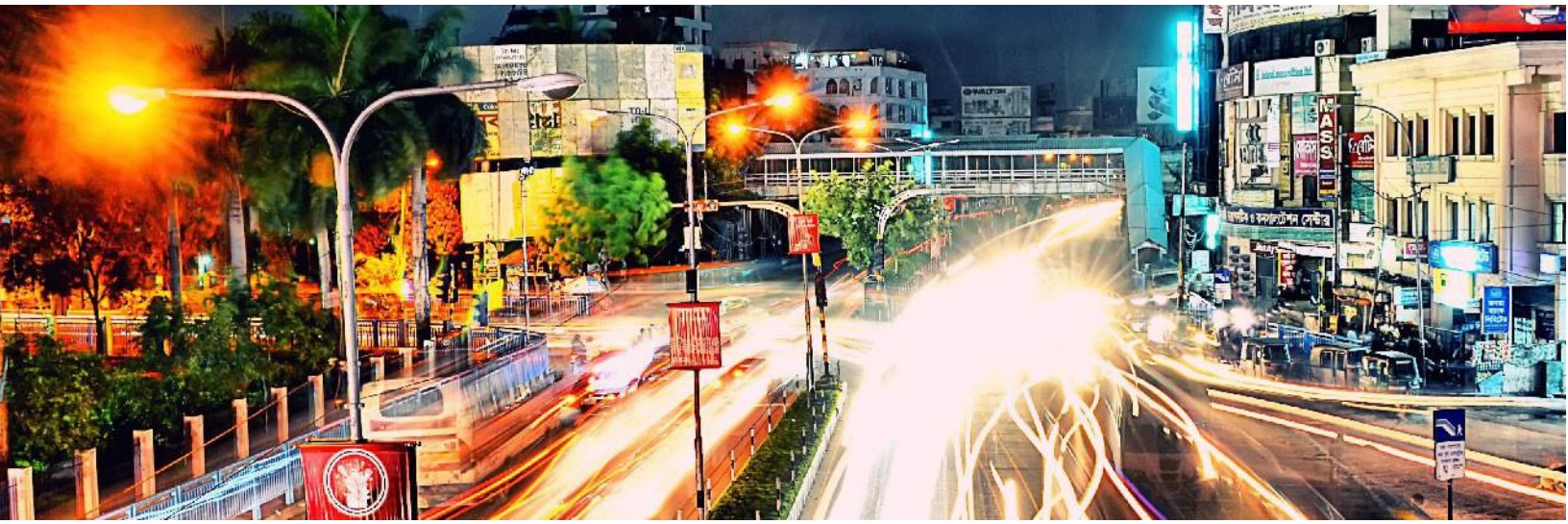
Cybersource

Cybersource Cartridge Controllers Integration Guide



cybersource
A Visa Solution

Cybersource Contact Information

For general information about our company, products, and services, go to http://www.cybersource.com.

For sales questions about any Cybersource Service, email sales@cybersource.com or call 650-432-7350 or 888-330-2300 (toll free in the United States).

For support information about any Cybersource Service, visit the Support Center: http://www.cybersource.com/support

Copyright

Restricted Rights Legends

Trademarks

Confidentiality Notice

This document is furnished to you solely in your capacity as a client of Cybersource and as a participant in the Visa payments system.

By accepting this document, you acknowledge that the information contained herein (the "Information") is confidential and subject to the confidentiality restrictions contained in Visa's operating regulations and/or other confidentiality agreements, which limit your use of the Information. You agree to keep the Information confidential and not to use the Information for any purpose other than its intended purpose and in your capacity as a customer of Cybersource or as a participant in the Visa payments system. The Information may only be disseminated within your organization on a need-to-know basis to enable your participation in the Visa payments system. Please be advised that the Information may constitute material non-public information under U.S. federal securities laws and that purchasing or selling securities of Visa Inc. while being aware of material non-public information would constitute a violation of applicable U.S. federal securities laws.

Release: April 2023
Version: 23.1.0

Table of Contents

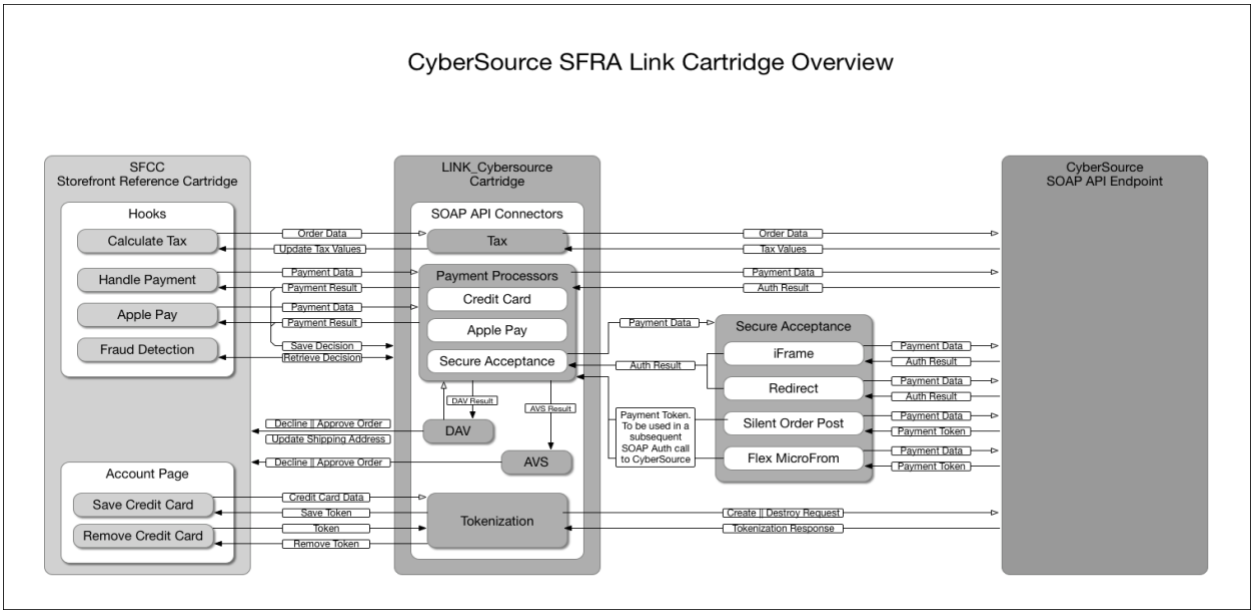
1. Introduction

- **Description:** Cybersource, a Visa solution, is the only global, modular payment management platform built on secure Visa infrastructure with the payment reach and fraud insights of a massive $500B+ global processing network. You can find out more about what Cybersource does here.
- **Categories:** Payment Processing, Fraud Detection, Address Validation, Tax Computation
- **Version:** 23.1.0
- **Compatibility:** This version of the Cybersource cartridge is not compatible with the latest versions of SG. This version is compatible with Salesforce B2C Commerce 21.2 release.

Contact

- Global Partner Solutions - CS: GlobalPartnerSolutionsCS@visa.com

2. CyberSource SG Cartridge Architecture



---

3. Install the Cartridge and Setup Workspace

Step 1: Install the Cartridge

1. Cybersource Controllers Integration Cartridge can be installed from Salesforce Commerce Cloud's marketplace link.

Step 2: Setup workspace

1. Create a folder “CyberSource” in your workspace and place the cartridge (**app_storefront_core, app_storefront_controllers, int_cybersource, int_cybersource_controllers**) downloaded from Marketplace.

2. If using VSCode, install the extension Prophet Debugger link or any other SFCC extension and include below in dw.json ().

```
{
    "hostname": "your-sandbox-hostname.demandware.net",
    "username": "yourlogin",
    "password": "yourpwd",
    "version": "version_to_upload_to",
    "cartridge": ["app_storefront_controllers”, “app_storefront_core”
,“int_cybersource”, “int_cybersource_controllers”]

}
```

NOTE: If you are using different IDE, refer respective developer guide to setup the workspace.

Step 3: Build and Upload the code

Prerequisite: Install node under "Cybersource" folder.

- cd into the sitegenesis directory.
- npm install:
- Build js and css with this command : npm run build

This assumes that you already have npm installed on your command line. If not, please install node first. If you encounter an error, please try and address that first, either by Googling or contacting us.

- Install either gulp or grunt (see below).

Gulp

Install gulp globally

```
npm install -g gulp
```

Grunt

Install the grunt command line tools

```
npm install -g grunt-cli
```

Now that you have gulp (or grunt) and its dependencies installed, you can start using it in your workflow.

SCSS

Before authoring SCSS, make sure to check out the README in the app_storefront_core/cartridge/scss directory.

---

4. Configure the Cartridge

Prerequisite

If you are new to Cybersource, and would like to start using Cybersource Cartridge quickly, begin by signing up for a Sandbox Account.

You will also need to create an API Key and API Shared Secret Key that you can use to authenticate requests to our sandbox server. Follow same steps to generate Production key and shared secret.

Step 1: Setup Cartridge Path

To set up the cartridge path:

1. In the Business Manager, go to **Administration > Sites > Manage Sites > [yourSite] > Settings**
2. For the Cartridges, enter **int_cybersource, int_cybersource_controllers** and **app_storefront_core, app_storefront_controllers,** select **Apply**

Step 2: Upload metadata

Cybersource cartridge installed from Market place comes with metadata to import.

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.

On successful import, it creates following metadata:

- Site Preferences (Cybersource, Cybersource_SecureAcceptance, Cybersource_Paypal, Cybersource_GooglePay, Cybersource_Klarna, Cybersource_VisaCheckout, Cybersource_BankTransfer, Cybersource_PayerAuthentication, Cybersource_WeChat, Cybersource_Tokenization, Cybersource_DecisionManager, Cybersource_TaxConfiguration, Cybersource_DeliveryAddressVerification, Cybersource_DeviceFingerprint, Cybersource_Alipay)
- Service (cybersource.conversiondetailreport, cybersource.soap.transactionprocessor.generic, cybersource.http.flextoken)
- Payment Processor (KLARNA_CREDIT, CYBERSOURCE_ALIPAY, BANK_TRANSFER, CYBERSOURCE_WECHAT)

- Payment Method
- Job (CyberSource: Decision Manager Order Update)

Cybersource Core (Required)

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/site_genesis_meta/meta.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource Core** and set values for the following parameters:

Field	Description
Enable Cybersource Cartridge	Enable or disable Cybersource Cartridge. If disabled none of the Cybersource services are invoked
Cybersource Merchant ID	Cybersource Merchant ID
CyberSource Merchant Key	Cybersource SOAP Key. Follow the link for directions.
CyberSource Endpoint	Select Test(Test) or Production(Production)
CyberSource Developer ID	Unique identifier generated by Cybersource for System Integrator

Services (Required)

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/site_genesis_meta /meta/Payment-Services.xml"** in Business Manager (**Administration > Operations > Import & Export**).

Step 2: Go to **Administration > Operations > Services**

Step 3: Make sure service with name **cybersource.soap.transactionprocessor.generic** exist.

Payment Processor (Required)

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/site_genesis_meta /sites/yourSiteID/payment-processors.xml"** in Business Manager (**Merchant Tools > Ordering > Import & Export**).

---

5. Configure the Payment method

Payment Methods

1. Credit Card Authorization
   1.1. Secure Acceptance Hosted Checkout – iFrame
   1.2. Secure Acceptance Redirect
   1.3. Secure Acceptance Checkout API
   1.4. Secure Acceptance Flex MicroForm
   1.5. Direct Cybersource SOAP API
2. Apple Pay
3. PayPal
4. Google Pay
5. Visa SRC
6. Bank Transfer
7. Alipay
8. Klarna
9. WeChat Pay

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()).getPaymentProcesso
    r() === 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(CybersourceC
onstants.METHOD_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(PaymentInstru
ment.METHOD_CREDIT_CARD)) {
cart.removePaymentInstruments(cart.getPaymentInstruments(CybersourceConstants.METHOD_PAYP
AL));
        } else if
(app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstru
ment.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 .saIframeCCToken .field-
wrapper').val(data.selectedCardID);

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

    }
    });

}
```

Update "#creditCardList" on change function

- Update the method 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',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>


        <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="ch
ecked"</isif> />

    <input id="is-${radioID}" type="radio" class="input-radio <isif
condition="${paymentMethodType.object.paymentProcessor.ID.equalsIgnoreCase("bank_tra
nsfer") == true }">bank-transfer</isif>"
name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlNam
e}" value="${paymentMethodType.htmlValue}" <isif
condition="${paymentMethodType.value ==
pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlValue}">checke
d="checked"</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.creditcardlistexpdate','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="${dw.order.PaymentMgr.getPaymentMethod(paymentInstr.paymentMethod).ID==Cybersour
ceConstants.METHOD_SA_IFRAME}">
            <isset name="summaryaction" value="${URLUtils.https('COSummary-SubmitOrder')}"
```

```
scope="page" />
                <isset name="isIFrame" value="${true}" scope="page" />
            <iselseif
condition="${CybersourceConstants.KLARNA_PAYMENT_METHOD.equals(dw.order.PaymentMgr.get
PaymentMethod(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;
            }
        }
```

1. Credit Card Authorization

Overview

The CC Auth service is integrated via the SG OOTB dynamically generated app.payment.processor.cybersource_credit hook. The cybersource_credit hook is registered in the hooks.json file with script ./cartridge/scripts/hooks/payment/processor/cybersource_credit. This script acts as a wrapper to the core CyberSource Authorization code. Behind this wrapper, an API request is constructed, sent to CS, and the response parsed. In the case of a successful authorization (response code 100), the hook returns a JSON object without an error. All other response codes received result in an error being present in the return object, triggering the storefront to display an error message, and not create the order. Actions taken when making the Authorization call are as follows:

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 a credit card authorization request.
4. If DAV is enabled, set up DAV business rules, as needed.
5. Set up AVS if enabled.
6. Make the service call to CyberSource via the SOAP API.
7. If Delivery Address Verification is enabled, then: a. Capture pertinent DAV result information & DAV Reason Code. Update shipping address if a suggestion was returned and the ‘CS DAV Update Shipping Address With DAV Suggestion’ site preference is enabled. 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. Capture Fraud response in a session variable to be handled later.
11. Validate authorization reason code and set corresponding values, based on Auth response code.

Implementation

Cybersource Cartridge supports the following ways of processing Credit Card a. Secure Acceptance Hosted Checkout – iFrame b. Secure Acceptance Redirect c. Secure Acceptance Checkout API d. Secure Acceptance Flex MicroForm e. Direct Cybersource Payment API

Prerequsite

In the Business Manager, go to **Merchant Tools > Ordering > Payment Methods** and select **CREDIT_CARD**. And in **CREDIT_CARD details**, double check if **Payment Processor** = **"CYBERSOURCE_CREDIT"**

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_CREDIT"
        },
```

1.1. To Setup Secure Acceptance Hosted Checkout – iFrame

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta /meta/Cybersource_SecureAcceptance.xml**" in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_SecureAcceptance** and set values for the parameter:

Field	*Description*	*Value to Set*
CsSAType	Secure Acceptance Type	SA_IFRAME
SA_Iframe_ProfileID	Secure Acceptance Iframe Profile ID. Follow **"Creating a Hosted Checkout Profile"** step from SA Guide url	
SA_Iframe_SecretKey	Secure Acceptance Iframe secret key. Follow this link.	
SA_Iframe_AccessKey	Secure Acceptance Iframe Access Key	
CsSAIframetFormAction	CyberSource secure acceptance Iframe form action	
CsSAOverrideBillingAddress	CyberSource Secure Acceptance Override Billing Address	
CsSAOverrideShippingAddress	CyberSource Secure Acceptance Override Shipping Address	
CsCvnDeclineFlags	CyberSource Ignore CVN Result (CVN)	
CsTransactionType	Select Auth/Sale transaction	

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 ${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;
}
}
```

1.2. To Setup Secure Acceptance Redirect

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or
import **"metadata/site_genesis_meta/meta/Cybersource_SecureAcceptance.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_SecureAcceptance** and set values for the parameter:

Field	***Description***	***Value to Set***
CsSAType	Secure Acceptance Type	SA_REDIRECT
SA_Redirect_ProfileID	Secure Acceptance Redirect Profile ID. Follow ****"Creating a Hosted Checkout Profile"**** step from SA Guide url.	
SA_Redirect_SecretKey	Secure Acceptance Redirect Secret Key. Follow this link.	
SA_Redirect_AccessKey	Secure Acceptance Redirect Access Key. Get the access key from above step.	
CsSARedirectFormAction	CyberSource secure acceptance redirect form action.	
CsSAOverrideBillingAddress	CyberSource Secure Acceptance Override Billing Address.	
CsSAOverrideShippingAddress	CyberSource Secure Acceptance Override Shipping Address.	
CsCvnDeclineFlags	CyberSource Ignore CVN Result (CVN).	
CsTransactionType	Select Auth/Sale transaction	

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) {
```

1.3. To Setup Secure Acceptance Checkout API

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/ site_genesis_meta /meta/Cybersource_SecureAcceptance.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_SecureAcceptance** and set values for the parameter:

Field	*Description*	*Value to Set*
CsSAType	Secure Acceptance Type	SA_SILENTPOST
SA_Silent_ProfileID	Secure Acceptance Silent Post Profile ID. Follow ****"Creating Checkout API Profile"**** section from this link.	

SA_Silent_SecretKey	Secure Acceptance Silent Post Secret Key. Follow this link.
SA_Silent_AccessKey	Secure Acceptance Silent Post Access Key. Get the access key from above step.
Secure_Acceptance_Token_Create_Endpoint	Secure_Acceptance_Token_Create_Endpoint.
Secure_Acceptance_Token_Update_Endpoint	Secure_Acceptance_Token_Update_Endpoint.
CsSAOverrideBillingAddress	CyberSource Secure Acceptance Override Billing Address.
CsSAOverrideShippingAddress	CyberSource Secure Acceptance Override Shipping Address.
CsCvnDeclineFlags	CyberSource Ignore CVN Result (CVN).
CsTransactionType	Select Auth/Sale transaction

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.cont
inueplaceorder','locale',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();

}
```

1.4. To Setup Secure Acceptance Flex MicroForm

Prerequisites: You will also need to create an API Key and API Shared Secret Key that you can use to authenticate requests to our sandbox. Follow same steps to generate Production key and shared secret.

Step 1: In Business Manager, go to Administration > Customization > Services and click on the 'cybersourceflextoken' Credentials. Ensure the appropriate URL is set for the environment you

are configuring. - Test: https://apitest.cybersource.com/flex/v1/keys - Production: https://api.cybersource.com/flex/v1/keys

Step 2: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/ site_genesis_meta /meta/Cybersource_SecureAcceptance.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 3: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource-FlexMicroform** and set values for the parameter:

Field	***Description***	***Value to Set***
CsSAType	Secure Acceptance Type	SA_FLEX
Flex_Microform_HostName	Test: testflex.cybersource.com Production: flex.cybersource.com	
Flex_Microform_KeyID	Flex Microform Key ID. Follow link to generate keys.	
Flex_Microform_SharedSecret	Flex Microform Shared Secret	
CsTransactionType	Select Auth/Sale transaction	

Step 4: Navigate to '**Administration > Global Preferences > Locales**' and ensure the local 'en_US' is present. If not present, create a new local with the following information:

- Language Code: en
- Country Code: US
- XML Code: en-US
- ISO-3 Language: eng
- ISO-3 Country: USA
- Fallback Locale: English(en)

Once present, select the 'en_US' local, and navigate to the 'Regional Settings' tab. In the 'Long Date Pattern' field, enter the sting: EEE, dd MMM yyyy HH:mm:ss z

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(Pay
mentInstrument.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()
```

1.5. To Setup Direct Cybersource SOAP API

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or
import **"metadata/site_genesis_meta/meta/Cybersource_SecureAcceptance.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_SecureAcceptance** and set values for the parameter:

Field	*Description*	*Value to Set*
CsSAType	Secure Acceptance Type	**None**
CsTransactionType	Select Auth/Sale transaction	

Payer Authentication (3D Secure)

Prerequisite

- Please contact your Cybersource Representative to sign up and receive your Payer Authentication credentials.

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/site_genesis_meta/meta/PayerAuthentication.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_PayerAuthentication** and set values for the following parameters:

Field	*Description*
CS PA Merchant ID	Payer Auth merchant ID
CS PA Save Proof.xml	To enable/disable saving of proof.xml in order object
CS PA Save ParesStatus	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
CruiseApiKey	A shared secret value between the merchant and Cardinal. This value should never be exposed to the public.
CruiseApiIdentifier	GUID used to identify the specific API Key
CruiseMerchantName	Merchant Name
CruiseOrgUnitId	GUID to identify the merchant organization within Cardinal systems

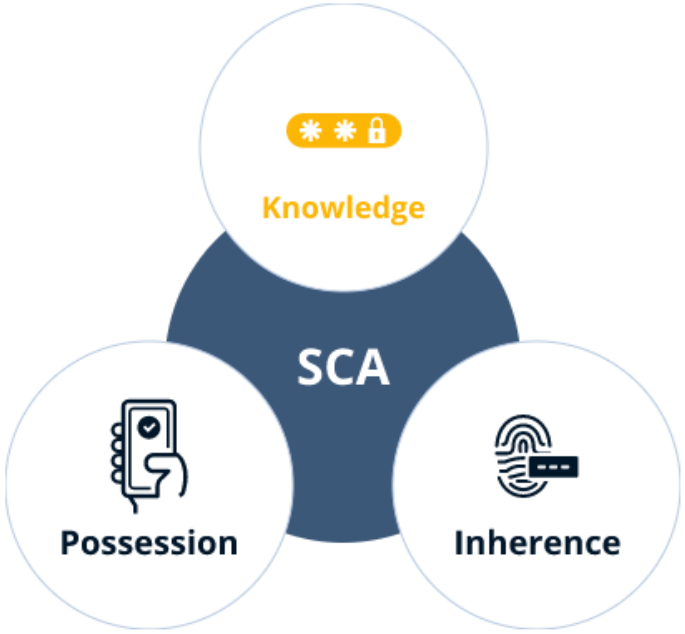
CardinalCruiseApiPath	Songbird.js script API path. Refer the section *CDN* in this link to get API path.

Step 3: Go to **Merchant Tools > Ordering > Payment Methods**, select 'Credit/Debit cards' and check the payer authentication checkbox on any credit card types you want to support Payer Authentication.

SCA (Strong Customer Authentication) :

The new rules, called Strong Customer Authentication (SCA), are intended to enhance the security of payments and limit fraud.

Strong customer authentication (SCA) is defined as “an authentication based on the use of two or more elements categorised as **knowledge** (something only the user knows), **possession** (something only the user possesses) and **inherence** (something the user is). These must be independent from one another, in that the breach of one does not compromise the reliability of the others, and is designed in such a way as to protect the confidentiality of the authentication data.”



If response code 478 is received, the issuer is declining the authorisation request but advising that if the card holder gets SCA’d, the issuer will approve the authorisation.

Proposed Approach

Automatically retry the transaction adding Payer Auth Mandate Challenge (code 04) to force cardholder SCA challenge. This flow should appear seamless to the card holder

Site Preferences:

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/sfra_meta/meta/PayerAuthentication.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_PayerAuthentication** and set values for the following parameters:

Field	*Description*
IsSCAEnabled	Enable Strong Customer Authentication

Set the value for IsSCAEnabled to yes to use Strong Customer Authentiaction feature.

Upgrade to 3DS2.0

If you are currently using CYBS cartridge and would like to upgrade to 3DS2.0, please refer doc **CyberSource Storefront Reference Architecture LINK Cartridge 3DS2.docx** under cartridge documentation folder.

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}"/>
```

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

---

2. Apple Pay

Step 1: Create a merchant identifier in Apple Portal:

A merchant identifier uniquely identifies you to Apple Pay as a merchant who is able to accept payments. You can use the same merchant identifier for multiple native and web apps. It never expires.

1. Go to Apple portal: https://help.apple.com/developer-account/#/devb2e62b839?sub=dev103e030bb
2. In Certificates, Identifiers & Profiles, select Identifiers from the sidebar, then click the Add button (+) in the upper-left corner.
3. Select Merchant IDs, then click Continue.
4. Enter the merchant description and identifier name, then click Continue.
5. Review the settings, then click Register.

Step 2: Enrolling in Apple Pay in Cybersource

To enroll in Apple Pay:

1. Log in to the Business Center:
   - Test: link
   - Live: link
2. On the left navigation pane, click the **Payment Configuration** icon.
3. Click **Digital Payment Solutions**. The Digital Payments page appears.
4. Click **Configure**. The Apple Pay Registration panel opens.
5. Enter your Apple Merchant ID. (Created in Step 1.4)
6. Click **Generate New CSR**.
7. To download your CSR, click the **Download** icon next to the key.
8. Follow your browser's instructions to save and open the file.

Step 3: Complete the enrollment process by submitting your CSR to Apple

Create a payment processing certificate: A payment processing certificate is associated with your merchant identifier and used to encrypt payment information. The payment processing certificate expires every 25 months. If the certificate is revoked, you can recreate it.

1. In Certificates, Identifiers & Profiles, select Identifiers from the sidebar.
2. Under Identifiers, select Merchant IDs using the filter in the top-right.
3. On the right, select your merchant identifier. Note: If a banner appears at the top of the page saying that you need to accept an agreement, click the Review Agreement button, and follow the instructions before continuing.
4. Under Apple Pay Payment Processing Certificate, click Create Certificate.
5. Click Choose File.
6. In the dialog that appears, select the CSR file downloaded from Step 2.7, then click Choose.
7. Click Continue.

Step 4: Configure Apple Pay in SFCC Business Manager

Business Manager Configuration

1. Go to: “Merchant Tools > Site Preferences > Apple pay
2. Check “Apple Pay Enabled?”
3. Fill in the “Onboarding” form:
    - Ensure “Apple Merchant ID” and “Apple Merchant Name” match settings in your Apple account
    - Ensure all other fields match your supported Cybersource settings
4. Fill in the “Storefront Injection” form:
    - Selects where Apple Pay buttons should be displayed on your site.
5. Fill in “Payment Integration” form:
    - Leave all form fields blank
    - Ensure “Use Basic Authorization” is checked
6. Click "Submit"

Step 5: Domain Registration in SFCC Business Manager

1. Go to: **“Merchant Tools > Site Preferences > Apple Pay**
2. Under **Domain Registration** section
    a. Click on **Register Apple Sandbox** under Apple Sandbox section for registering SFCC to Apple Sandbox account.
    b. Click on **Register Apple Production** under Apple Production section for registering SFCC to Apple Production account.

Step 6: Payment Processor

1. In the Business Manager, go to **Merchant Tools > Ordering > Payment Methods** and select **DW_APPLE_PAY**. And in **DW_APPLE_PAY details**, double check if **Payment Processor** = **"CYBERSOURCE_CREDIT"**

Step 7: Site Preferences:

Step 1: Upload Cybersource metadata in Business Manager. If not follow “Step 2: Upload metadata” or import **"metadata/site_genesis_meta/meta/ApplePay.xml"** in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Apple Pay** and set values for the following parameters:

Field	***Description***
ApplePayTransactionType	Select Sale/Auth transaction type

3. PayPal

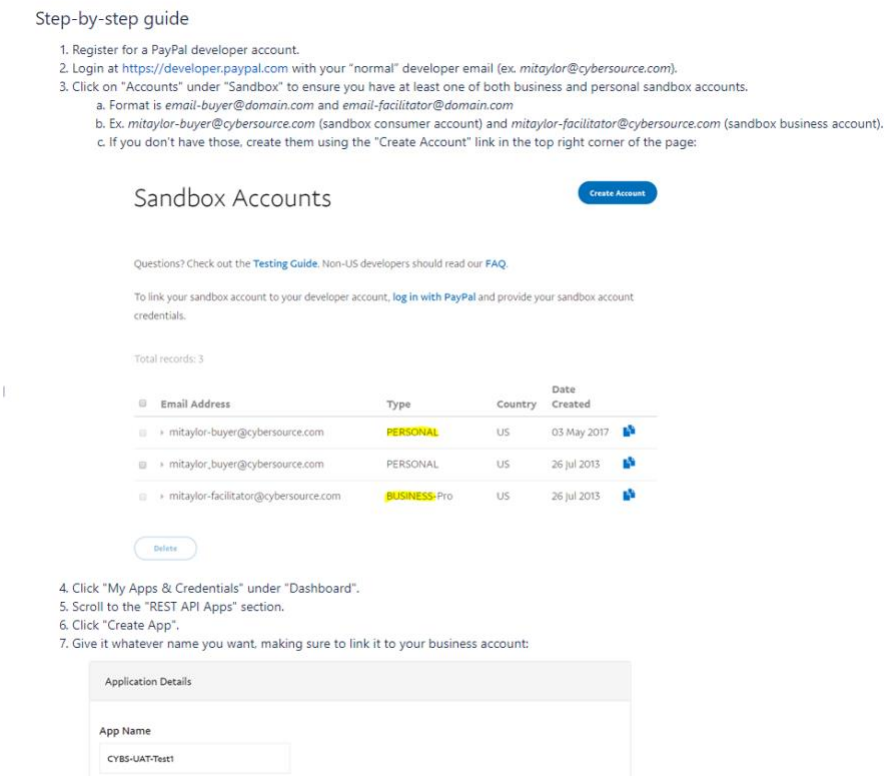
Implementation

Cybersource Cartridge supports the following for PayPal: a. PayPal Express b. PayPal Credit c. PayPal Billing Agreement

Prerequisite

Prior to development phase, there are a generic set of configurations that a development team needs to account for. These configurations include:

1. PayPal developer account
2. PayPal sandbox account Screenshot of the detailed set of configurations for #1 & #2.



3. Linking developer and sandbox account. On creating a PayPal developer account, get in touch with the CyberSource team, share the developer account details and get the developers' details configured on CyberSource (BackOffice Configuration tool). Share the following keys with Cybersource:

- ClientID (**paypalgateway_client_key**) - Follow this link to generate.
- Secret (**paypalgateway_secret_phrase**) - Follow this link to generate.
- Merchant Account ID (**paypalgateway_mid) –** Follow this link to generate.
- Merchant email (**paypalgateway_merchant_email) -** Follow this link to generate.

a. To setup PayPal Express

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_Paypal.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_Paypal** and set values for the parameter:

Field	***Description***
CsEnableExpressPaypal	Effectively enables or disables the PayPal Express checkout.
Paypal Order Type	The type of authorization to follow for PayPal orders. Select STANDARD for Authorize & Capture or select CUSTOM for just Authorize

b. To setup PayPal Credit

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_Paypal.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_Paypal** and set values for the parameter:

Field	***Description***
Paypal Order Type	The type of authorization to follow for PayPal orders. Select STANDARD for Authorize & Capture or select CUSTOM for just Authorize

b. To setup PayPal Billing Agreement

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_Paypal.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_Paypal** and set values for the parameter:

Field	*Description*
Billing Agreement	Effectively enables or disables the PayPal Billing Agreement.
Paypal Order Type	The type of authorization to follow for PayPal orders. Select STANDARD for Authorize & Capture or select CUSTOM for just Authorize

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','ch
eckout',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="${dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.METHOD_PAYPAL).is
Active() &&  dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}">
                            <isif
condition="${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.dynamic
HtmlName}"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="${dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.METHOD_PAYPAL).is
Active() &&  dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}">
                                <isif
condition="${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="${dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.METHOD_PAYPAL).is
Active() &&  dw.system.Site.current.getCustomPreferenceValue('CsEnableExpressPaypal')==true}">
                                <isif
condition="${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

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.

1. Device or App have code written for checkout journey where user opted for Android Pay
2. Android 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 “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]

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** (JSONParams). JSONParam will contains dw.order.LineItemCtnr, orderNo , IPAddress, encryptedPaymentData

Parameter	Type
lineItemCtnr	dw.order.LineItemCtnr
orderNo	String
IPAddress	String
encryptedPaymentDat a	String

The function “**MobilePaymentAuthRequest**” is called when network token is available **MobilePaymentFacade. MobilePaymentAuthRequest** (**MobilePaymentAuthRequest** (JSONParams).

d. 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

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 "**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

e. 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
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
ServiceResponseObject	Object

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

---

4. Visa Checkout

Prerequisite:

1. To complete the checkout process successfully create Visa Checkout profile (link) on CyberSource business center console under 'Account Management > Digital Payment Solutions > Profiles > Enable Visa Checkout'.
2. Click profile tab and add profile, configure all the mandatory settings, also use API Key from Setting Tab.
3. Create Secret key from **'Account Management > Client Integration Management > Payment Configuration > Key Management'**. Click Generate key and select shared secret.

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_VisaCheckout.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_VisaCheckout** and set values for the parameter:

FIeld	***Description***
cybVisaSdkJsLibrary	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
cybVisaTellMeMoreLinkActive	Indicate whether Tell Me More Link to be displayed with VISA button true (default) false
cybVisaButtonColor	The color of the Visa Checkout button. standard or neutral.
cybVisaButtonSize	The size of the Visa Checkout button
cybVisaButtonHeight	The height of the Visa Checkout button in pixels.
cybVisaButtonImgUrl	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
cybVisaCardBrands	Brands associated with card art to be displayed
cybVisaButtonWidth	The width of the Visa Checkout button in pixels.
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
cybVisaExternalProfileId	Profile created externally by a merchant whom Visa Checkout uses to populate settings
cybVisaSecretKey	The secret key specified VISA Checkout account profile
cybVisaAPIKey	The Visa Checkout account API key specified in cyberSource business center
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
cybVisaButtonOnCart	CyberSource Visa Button display on minicart and cart
cybVisaTransactionType	Select Sale/Auth transaction type

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

2.) 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.directcheck
out','checkout',null)} &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');
    }
}
```

5. Bank Transfer

Bank Transfer supports 5 types of Payment methods –

- SOFORT
- BANCONTACT
- IDEAL

Bank Transfer Service Support by Country

Payment Method	Country	Services
Bancontact	Belgium	Sale Check Status Refund
iDEAL	Netherlands	Options Sale Check Status Refund
Sofort	Austria Belgium Germany Italy Netherlands Spain	Sale Check Status Refund

Bank transfer supports 4 different services:

- **Option Service:** This service is valid only for iDEAL transactions. The options service (apOptionsService) retrieves a list of bank option IDs and bank names which you can display to the customer on your web site
- **Sale Service:** The sale service (apSaleService) returns the redirect URL for customer's bank. The customer is directed to the URL to confirm their payment details.
- **Check Status Service:** The check status service returns the latest status of a transaction. It is a follow-on request that uses the request ID value returned from the sale service request. The request ID value links the check status request to the payment transaction
- **Refund Service:** The refund service request (apRefundService) is a follow-on request that uses the request ID value returned from the sale request. The request ID value links the refund transaction to the original payment transaction

Bank Transfer functionality is specific to PMs with sale and check status service.
SG makes the call to CyberSource Sale service to authorize the purchase amount. A secondary call is made to check a Status service to determine result of the authorization, and the subsequent Order creation or failure.

Implementation

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_BankTransfer.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_BankTransfer** and set values for the parameter:

Field	*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

Step 3: Go to **Merchant Tools > Ordering > Payments Methods > Bank_Transfer** and set values for the parameter:

FIeld	*Description*
isBicEnabled	Attribute to check if BIC field is required for EPS and GIROPAY to display on billing page
isSupportedBankListRequired	Attribute to check if bank list is required for IDEAL to display on billing page

To setup Decision Manager for Bank Transfer

Refer to this link if you want to setup Decision Manager feature for Bank Trasfer transactions.

Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource: Decision Manager Configuration** and set values for the parameter:

Field	*Description*
Decision Manager Enable for Bank Transfer	Enable or disable Decision Manager for Bank Transfer transactions.

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
```

6. Alipay

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_Alipay.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_Alipay** and set values for the parameter:

Field	***Description***
apPaymentType	Alipay Payment Type for Domestic as well as International Payment
apTestReconciliationID	Test Reconciliation ID for Alipay

Step 3: Under **'Merchant Tools > Ordering > Payment Methods**' Make sure the 'ALIPAY' payment method is enabled and configured to use the 'CYBERSOURCE_ALIPAY' payment processor.

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 {};
    }
```

7. 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.

Step 1: 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.

Step 2: On the Payment Methods page, Select the locale (language) you want to set up, then select the Klarna payment method.

Step 3: Enter the merchantID and merchantKey field in CyberSource Credentials section of payment method. If you leave these empty, the service will fall back to the values you entered in the CS core site preferences.

Step 4: Select the appropriate bill-to language setting under the 'Klarna' custom attribute group.

Step 5: Upload Cybersource metadata in Business Manager. If not follow "Step 6: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_Klarna.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_Klarna** and set values for the parameter:

Field	***Description***
Klarna Decision Manager Required	Enable or Disable Decision Manager
Klarna JS API Path	Klarna JS API Library Path

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(Cybersource
Constants.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="${'klarnaJSAPIPath' in dw.system.Site.current.preferences.custom &&
!empty(dw.system.Site.current.preferences.custom.klarnaJSAPIPath)
                    &&
dw.order.PaymentMgr.getPaymentMethod(CybersourceConstants.KLARNA_PAYMENT_METHOD).is
Active()}">
    <script src="${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()
```

8. WeChat Pay

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource_WeChat.xml**" in Business Manager (**Administration > Site Development > Import & Export**).

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_WeChat** and set values for the parameter:

FIeld	***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
NumofCheckStatusCalls	Max number of calls to check status for each AP sale

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').getCybers
ourceHelper();

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(Cybersource
Constants.KLARNA_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(CybersourceC
onstants.WECHAT_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"
         }
```

6. Configure Features (OPTIONAL)

1. Tax Calculation

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "metadata/site_genesis_meta/meta**/Cybersource.xml**" in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_TaxConfiguration** and set values for the parameter:

Field	***Description***
CS Tax Calculation Enabled	Enable or disable CyberSource tax service
CS Tax Calculation Nexus States List	List of states to charge tax in
CS Tax Calculation No Nexus States List	List of States to not charge tax in
CS Tax Calculation Default Product Tax Code	Default tax code used when tax code is not set on a product
CS Tax Calculation Purchase Order Acceptance City	Purchase order acceptance state code
CS Tax Calculation Purchase Order Acceptance Zip Code	Purchase order acceptance zip code
CS Tax Calculation Purchase Order Acceptance Country Code	Purchase order acceptance country code
CS Tax Calculation Purchase Order Origin City	Purchase order origin city
CS Tax Calculation Purchase Order Origin StateCode	Purchase order origin state code
CS Tax Calculation Purchase Order Origin ZipCode	Purchase order origin zip code

CS Tax Calculation Purchase Order Origin Country Code	Purchase order origin country code
CS Tax Calculation ShipFrom City	Ship from city
CS Tax Calculation ShipFrom StateCode	Ship from state code
CS Tax Calculation ShipFrom ZipCode	Ship from zip code
CS Tax Calculation ShipFrom Country Code	Ship from country code

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').CalculateTa
xes(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 {
```

2. Delivery Address Verification

Step 1: To enable this service, Go to **Merchant Tools > Site Preferences > Custom Preferences > CyberSource: Core** and set the 'CS DAV Delivery Address Verification Enabled' preference to 'Yes'.

Step 2: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource.xml**" in Business Manager (**Administration > Site Development > Import & Export**)

Step 3: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_DeliveryAddressVerification** and set values for the parameter:

Field	***Description***
CS DAV Delivery Address Verification Enabled	Enable or disable the DAV service.
CS DAV Update Shipping Address With DAV Suggestion	Update the shipping address with the CS suggestion, if found.
CS DAV On Failure	Accept or Reject the order if DAV fails.

3. Address Verification Service (AVS)

Assuming you have implemented the Credit Card Authorization service, you are ready to use the AVS service.

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta /meta/Cybersource.xml**" in Business Manager (Administration > Site Development > Import & Export)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource_DeliveryAddressVerification** and set values for the parameter:

FIeld	***Description***
CS AVS Ignore AVS Result	Effectively enables or disables the AVS service
CS AVS Decline	Leave empty to follow CS default decline flag strategy Enter flags separated by

Flags	commas to overwrite the default flag rules

4. Device FingerPrint

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource.xml**" in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource** and set values for the parameter:

Field	***Description***
CS Device Fingerprint Enabled	Enable or Disable the Device Fingerprint Service
CS Device Fingerprint Organization ID	Device Fingerprint Organization ID
CS Device Fingerprint ThreatMetrix URL	URL pointing to JS that generates and retrieves the fingerprint
CS Device Fingerprint Time To Live	Time, in milliseconds between generating a new fingerprint for any given customer session

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.

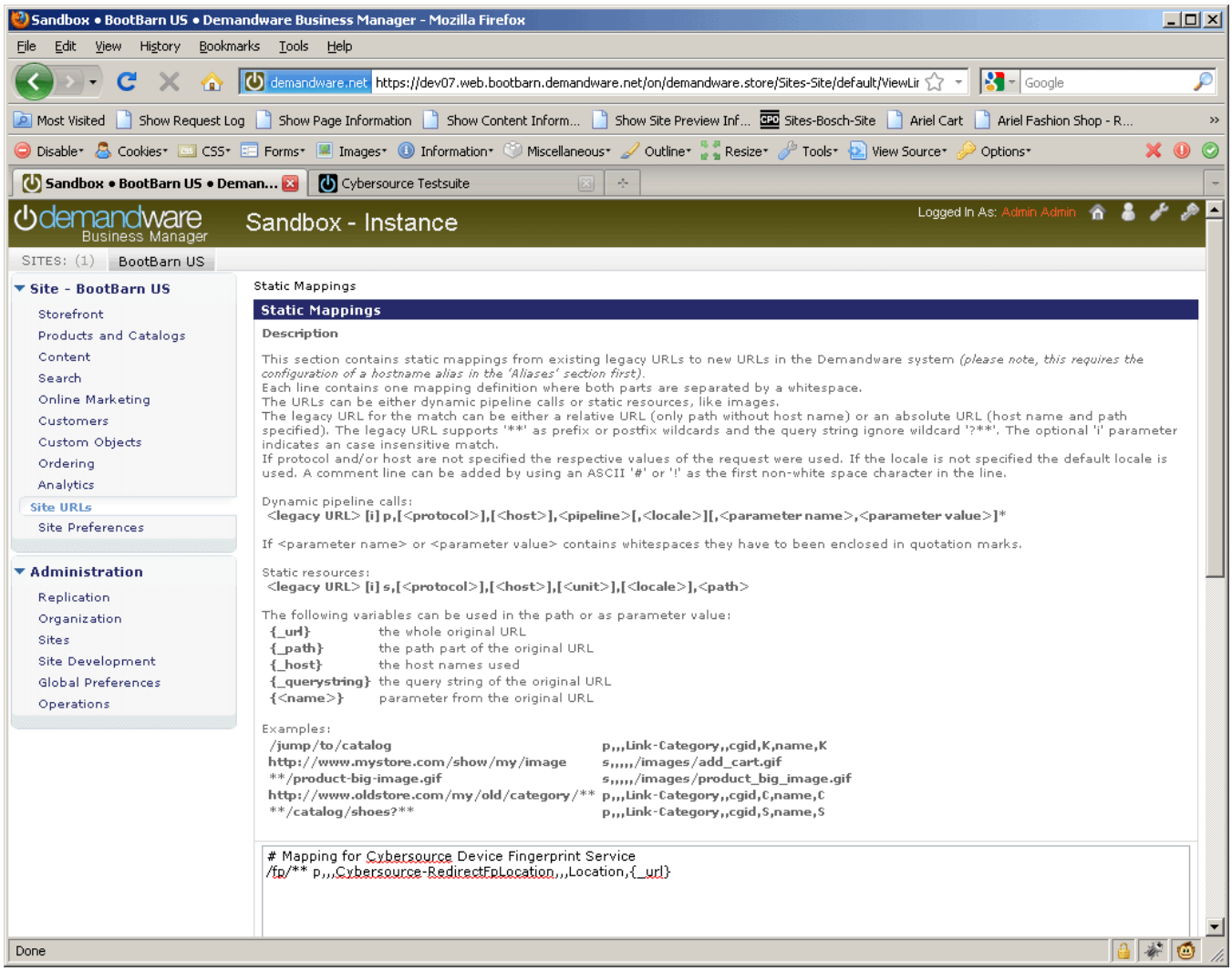
2. Include following snippet i.e. at **the billing.isml and summary.isml** page (Recommended: at bottom of page to have no visual impacts)

   [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 a95emandware 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:

Merchant Tools > Site URLs > Aliases

Aliases

This section is used to set hostname aliases for a site.

```
 *
 * Go to http://www.jsonlint.com/ to validate our file.
 *
 */

/*
 * __version must be set to "1"
 */

"__version" : "1",

/*
 * Settings section is used to configure main HTTP and HTTPS hostnames for a site.
 *
 */
"settings" : {
        "http-host" : "dev5.sitegen.com",
        "https-host" : "dev5.sitegen.com",
         "site-path": "SiteGenesis"
},

/*
 * Host name definitions.
 * The following section allows to define additional hostnames associated with the site.
 * With each hostname it is possible to define set of redirect rules.
 *
 */

/*
 * Examples
 */
```

5. Decision Manager

Refer to this link to learn about Cybersource's Decision Manager.

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource.xml**" in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource** and set values for the parameter:

Field	***Description***
CS Decision Manager Enabled	Enable or Disable Decision Manager

Step 3: SG version lower contain a hook that interfere with this service. While the hook manager has been updated in later versions of SG to prevent this, the CS cartridge is not yet compatible with those storefront versions. As suggested by SFCC, manual removal of the following hook from SG is required for this integration to function properly.

Remove

```
{
"name": "app.fraud.detection",
"script": "./cartridge/scripts/hooks/fraudDetection"
}
```

From app_storefront_base/hooks.json

Step 4: To enable **Decision Manager Order Update Job**: Decision Manager Order Update Job uses a REST API to retrieve order decisions from Cybersource and update the order confirmation status in SFCC.

To Integrate this job into your site, follow the below steps:

Step 4.1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import **"metadata/site_genesis_meta/jobs.xml"** in Business Manager (**Administration > Operations > Import & Export**)

Step 4.2: Navigate to '**Administration > Operations > Job**'. Select the Job '**CyberSource: Decision Manager Order Update**'.

Step 4.3: Select the "Job Steps" tab. Select the Sites you want the Job to run on, from the 'Scope' button.

Step 4.4: Select "UpdateOrderStatus" and update the following "custom parameters".

Field	***Description***

MerchantId	CS Merchant ID for the account to get Decisions from
SAFlexKeyID	Key ID. Work with CS to generate this value or Follow link to generate keys.
SAFlexSharedSecret	Shared secret. Work with CS to generate this value or Follow link to generate secret.

Step 4.5: Navigate to the 'Schedule and History' tab and configure the frequency you would like the job to run.

Step 4.6: Ensure the 'Enabled' check box is selected.

Step 4.7: Go to **Merchant Tools > Site Preferences > Custom Preferences** set values for the parameter:

Field	***Site Pref Group***	***Description***
CS Decision Manager OrderUpdate Lookback time	Cybersource: 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	Cybersource: Secure Acceptance	Host Name. CS can provide this value

Step 4.8: When moving to a production environment, the URL for the API call needs to be updated. This can be done in:
Administration > Operations > Services > Service Credentials > ConversionDetailReport – Details

6. Payment Tokenization

Refer to this link to learn about Cybersource's Token Management service.

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource.xml**" in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource** and set values for the parameter:

Field	*Description*
CS Tokenization Enabled	Enable or Disable the Tokenization Service
CS Subscription Tokens Enabled	Enable the request of a subscription token on credit card authorizations
LimitSavedCardRate	Enable Save Card Limit feature
SavedCardLimitFrame	Provide the number of cards that can be saved in a certain time period
SavedCardLimitTimeFrame	Provide the number of hours that saved card attempts are counted

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.ite
rator());

    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').getCybersourceHelper();

        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'));
    }
}
```

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.

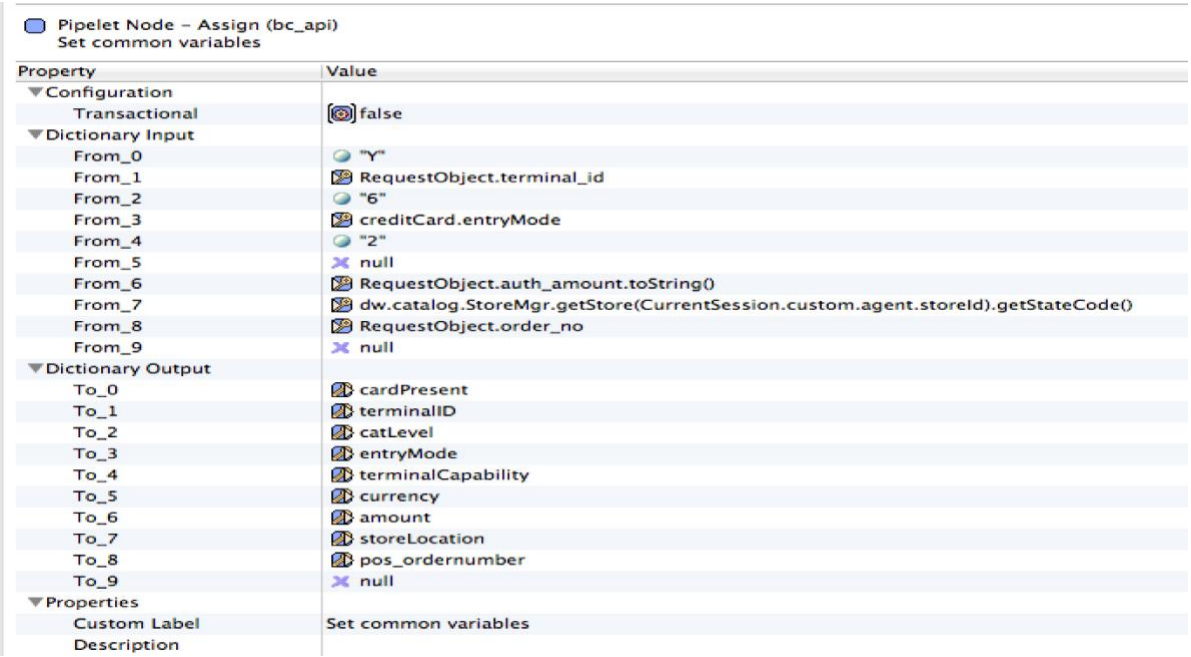
```
var expMonth : Number = 0;
var expYr : Number = 0;
var ccType : String = "";
var mode : String = "swiped";

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 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 (;)	Required if entryMode=swiped.

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

Batch Jobs

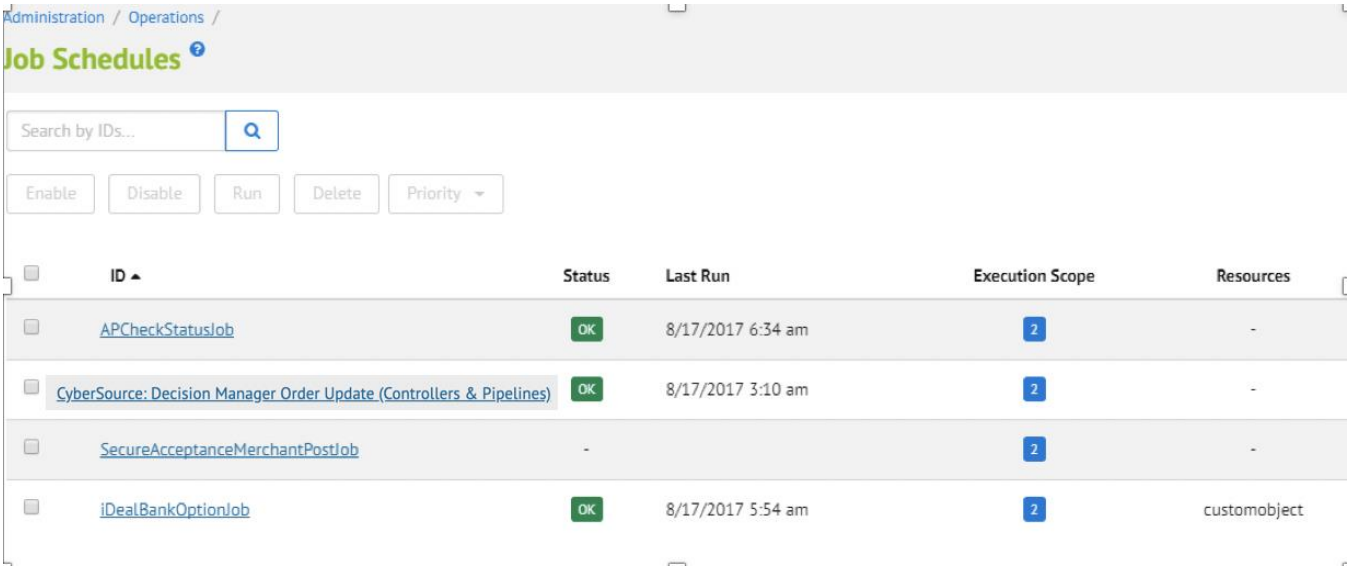
Cybersource cartridge has 4 batch jobs created for different functional items and are placed under int_cybersource cartridge:

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)

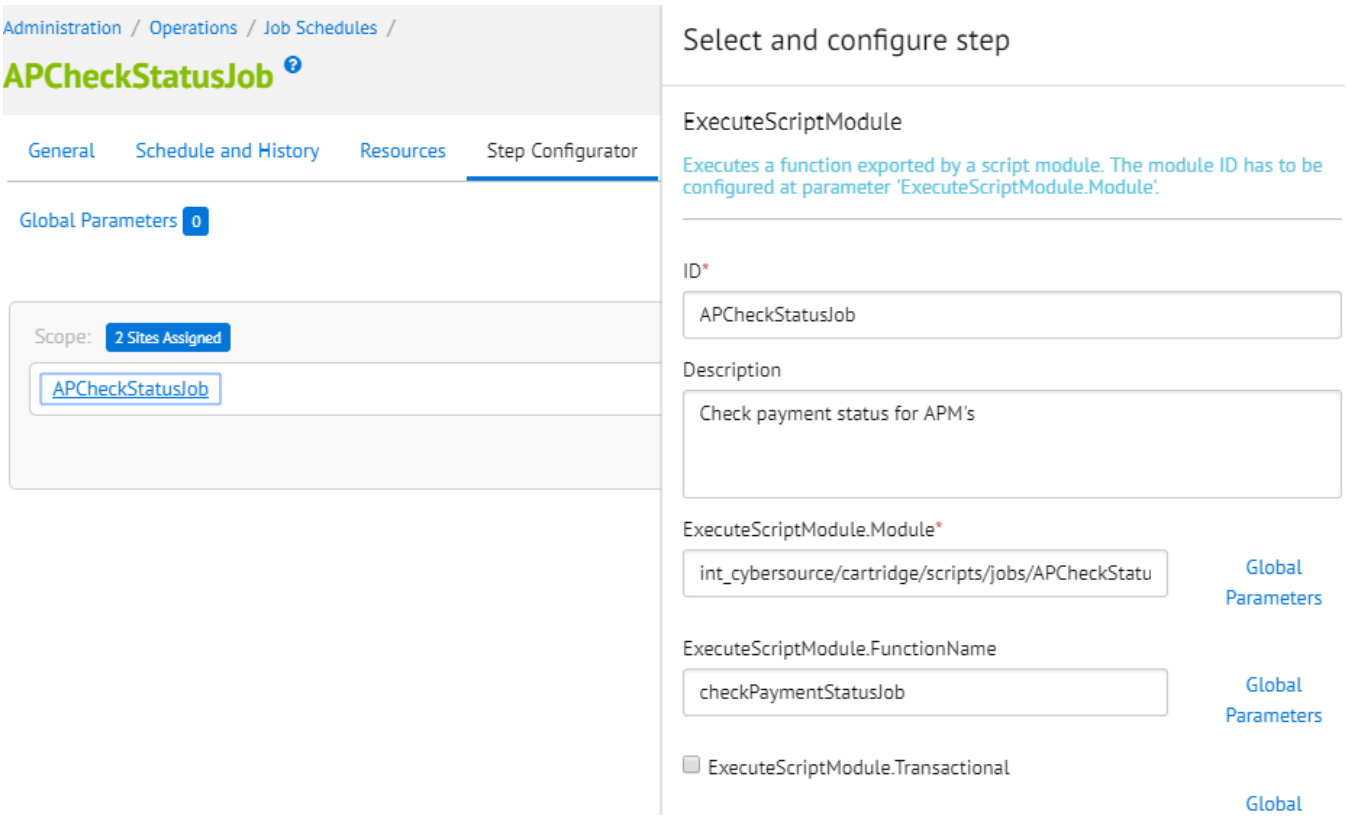
3. SecureAcceptanceMerchantPostJob
4. iDealBankOptionJob



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

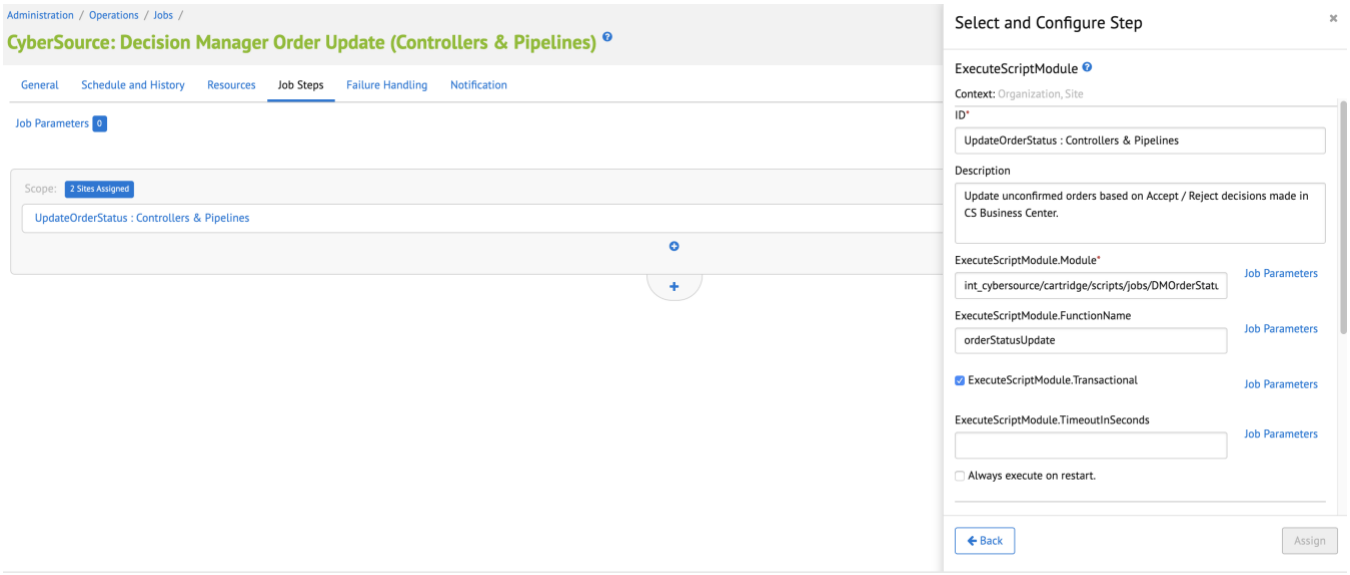


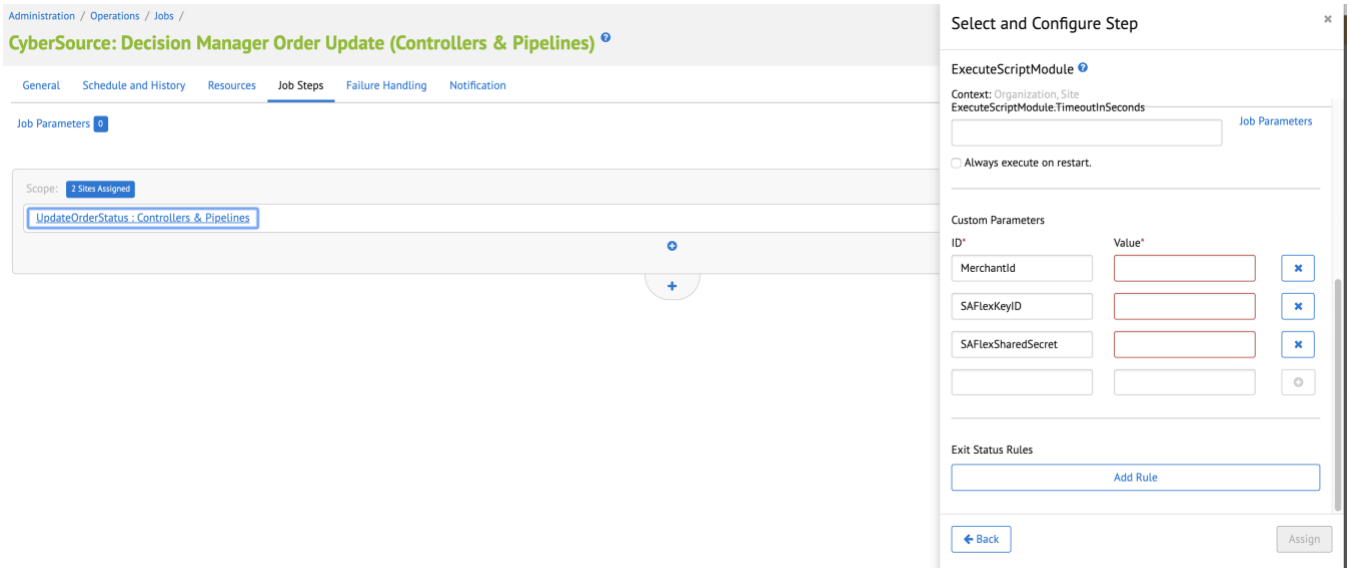
Batch Job for Conversion Detail Report

- 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



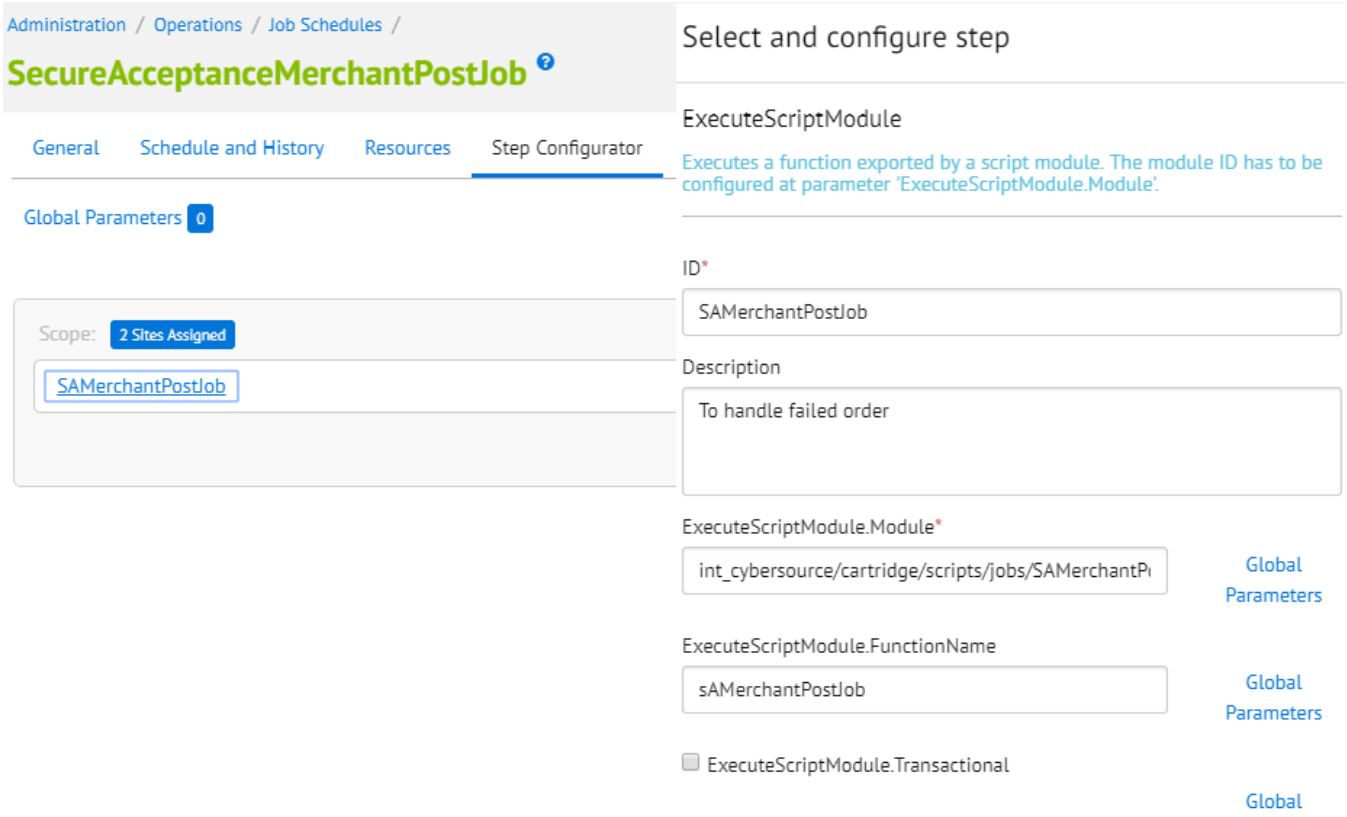


The batch job created for cybersource conversion detail report specified below, it updates the status of order in demandware which are in CREATED state and mark them as “CANCELLED” for rejected order or “NEW” for accepted order. The accepted orders are marked for “READY FOR EXPORT” as well.

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.

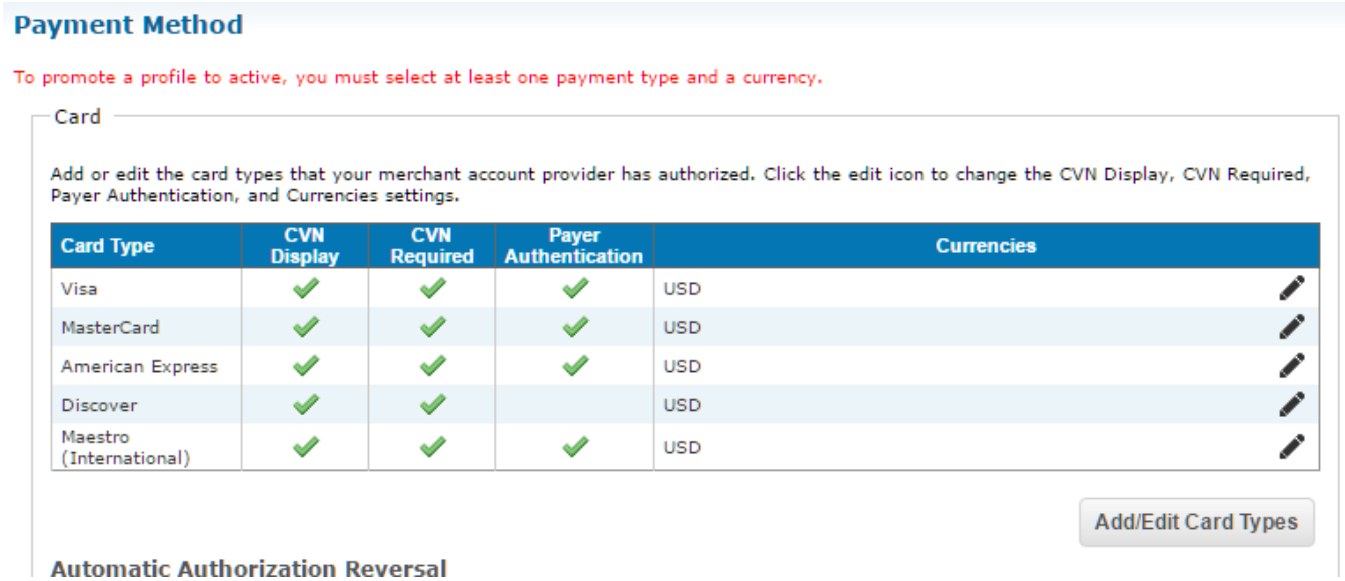


Secure Acceptance Profile Configuration into CyberSource Business Manager

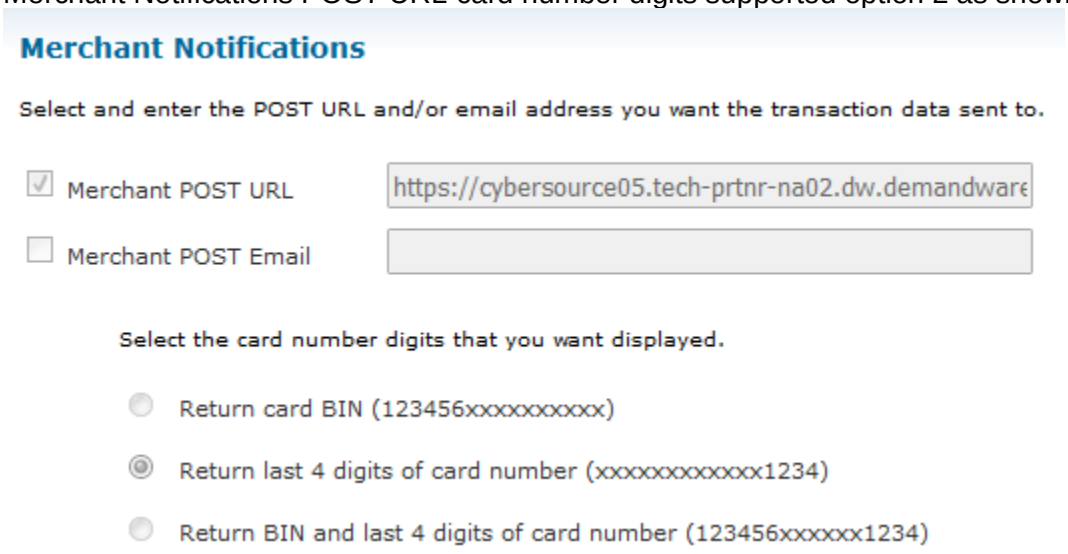
Secure Acceptance profile settings are configured on CyberSource business center console; along with other settings below are key settings which must be configured in cybersource profiles in order to complete the checkout process successfully.

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.

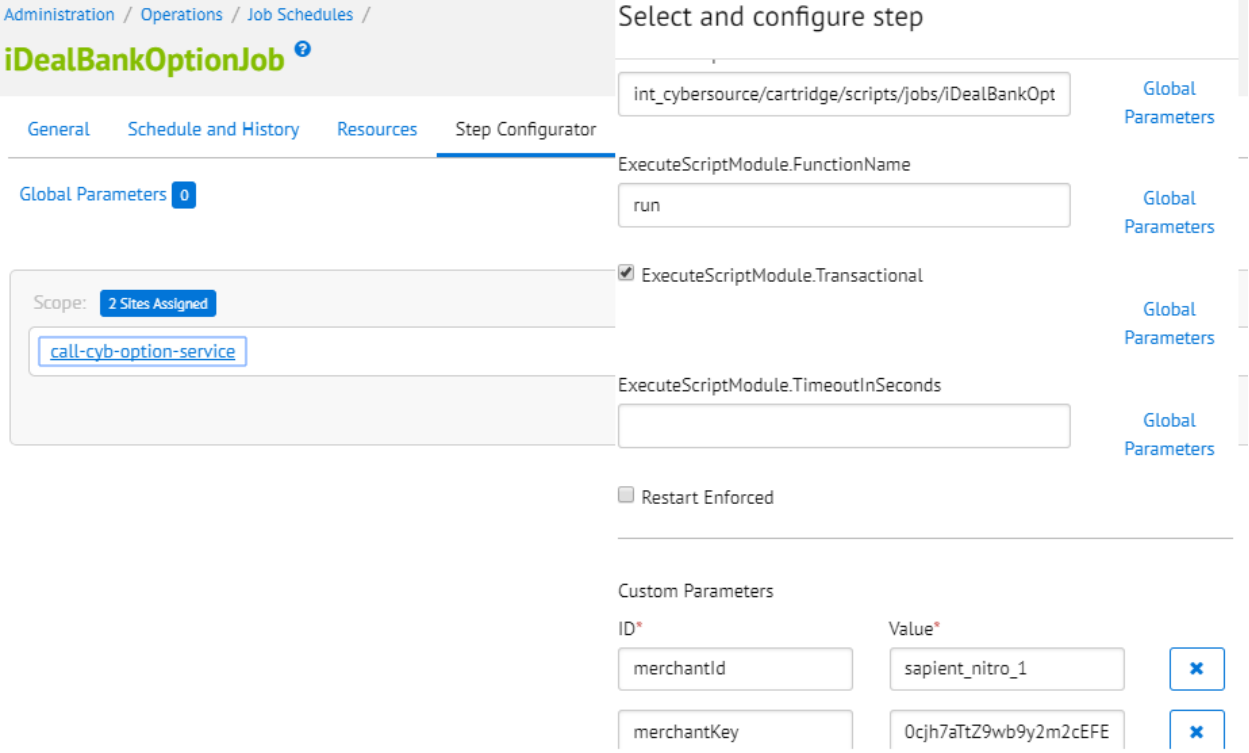


Batch Job for Bank Option List

- 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



7. Subscription Token Creation

Step 1: Upload Cybersource metadata in Business Manager. If not follow "Step 2: Upload metadata" or import "**metadata/site_genesis_meta/meta/Cybersource.xml**" in Business Manager (**Administration > Site Development > Import & Export**)

Step 2: Go to **Merchant Tools > Site Preferences > Custom Preferences > Cybersource** and set values for the parameter:

Field	***Description***
CS Subscription Tokens Enabled	Enable or Disable the option to generate subscription tokens

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

[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 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 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');
/**
*Verifiesacreditcardagainstavalidcardnumberandexpirationdateandpossiblyinvalidatesinvalid
formfields.
*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(Cybersourc
eConstants.METHOD_ApplePay)) {
      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(Cybersourc
eConstants.METHOD_AndroidPay)) {
      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_CREDITpro
cessor
*onlyandsettingtheordernoasthetransactionID.Customizationsmayuseotherprocessorsandcus
tom
*logictoauthorizecreditcardpayment.
*/
function Authorize(args) {
var orderNo = args.OrderNo;
var paymentInstrument = args.PaymentInstrument;
var paymentProcessor =
PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentPro
cessor();

   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
Install the REST client on machine or browser
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]
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

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.

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)

Test the Success response JSON

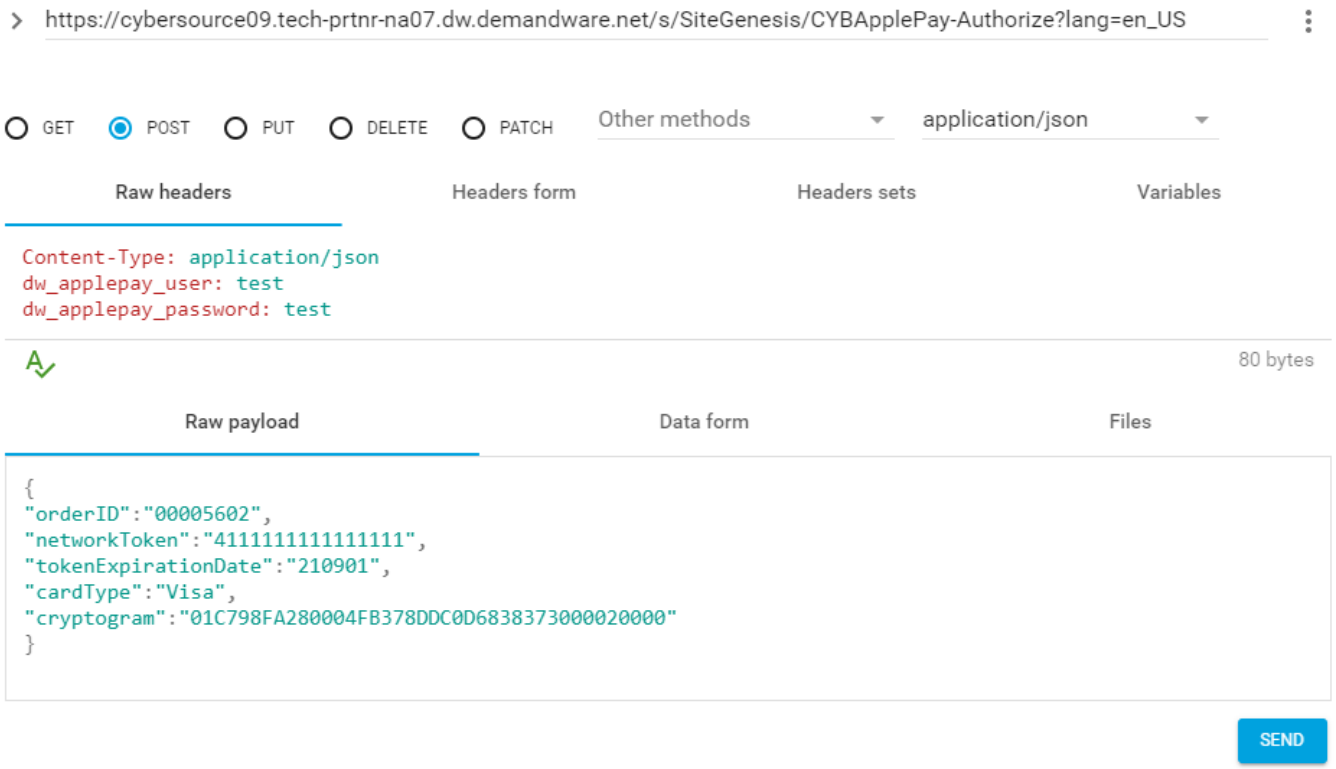
JSON KEY	JSON VALUE
TRANSACTION_RESULT	Below json key-value pairs
DECISION	Possible values ACCEPT | REVIEW | REJECT | ERROR | CANCEL
REASON_CODE	ReasonCode
REQUEST_ID	RequestID
REQUEST_TOKEN	RequestToken
AUTHORIZATION_AMOUNT	AuthorizationAmount
AUTHORIZATION_CODE	AuthorizationCode
AUTHORIZATION_REASON_CODE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

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

```
/**
*Verifiesacreditcardagainstavalidcardnumberandexpirationdateandpossiblyinvalidatesinvalid
formfields.
*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(Cybersourc
eConstants.METHOD_ApplePay)) {
       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(Cybersourc
eConstants.METHOD_AndroidPay)) {
       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_CREDITpro
cessor
*onlyandsettingtheordernoasthetransactionID.Customizationsmayuseotherprocessorsandcus
tom
*logictoauthorizecreditcardpayment.
*/
function Authorize(args) {
var orderNo = args.OrderNo;
var paymentInstrument = args.PaymentInstrument;
var paymentProcessor =
PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentPro
```

```
cessor();

   Transaction.wrap(function () {
      paymentInstrument.paymentTransaction.transactionID = orderNo;
      paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;
   });
     if
(session.forms.billing.paymentMethods.selectedPaymentMethodID.value.equals('DW_METH
OD_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
Install the REST client on machine or browser
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]
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

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.

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)

Test the Success response JSON

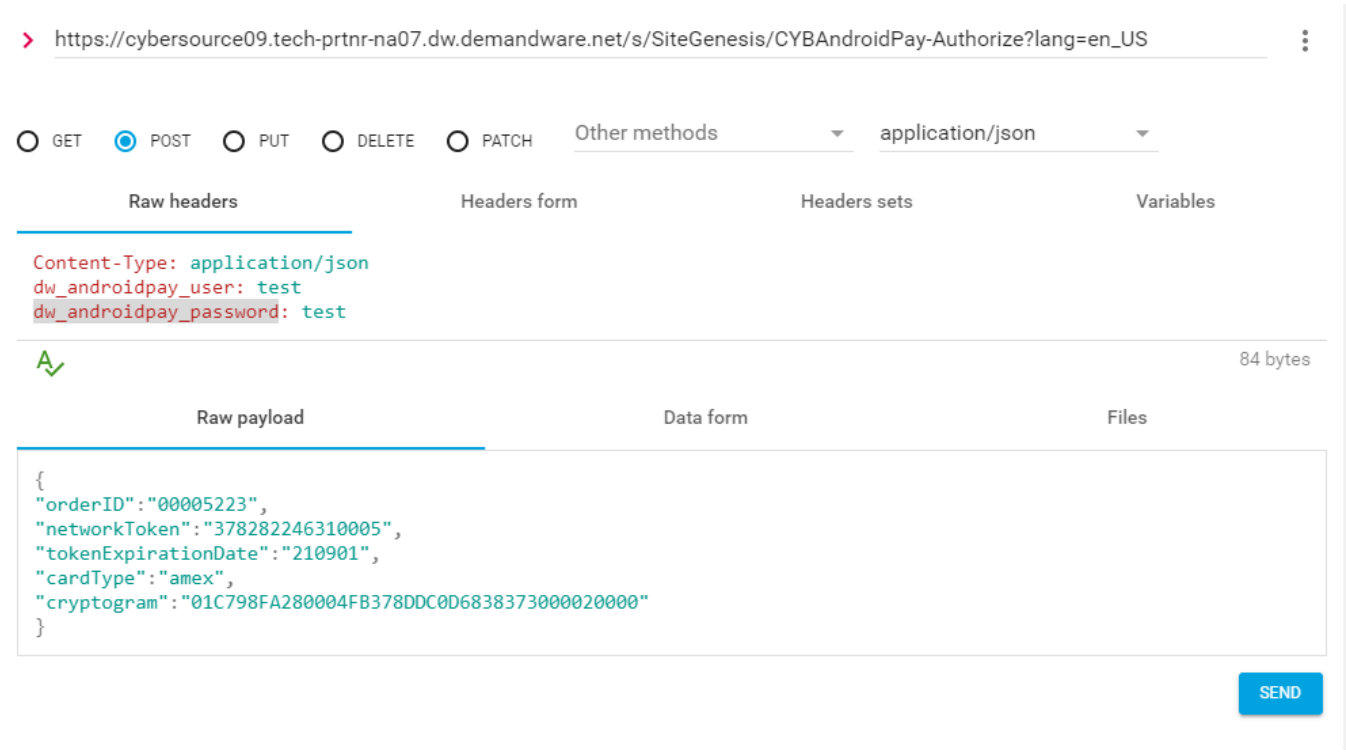
JSON KEY	JSON VALUE
TRANSACTION_RESULT	Below json key-value pairs
DECISION	Possible values ACCEPT | REVIEW | REJECT | ERROR | CANCEL
REASON_CODE	ReasonCode
REQUEST_ID	RequestID
REQUEST_TOKEN	RequestToken
AUTHORIZATION_AMOUNT	AuthorizationAmount
AUTHORIZATION_CODE	AuthorizationCode
AUTHORIZATION_REASON_CODE	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

Test the Validation/Failure response JSON

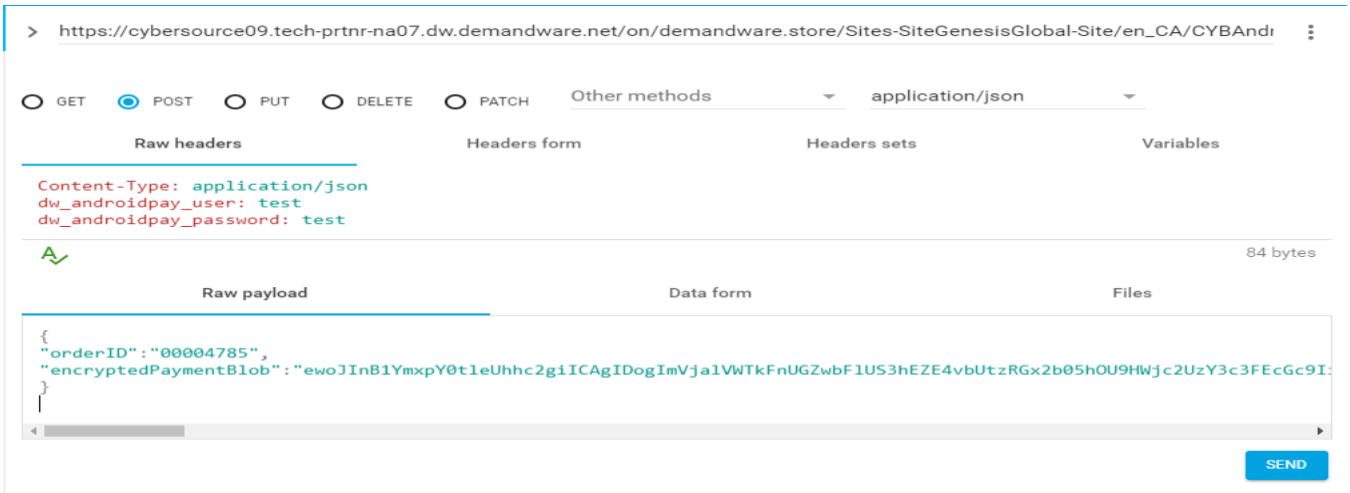
JSON KEY	JSON VALUE
ERROR_CODE	Validation failure error code of interface
ERROR_MSG	Validation failure message of interface

Sample AndroidPay InterfaceJSON Request /Response format

Interface 1: Request with network Token and Cryptogram data:



Interface2: Request with encrypted payment BLOB data.



12. Failover/Recovery Process

Visa has dedicated data centers in Virginia and Colorado. There are no single points of failure. Visa Data Centers implement redundant, dual-powered equipment, multiple data and power

feeds, and fault tolerance at all levels with 99.995% uptime. In case of any failover, please open support case @ https://support.cybersource.com.

13. Supported Locales

Out of box cartridge supports most of the locales like English (United States), English (United Kingdom), French (FRANCE), English (Austria), German (GERMANY), Dutch (NETHERLANDS) and more.

---

7. Test and Go Live

Test your integration, and configure your live account, so you can start processing live transactions.

Test your integration

Before you start accepting payments, test your integration in Test sandbox:

> The sandbox simulates the live payment gateway. The sandbox never processes an actual payment. We do not submit sandbox transactions to financial institutions for processing. The sandbox environment is completely separate from the live production environment, and it requires separate credentials. If you use your production credentials in the sandbox or vice versa, you get a 401 HTTP error.

Sign up for a sandbox account if you have not yet.

Use our test card numbers to make test payments: > The following test credit card numbers work only in the sandbox. If no expiration date is provided, use any expiration date after today's date. If the card verification code is required and is not provided, use any 3-digit combination for Visa, Mastercard, Discover, Diners Club and JCB; use a 4-digit combination for American Express.

Register SFCC sandbox to Apple Sandbox Account.

1. Go to: **"Merchant Tools > Site Preferences > Apple pay**
2. Under **Domain Registration** section
    a. Click on **Register Apple Sandbox** under Apple Sandbox section for registering SFCC to Apple Sandbox account.

To manage your evaluation account, log in to the Test Business Center and do the following: - View test transactions. - Access administrator users and access privileges. - Create roles with predefined access permissions. - View reports.

> **Important** Cybersource recommends that you submit all banking information and required integration services in advance of going live. Doing so will speed up your merchant account configuration.

Configure your live account

Once you have the credentials for the live environment:

1. Configure cartridge using your live account settings.
2. Test your integration.

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/

Upgrade Steps

Please follow the below steps to upgrade your cartridge version from 22.1.3 to 23.1.0.

1. Cartridge version 23.1.0 supports SFRA v6.3
2. Cybersource's Storefront Reference Architecture Cartridge can be installed from Salesforce Commerce Cloud's marketplace link.
3. Upload metadata to use Sale functionality

-> Cybersource cartridge installed from App Exchange comes with metadata to import

-> Go to **Cybersource/metadata/site_genesis_meta/sites/** folder.

-> Zip **site_genesis_meta** folder.

-> Go to **Administration->Site Development->Site Import & Export** and upload **site_genesis_meta.zip** file.

-> Import the uploaded zip file.

On successful import .

You will get the preference to select Transaction type (Sale/Auth) in all the payment method attribute groups.

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	7/10/2021	-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.
21.3.0	11/30/2021	- We have implemented Decision manager in Payer Authentication call. - We have implemented standalone Decision manager service so merchants can call this service on demand. - We have updated the cartridge to make it compatible with Salesforce B2C Commerce release 21.2. - We have disable Giropay and EPS Bank transfer method. - We have implemented Decision Manager service for Bank Transfer. - We have implemented Decision Manager service for Visa Click to Pay. - We have implemented Decision Manager service for PayPal. - We have implemented standalone service for Capture, Credit, Auth reversal for Klarna.
22.1.0	05/15/2022	- We have updated Flex token configuration in service meta file. - We have fixed the datetime format of paAuthenticationDate. - We have fixed a Decision Manager issue for Credit card transactions.
22.1.1	07/14/2022	- We fixed an issue in PayPal where non-english characters was not returned in the Cybersource response. - We added bin detection in 3ds flow. - We have added support for SCA changes for Irish processor.
22.1.2	09/05/2022	- We have updated the cartridge to make it compatible with Salesforce B2C Commerce release 22.7

22.1.3	02/10/2023	- Upgrade the cartridge to support SFRA v6.3 - Mapped requestID to transactionID in Klarna payment method
23.1.0	04/19/2023	- Implemented Sale for Credit Card, Google Pay, Visa Checkout and Apple Pay - We have extended the fix for other address fields in PayPal where non-english characters were not returned in the Cybersource response. - Fixed incorrect unit product price in line items in Business center.