);
exports.TestAlipayInitiateService=guard.ensure(['https'], TestAlipayInitiateService);
exports.StartPOS=guard.ensure(['https'], StartPOS);
exports.CreateSubscription=guard.ensure(['https'], CreateSubscription);
exports.ViewSubscription=guard.ensure(['https'], ViewSubscription);
exports.UpdateSubscription=guard.ensure(['https'], UpdateSubscription);
exports.DeleteSubscription=guard.ensure(['https'], DeleteSubscription);
exports.TestSATokenCreate=guard.ensure(['https'], TestSATokenCreate);
exports.TestSATokenCreateResponse=guard.ensure(['https'], TestSATokenCreateResponse);
exports.StartServices=guard.ensure(['https'], StartServices);
exports.TestSaleService=guard.ensure(['https'], TestSaleService);
exports.TestRefundService=guard.ensure(['https'], TestRefundService);
exports.TestAuthorizeService=guard.ensure(['https'],TestAuthorizeService);
exports.TestCancelService=guard.ensure(['https'],TestCancelService);
exports.TestCaptureService=guard.ensure(['https'],TestCaptureService);
exports.TestAuthReversalService=guard.ensure(['https'],TestAuthReversalService);
exports.TestCheckStatusService=guard.ensure(['https'],TestCheckStatusService);
exports.TestPOS=guard.ensure(['https'], TestPOS);
exports.OnDemandPayment=guard.ensure(['https'], OnDemandPayment);
```

Authorize Credit Card

Use and modify the **CYBServicesTesting**-TestCCAuth controller and associated scripts. The unit test controller 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.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestCCAuth

Tax Service

Use and modify the **CYBServicesTesting**-TestTax controller and associated scripts. The script 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. Controller displays all relevant request/response information in an easy to digest manner, to aid the debugging the various response codes and corrected address response.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestTax

Address Verification Service (AVS)

Use and modify the **CYBServicesTesting**-TestCCAuth controller and associated scripts. 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.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestCCAuth

Delivery Address Verification Service (DAV)

To test standalone DAV service, use and/or modify **CYBServicesTesting**-TestDAVCheck and associated scripts. Test data can be customized to simulate various situations that need to be handled.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestDAVCheck

Payment Tokenization

Use the **CYBServicesTesting**-StartSubscription controller 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.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-StartSubscription

Rate Limiting can be added to the My Accounts page, so a merchant can determine the number of cards that can be edited or added.

Device Fingerprint

Call the controller **CYBServicesTesting**-TestFingerprint to test the device Fingerprint Service. A CreditCard Authorization is done and a device fingerprint will be additionally submitted.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestFingerprint

Payer Authentication

Call the controller **CYBServicesTesting**-TestPA to test the Payer Authentication Service.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestPA

Retail POS Authorization Request

Call the controller **CYBServicesTesting**-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.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-StartPOS

Alipay Initiate Request

Call the controller **CYBServicesTesting**-TestAlipayInitiateService to test Alipay Initiate request. Use and modify the mentioned scripts to test initiate request. The end view of the unit test controller 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.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestAlipayInitiatesService

Create Subscription Request

Call the controller**CYBServicesTesting**-CreateSubscription to test Create Subscription request. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter the dummy values printed below the form.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-CreateSubscription

View Subscription Request

Call the controllerCYBServicesTesting-ViewSubscription to test View Subscription request. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter a valid subscription ID.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-ViewSubscription

Update Subscription Request

Call the controllerCYBServicesTesting-UpdateSubscription to test Create Subscription request. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter the dummy values printed below the form.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-UpdateSubscription

Delete Subscription Request

Call the controllerCYBServicesTesting-CreateSubscription to test Create Subscription request. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter a valid subscription ID.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-DeleteSubscription

On Demand Payment Request

Call the controllerCYBServicesTesting-OnDemandPayment to test On Demand Payment request. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter a valid subscription ID witht the amount.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-OnDemandPayment

Check Status Request

Call the controllerCYBServicesTesting-TestCheckStatusService to test Check Status request for Klarna,BanContact,EPS,Giropay,Ideal and,Sofort. The end node of the unit test controlleris a template which displays all relevant request/response information. User will be presented with a form and needs to enter Merchant Reference number,requestID,amount,currency and select the APM from dropdown menu.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestCheckStatusService

Capture Request

Call the controllerCYBServicesTesting-TestCaptureService to test Capture request for PayPal,Klarna,Credit Card and Visa Checkout. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter Merchant Reference number,requestID,amount,currency and select the APM from dropdown menu.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestCaptureService

Auth Reversal Request

Call the controllerCYBServicesTesting-TestAuthReversalService to test Auth reversal request for PayPal,Klarna,Credit Card. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter

Merchant Reference number,requestID,amount,currency and select the APM from dropdown menu.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestAuthReversalService

Sale Request

Call the controllerCYBServicesTesting-TestSaleService to test Sale request for PayPal. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter Merchant Reference number,requestID,amount,currency and enter the APM name.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestSaleService

Authorize Request

Call the controllerCYBServicesTesting-TestAuthorizeService to test Authorize request for PayPal. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter Merchant Reference number,requestID,amount,currency and enter the APM name.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestAuthorizeService

Refund Request

Call the controllerCYBServicesTesting-TestRefundService to test Refund request for PayPal,Klarna,Bancontact,Sofort and, IDeal. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter Merchant Reference number,requestID,amount,currency and select the APM from dropdown menu.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestRefundService

Cancel Request

Call the controllerCYBServicesTesting-TestCancelService to test Cancel request for PayPal. The end node of the unit test controlleris a template which displays all relevant request/response information.User will be presented with a form and needs to enter Merchant Reference number,requestID and enter the APM name.Once the correct information is submitted, the result will be displayed.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestCancelService

Secure Acceptance Web / Mobile Create Token Request

Before TESTING please complete the profile setup for service to work refer section **Secure Acceptance profile setup** for more details
Call the controller CYBServicesTesting-TestSATokenCreate to test the secure acceptance redirect creates token Service. This renders a secure acceptance hosted page at cybersource having details of card options to choose to enter/select values. The service response is shown after the pay button is clicked. The field's label turns to red colored font if the field was mandatory. The response arrived to controllerCYBServicesTesting-TestSATokenCreateResponse which displays the service result fields.
https:// <Sandbox Name>/on/demandware.store/Sites-SiteGenesis-Site/default/CYBServicesTesting-TestSATokenCreate

Apple Pay

How to test on Demandware Storefront

To test ApplePay on Demandware site, following files need to be updated:

Script – applepay.js

Update the file with below changes:

```
var Status = require('dw/system/Status');
var ApplePayHookResult = require('dw/extensions/applepay/ApplePayHookResult');
```

Add new method getRequest at the end of file

```
exports.getRequest = function (basket, request) {
if (request.shippingContact) {
// convert country code from lower case to upper case
      request.shippingContact.countryCode =
        request.shippingContact.countryCode.toUpperCase();
  }
return new ApplePayHookResult(new Status(Status.OK), null);
};
```

hooks.json

Add hook for applepay at the end of file present at <core SG cartridge>/cartridge/script

```
{
                        "name": "dw.extensions.applepay.paymentAuthorized.authorizeOrderPayment",
                        "script": "./checkout/applepay.js"
                },
                {
                        "name": "dw.extensions.applepay.getRequest",
                        "script": "./checkout/applepay.js"
```

```
}
```

Controller – BASIC_CREDIT.js

Update Handle() function with the code below

```
var CybersourceConstants = require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
/**
*Verifiesacreditcardagainstavalidcardnumberandexpirationdateandpossiblyinvalidatesinvalidformfields.
*Iftheverificationwassuccessfulacreditcardpaymentinstrumentiscreated.
*/
function Handle(args) {
var cart = Cart.get(args.Basket);
var creditCardForm = app.getForm('billing.paymentMethods.creditCard');
var PaymentMgr = require('dw/order/PaymentMgr');

var CommonHelper = require('int_cybersource/cartridge/scripts/helper/CommonHelper');
if
(session.forms.billing.paymentMethods.selectedPaymentMethodID.value.equals(CybersourceConstants.METHOD_Apple
Pay)) {
      Transaction.wrap(function () {
         CommonHelper.removeExistingPaymentInstruments(cart);
         var paymentInstrument = cart.createPaymentInstrument('DW_APPLE_PAY', cart.getNonGiftCertificateAmount());
      });
            return {success:true};
   }else if
(session.forms.billing.paymentMethods.selectedPaymentMethodID.value.equals(CybersourceConstants.METHOD_Andro
idPay)) {
      Transaction.wrap(function () {
         CommonHelper.removeExistingPaymentInstruments(cart);
         var paymentInstrument = cart.createPaymentInstrument('DW_ANDROID_PAY',
cart.getNonGiftCertificateAmount());
      });
            return {success:true};
   }
var cardNumber = creditCardForm.get('number').value();
var cardSecurityCode = creditCardForm.get('cvn').value();
var cardType = creditCardForm.get('type').value();
var expirationMonth = creditCardForm.get('expiration.month').value();
var expirationYear = creditCardForm.get('expiration.year').value();
var paymentCard = PaymentMgr.getPaymentCard(cardType);
```

Update Authorize() function with the code below

```
/**
*AuthorizesapaymentusingacreditcardThepaymentisauthorizedbyusingtheBASIC_CREDITprocessor
*onlyandsettingtheordernoasthetransactionID.CustomizationsmayuseotherprocessorsandcustomLogictoauthorizecreditcardpayment.
*/
function Authorize(args) {
var orderNo = args.OrderNo;
var paymentInstrument = args.PaymentInstrument;
```

```
var paymentProcessor =
PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();

   Transaction.wrap(function () {
      paymentInstrument.paymentTransaction.transactionID = orderNo;
      paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;
   });
            if (session.forms.billing.paymentMethods.selectedPaymentMethodID.value.equals('DW_APPLE_PAY')) {
                        return {review:true};
            }
return {authorized: true};
}
```

[Note: this change is for testing purpose only]

Rest Interface Testing

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

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

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

4. Pass below JSON when Payload test data available

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

5. Pass below JSON when Network Token test data available

JSON KEY	JSON VALUE
orderID	The order ID of ApplePay order object created during checkout journey of ApplePay

networkToken	Network Token returned by ApplePay for PSP authorization (Max length 20 character)
cardType	Card Type returned by ApplePay for PSP authorization. Supported types visa/mastercard/amex
tokenExpirationDate	Network Token Expiration Date returned by ApplePay for PSP authorization. Format YYMMDD
cryptogram	Cryptogram encoded form (max length 40 character)

6. Test the Success response JSON

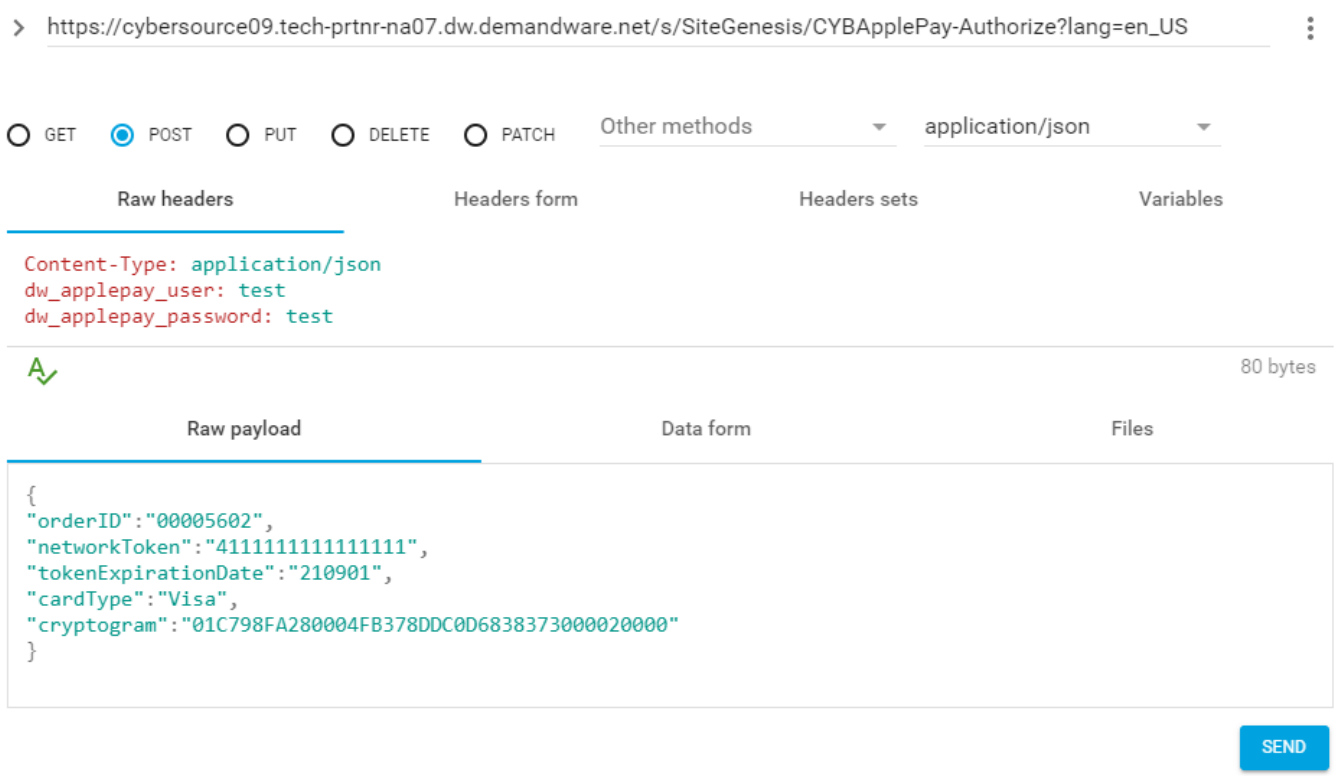
JSON KEY	JSON VALUE
TRANSACTION_RESULT	Below json key-value pairs
DECISION	Possible values ACCEPT | REVIEW | REJECT | ERROR | CANCEL
REASON_CODE	ReasonCode
REQUEST_ID	RequestID
REQUEST_TOKEN	RequestToken
AUTHORIZATION_AMOUNT	AuthorizationAmount
AUTHORIZATION_CODE	AuthorizationCode
AUTHORIZATION_REASON_CODE	AuthorizationReasonCode
DAV_REASON_CODE	DAVReasonCode
RAW_SERVICE_RESPONSE	Entire service response in form of JSON

7. Test the Validation/Failure response JSON

JSON KEY	JSON VALUE
ERROR_CODE	Validation failure error code of interface
ERROR_MSG	Validation failure message of interface

Sample ApplePay InterfaceJSON Request /Response format

Interface 1: Request with network Token and Cryptogram data:



Android Pay

How to test on Demandware Storefront

To test AndoridPay on Demandware site, following files need to be updated:

Controller – BASIC_CREDIT.js

Include CybersourceConstants in API include section

```
/* Script Modules */
var app = require('~/cartridge/scripts/app');
var CybersourceConstants =
require('int_cybersource/cartridge/scripts/utils/CybersourceConstants');
```

Update Handle() function with the code below

```
/**
*Verifiesacreditcardagainstavalidcardnumberandexpirationdateandpossiblyinvalidatesinvalidformfields.
*Iftheverificationwassuccessfulacreditcardpaymentinstrumentiscreated.
*/
function Handle(args) {
var cart = Cart.get(args.Basket);
var creditCardForm = app.getForm('billing.paymentMethods.creditCard');
var PaymentMgr = require('dw/order/PaymentMgr');

var CommonHelper = require('int_cybersource/cartridge/scripts/helper/CommonHelper');
```

```
 if
(session.forms.billing.paymentMethods.selectedPaymentMethodID.value.equals(CybersourceConstants.METHOD_Apple
Pay)) {
      Transaction.wrap(function () {
         CommonHelper.removeExistingPaymentInstruments(cart);
         var paymentInstrument = cart.createPaymentInstrument('DW_APPLE_PAY', cart.getNonGiftCertificateAmount());
      });
            return {success:true};
   }else if
(session.forms.billing.paymentMethods.selectedPaymentMethodID.value.equals(CybersourceConstants.METHOD_Andro
idPay)) {
      Transaction.wrap(function () {
         CommonHelper.removeExistingPaymentInstruments(cart);
         var paymentInstrument = cart.createPaymentInstrument('DW_ANDROID_PAY',
cart.getNonGiftCertificateAmount());
      });
            return {success:true};
   }
var cardNumber = creditCardForm.get('number').value();
var cardSecurityCode = creditCardForm.get('cvn').value();
var cardType = creditCardForm.get('type').value();
var expirationMonth = creditCardForm.get('expiration.month').value();
var expirationYear = creditCardForm.get('expiration.year').value();
var paymentCard = PaymentMgr.getPaymentCard(cardType);
```

Update Authorize() function with the code below

```
/**
*Authorizesapaymentusingacreditcard.ThepaymentisauthorizedbyusingtheBASIC_CREDITprocessor
*onlyandsettingtheordernoasthetransactionID.CustomizationsmayuseotherprocessorsandcustomLogic
*logictoauthorizecreditcardpayment.
*/
function Authorize(args) {
var orderNo = args.OrderNo;
var paymentInstrument = args.PaymentInstrument;
var paymentProcessor =
PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();

   Transaction.wrap(function () {
      paymentInstrument.paymentTransaction.transactionID = orderNo;
      paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;
   });
            if
(session.forms.billing.paymentMethods.selectedPaymentMethodID.value.equals('DW_METHOD_AndroidPay')) {
            return {review:true};
   }
return {authorized: true};
}
```

[Note: this change is for testing purpose only]

Rest Interface Testing

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

8. Install the REST client on machine or browser
9. Hit the secure End Point URL as POST request having merchant site URL for "Cybersource_ApplePay-Authorize" [example: https://<merchant sandbox>/on/demandware.store/Sites-<merchant site>-Site/default/Cybersource_ApplePay-Authorize]
10. Add key-value pairs in header for credentials

HEADER KEY	HEADER VALUE
dw_androidpay_user	User is configured by merchant in demandware platform under site preferences
dw_androidpay_password	Password is configured by merchant in demandware platform under site preferences.
Content-Type	application/json

11. Pass below JSON when Payload test data available

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

12. Pass below JSON when Network Token test data available

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

13. Test the Success response JSON

JSON KEY	JSON VALUE
TRANSACTION_RESULT	Below json key-value pairs
DECISION	Possible values ACCEPT | REVIEW | REJECT | ERROR | CANCEL
REASON_CODE	ReasonCode
REQUEST_ID	RequestID

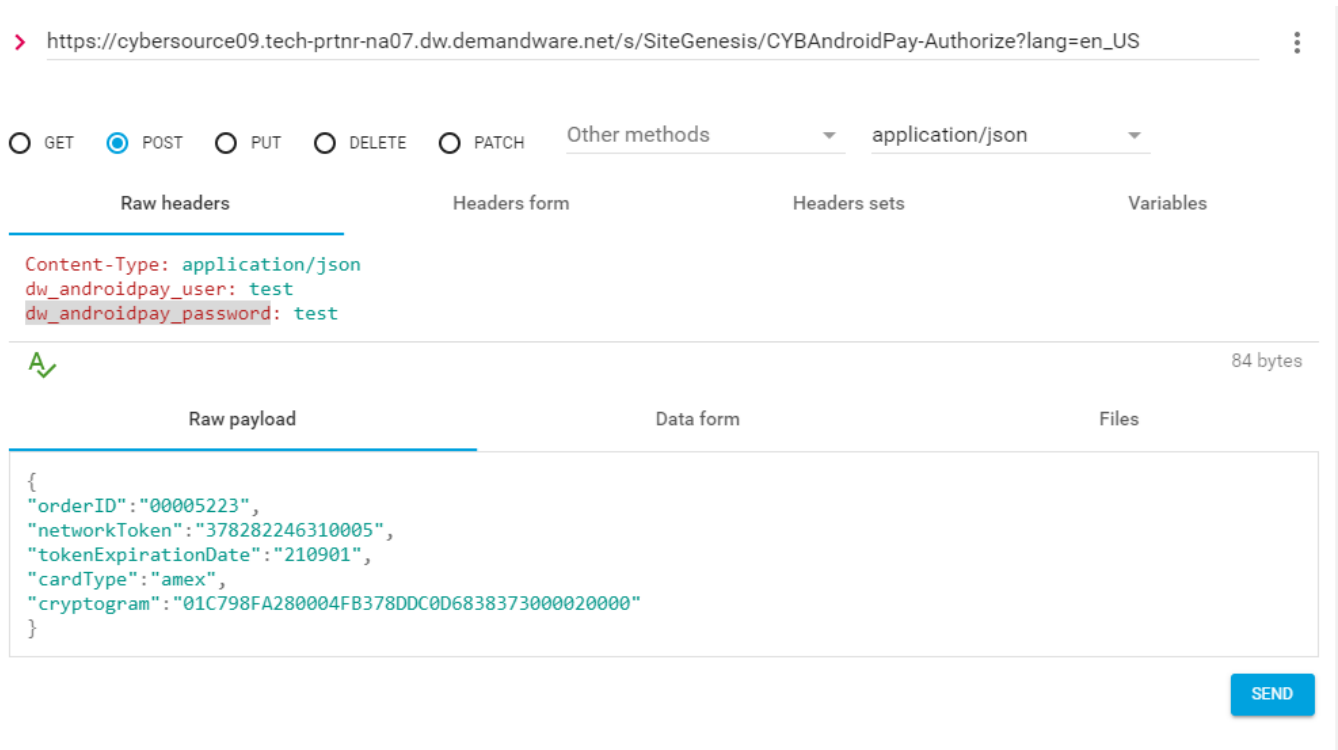
REQUEST_TOKEN	RequestToken
AUTHORIZATION_AMOUNT	AuthorizationAmount
AUTHORIZATION_CODE	AuthorizationCode
AUTHORIZATION_REASON_CODE	AuthorizationReasonCode
SUBSCRIPTION_ID	Subsciption id in case of tokenization is enabled in BM
DAV_REASON_CODE	DAVReasonCode
RAW_SERVICE_RESPONSE	Entire service response in form of JSON

14. Test the Validation/Failure response JSON

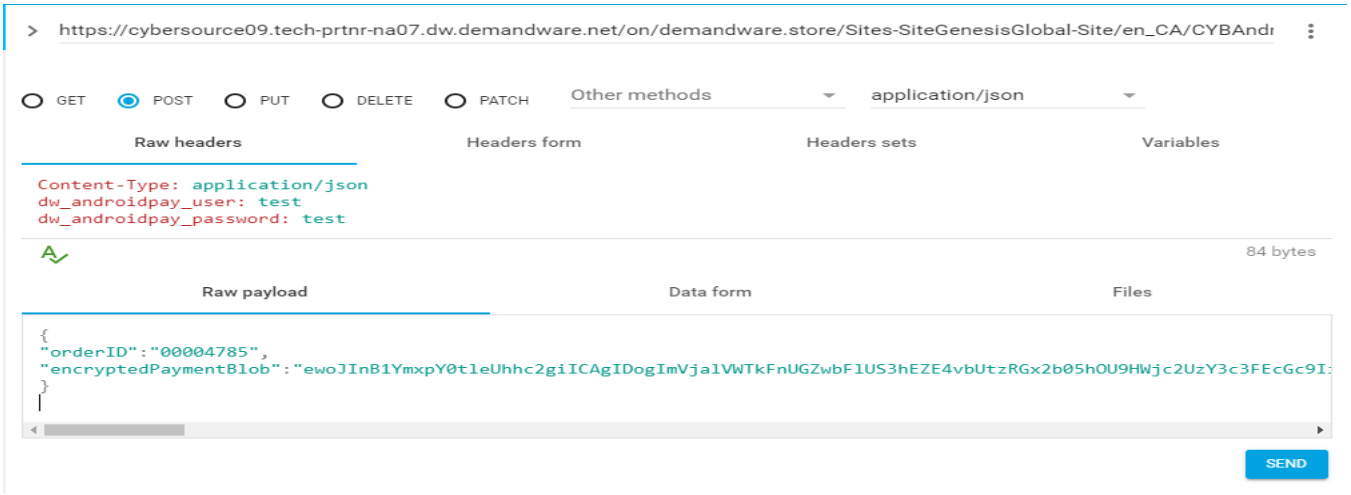
JSON KEY	JSON VALUE
ERROR_CODE	Validation failure error code of interface
ERROR_MSG	Validation failure message of interface

Sample AndroidPay InterfaceJSON Request /Response format

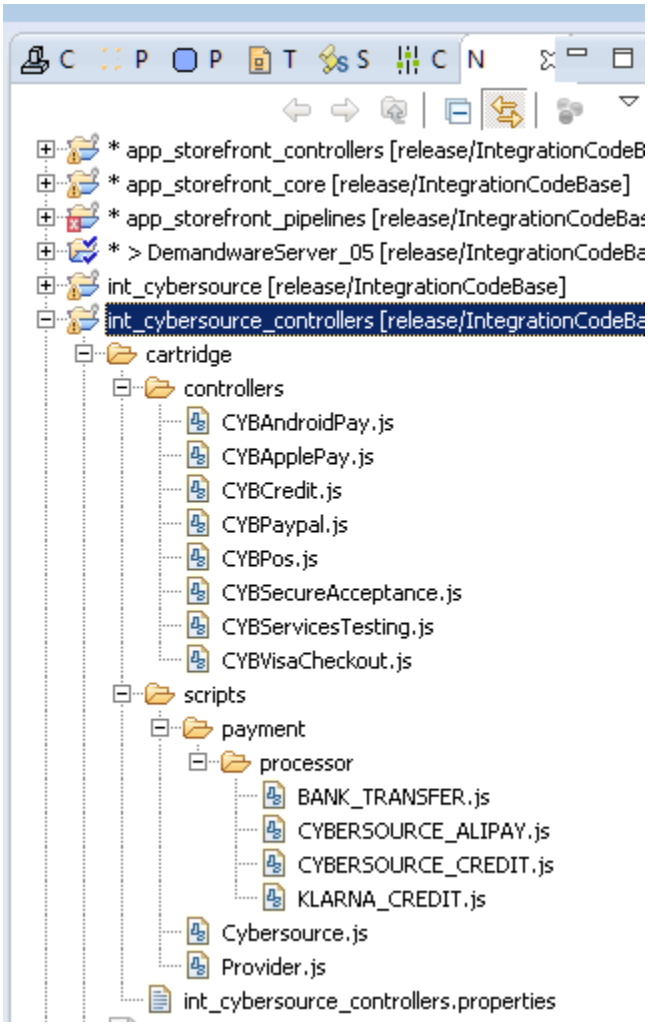
Interface 1: Request with network Token and Cryptogram data:



Interface2: Request with encrypted payment BLOB data.



Cartridges Structure and Reference



Typical Project Plan

Roles, Responsibilities

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

Typical Efforts and Timelines

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

CyberSource Service	Level of Effort (LOE)	Dependencies
Initial Cartridge Setup	**0.5**– Person Day List of tasks involved: • Add CyberSource Cartridges to the project • Import Configuration files as specified in configuration section	• Cartridge is available
Authorize Credit Card	**0.5**– Person Day List of tasks involved: • Integrate CyberSource-AuthorizeCreditCard controller with COPlaceOrder.	• Merchant ID and Key is established for the client. • Site Preferences for authorization configured with above ID and Key.
Device Fingerprint (as addition to Authorize Credit Card)	**0.5** Person Day	• Enable Device Fingerprint, set Organization ID • Add include at billing page.
Address Verification Service (AVS)*	**0.5**– Person Day	• Initial Cartridge Setup
Delivery Address Verification (DAV)*	**0.5**– Person Day	• Initial Cartridge Setup
Decision Manager	**0.5**– Person Day	• Access to Decision Manager. • Business rules are defined. • Order status notification URL pointing to Cybersource-New Decision 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 • Handle error scenarios in merchant specific ways
Alipay Integration on Payment Page	**1.0**– Person Day	• Initial Cartridge setup • Update CoPlaceOrder-HandlePayments • Handle error scenarios in merchant specific ways
Visa Checkout	**0.5**– Person Day List of tasks involved: Integrate VISACHECKOUT controller and merchant site specific button injection on minicart, cart and billing page.	• Merchant ID and Key is established for the client. • Visa checkout account setup required • Site Preferences for authorization configured with above ID and Key.
Apple Pay	**2**– Person Day List of tasks involved: Choose and decide the integration mechanism with apple pay interface.	• Site Preferences for heeder authentication exposed.
Secure Acceptance (Redirect/Iframe/Silent post)	**0.5**– Person Day (1 out of 3 methods) List of tasks involved: Integrate SECURE_ACCEPTANCE Controller	• Cartridge setup • Configure profile and URL in Cybersource • Site preference configuration in Demandware Business manager • config
Klarna	0.5 - Person Day Integrate KLARNA_CREDIT controller, changes on billing and summary pages for Klarna	• Cartridge setup • Site preference configuration in business manager • Merchant Id and Key for specific country and currency

Bank Transfer(SOFORT,BANCONTACT, EPS, GIROPAY, IDEAL)	0.5- Person Day Integrate BANK_TRANSFER controller, changes for billing page for Bank Transfer to display BIC field or bank list	• Cartridge setup • Site preference configuration in business manager • Merchant Id and Key for IDEAL method
PayPal(Express, credit, billing agreement)	1- Person Day Integrate PAYPAL_EXPRESS and PAYPAL_CREDIT controller, changes on mini cart, cart and billing pages	• Cartridge setup • Site preference configuration in business manager
Andriod Pay	0.5 – Person Day Integrate BASIC_CREDIT controller, changes on billing page	• Cartridge setup • Site preference configuration in business manager

*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 controller methods on production instances methods are made private. It is advised to make following controller function export guard to be removed before pushing code to production instances.

CYBServicesTesting-TestCCAuth
CYBServicesTesting- TestAlipayInitiateService
CYBServicesTesting- TestAlipayCheckStatusService
CYBServicesTesting- TestPaypalCaptureService
CYBServicesTesting-TestTax
CYBServicesTesting-TestDAVCheck
CYBServicesTesting-TestPA
CYBServicesTesting-TestFingerprint
CYBServicesTesting -StartSubscription
CYBServicesTesting -CreateSubscription
CYBServicesTesting -ViewSubscription
CYBServicesTesting -UpdateSubscription
CYBServicesTesting -DeleteSubscription

CYBServicesTesting -OnDemandPayment
CYBServicesTesting-StartPOS
CYBServicesTesting- TestSATokenCreate
CYBServicesTesting- TestSaleService
CYBServicesTesting- TestPayPalAuthorizeService
CYBServicesTesting- TestRefundService
CYBServicesTesting- TestCancelService
CYBServicesTesting- TestAuthReversalService
CYBServicesTesting- TestCheckStatusService

Known Issues

1. In case 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.
2. 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.

3. There is an issue with Klarna session and authorization service in accepting value of tax rate field upto 4 or more decimal places. Klarna service accepts only tax rate value upto 2 decimal points and service is returning REJECT decision if tax rate exceed 2 decimal places.

CyberSource document links

1. http://www.cybersource.com/support_center/implementation/testing_info/simple_order_api/General_testing_info/soapi_general_test.html
2. http://www.cybersource.com/support_center/support_documentation/quick_references/view.php?page_id=422
3. http://apps.cybersource.com/library/documentation/dev_guides/CC_Svcs_SO_API/Credit_Cards_SO_API.pdf - Page 163 - Appendix C.
4. http://apps.cybersource.com/library/documentation/dev_guides/Getting_Started/Getting_Started_Advanced.pdf
5. http://www.cybersource.com/support_center/support_documentation/quick_references/
6. http://apps.cybersource.com/library/documentation/dev_guides/Payer_Authentication_IG/20090928_Payauth_IG.pdf
7. http://apps.cybersource.com/library/documentation/dev_guides/Payer_Authentication_IG/html/
8. http://apps.cybersource.com/library/documentation/dev_guides/Verification_Svcs_IG/20091012_Verification_IG.pdf
9. http://www.cybersource.com/support_center/support_documentation/services_documentation/tax.php
10. http://apps.cybersource.com/library/documentation/dev_guides/Tax_IG/Tax_Guide.pdf
11. http://apps.cybersource.com/library/documentation/dev_guides/Retail_SO_API/Retail_SO_API.pdf
12. http://apps.cybersource.com/library/documentation/dev_guides/AliPayDom/AliPay_Dom_SO_API.pdf
13. http://apps.cybersource.com/library/documentation/dev_guides/AliPayInt/AliPay_Int_SO_API.pdf
14. http://apps.cybersource.com/library/documentation/dev_guides/apple_payments/SO_API/Apple_Pay_SO_API.pdf
15. http://apps.cybersource.com/library/documentation/dev_guides/Secure_Acceptance_WM/Secure_Acceptance_WM.pdf
16. http://apps.cybersource.com/library/documentation/dev_guides/Secure_Acceptance_SOP/Secure_Acceptance_SOP.pdf
17. http://apps.cybersource.com/library/documentation/dev_guides/VCO_SO_API/Visa_Checkout_SO_API.pdf
18. http://apps.cybersource.com/library/documentation/dev_guides/apple_payments/getting_started/Getting_Started.pdf
19. http://apps.cybersource.com/library/documentation/dev_guides/tokenization_SO_API/Tokenization_SO_API.pdf
20. http://apps.cybersource.com/library/documentation/dev_guides/OnlineBankTransfers_SO_API/OnlineBankTransfers_SO_API.pdf
21. http://www.cybersource.com/support_center/support_documentation
22. https://developer.paypal.com/docs/integration/direct/express-checkout/integration-jsv4/
23. https://developer.paypal.com/demo/checkout/#/pattern/client
24. https://www.cybersource.com/products/payment_processing/android_pay/
25. https://www.cybersource.com/developers/integration_methods/apple_pay/

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 controller to remove unnecessary / redundant tax requests to reduce tax service charges.
1.0.0.4	12/18/2012	Updated Tax controller 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 methodsetGrossPrice for taxation
2.1.0	10/04/2013	V.me Clickjacking changes added
2.1.1	11/04/2013	Removed unsued code from controller
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 As Wrapper to call controller flows, defects fixes and change request** **Removed V.me support**
17.1	**01/02/2017**	**Removed:** • **BML** • **Removed PayPal Express support** **Added :** • **Visa Checkout** • **Secure Acceptance Web/Mobile [Redirect/Iframe]** • **Secure Acceptance Silent Order Post** • **Apple Pay REST Interface**
17.2	**09/01/2017**	**Added :** • **Klarna** • **Bank Transfer** • **PayPal Credit** • **PayPal Express** • **PayPal Credit** • **PayPal Billing Agreement** • **Android Pay** • **Check Status Service job** • **IDeal Option Job** • **Cartridge folder structure changes** • **File extension changes from .ds to .js** • **Controllers name changed** • **Removed/Repurposed unwanted files**
19.3.0	**07/26/2019**	**Update 3DS to version 2.0, utilizing Cardinal Cruise.**
19.3.4	**11/19/2019**	**Klarna payment method and replacing conversion detail report to REST Api.**
19.4.0	**02/25/2020**	**Bug fixes on Converstion Detail report**
19.4.1	**03/17/2020**	- Rate Limiting added to the My Accounts page, so a Merchant can determine the number of cards that can be edited or added. - Update CyberSource WSDL to support Apache CXF v3 upgrade.

		- Bug fix on 3D Secure 2.0. When this service was called repeatedly it got switched to 3D Secure 1.0. Fix is to stop that from happening.
19.5.0	**05/20/2020**	-Add WeChat Pay payment method.
19.5.1	**11/30/2020**	-Improved security on accessing and modifying sensitive fulfillment-related actions on an order (e.g., order acceptance, canceling etc.).
19.5.2	**12/5/2020**	N/A
19.5.3	**12/30/2020**	-Support 3ds for French processor.
21.1.0	**3/15/2021**	- Support Credit card authorization using Flex microform -Removed the code that supports Device fingerprint using the Flash approach. Now the cartridge will only support the JavaScript approach for Device fingerprint. -Removed logger.fatal from cartridge code. (GitHub issue#3, 2) -Removed reference of base cartridge. (GitHub issue#11) -We removed hard coding of host URL from Decision manager job. (GitHub issue#49) -We fixed issue in common.js. (GitHub#11) -We Converted .ds files to .js files. -We replaced em or rems with px. -We replaced importpackage() with require(). -We removed inline styles and separate them out into css files. -Convert the SG metadata so that it can be imported via Site Import & Export instead of separately. -Replaced the deprecated webreference package with webreference2 package.
21.2.0	**8/27/2021**	-We have updated credit card form page in the My Account page with Flex Microform v0.11 implementation. -We have updated the cartridge to make it compatible with Salesforce B2C Commerce release 21.2.