ACI Integration Guide

**Document Revision History**

| **Revision** | **Author** | **Date** | **Status and Description** |
| --- | --- | --- | --- |
| 19.1.0 | Ganga Sukumaran | 08-05-2018 | Initial version |
| 21.1.0 | Aneesh Sudhakaran | 01-12-2020 | * Updated as part of recertification changes * 3DSecure changes |
| 21.2.0 | Sourabh Jacob | 24-06-2021 | * Replaced deprecated job step with custom job step |
| 24.0.0 | Chandana K M | 01-04-2024 | * Upgraded the ACI cartridge to latest SiteGenesis version (105.2.1). |

Contents

[1 Overview 5](#_Toc58259592)

[2 Features 5](#_Toc58259593)

[3 Compatibility 6](#_Toc58259594)

[4 Installation 6](#_Toc58259595)

[4.1 Cartridges 6](#_Toc58259596)

[4.2 BM Imports 6](#_Toc58259597)

[4.2.1 Cartridge path 6](#_Toc58259598)

[4.2.2 Metadata Import 6](#_Toc58259599)

[4.2.3 Services Import 6](#_Toc58259600)

[4.2.4 Payment Processor 7](#_Toc58259601)

[4.2.5 Jobs Import 7](#_Toc58259602)

[4.2.6 Content Library Import 7](#_Toc58259603)

[5 Configurations 8](#_Toc58259604)

[5.1 Site Preferences 8](#_Toc58259605)

[5.1.1 ACI 8](#_Toc58259606)

[5.1.2 ACI Merchant Configurations 9](#_Toc58259607)

[5.2 Configure Shipping Method Custom Attribute 9](#_Toc58259608)

[5.3 Configuring Payment Methods 10](#_Toc58259609)

[5.3.1 Configuring Credit Cards 10](#_Toc58259610)

[5.3.2 Configure PayPal 11](#_Toc58259611)

[5.3.3 Configuring additional payment methods 12](#_Toc58259612)

[5.3.4 Available payment methods 13](#_Toc58259613)

[6 Storefront Cartridge Changes 13](#_Toc58259614)

[6.1 Controller Changes 13](#_Toc58259615)

[6.1.1 COBilling.js 13](#_Toc58259616)

[6.1.2 COSummary.js 14](#_Toc58259617)

[6.1.3 COPlaceOrder.js 16](#_Toc58259618)

[6.1.4 CartModel.js 22](#_Toc58259619)

[6.2 ISML Changes 23](#_Toc58259620)

[6.2.1 billing.isml 23](#_Toc58259621)

[6.2.2 paymentmethods.isml 24](#_Toc58259622)

[7 Cartridge Extensions 26](#_Toc58259623)

[7.1 Sending additional parameters to ACI 26](#_Toc58259624)

[7.1.1 Custom Parameters collected from browser 26](#_Toc58259625)

[7.1.2 Custom Risk Parameters 26](#_Toc58259626)

[7.1.3 Additional custom parameters 26](#_Toc58259627)

[7.2 Backoffice operations 27](#_Toc58259628)

[7.2.1 Capture 27](#_Toc58259629)

[7.2.2 Refund 28](#_Toc58259630)

[7.2.3 Reversal 28](#_Toc58259631)

[7.3 Widget Customization 29](#_Toc58259632)

[7.3.1 Extending Java script functionality 29](#_Toc58259633)

[7.3.2 Modify widget styling 29](#_Toc58259634)

[8 Operations, Maintenance 30](#_Toc58259635)

[8.1 Availability 30](#_Toc58259636)

[8.2 Failover / Recovery Process 30](#_Toc58259637)

[8.3 Support 30](#_Toc58259638)

[9 Appendix 30](#_Toc58259639)

[9.1 System Extensions 30](#_Toc58259640)

[9.1.1 Basket 30](#_Toc58259641)

[9.1.2 Order 30](#_Toc58259642)

[9.1.3 PaymentMethod 31](#_Toc58259643)

[9.1.4 PaymentTransaction 31](#_Toc58259644)

[9.1.5 ShippingMethod 31](#_Toc58259645)

[9.1.6 CustomerPaymentInstrument 31](#_Toc58259646)

# Overview

The ACI payment cartridge enables a commerce cloud store to integrate with ACI payment service. This document provides detailed instructions on installing ACI payment cartridge onto a commerce cloud store.

The integration is based on the Site Genesis demo store, version 105.2.1, provided by SFCC.

The integration consists of an archive with contents as described in the below table.

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| cartridges | Contains ACI integration cartridges |
| cartridges\int\_aci\_controllers | This cartridge contains the controller version changes for ACI integration |
| cartridges\int\_aci\_core | This cartridge contains the API calls for ACI integration |
| site\_template.zip | Contains custom attributes and settings required for the integration |
| Documentation | Contains this document “ACI Integration Guide” |

# Features

The integration described in this document supports the following features:

* Hosted payment page checkout for payment methods offered by ACI using the ACI CopyAndPay widget
* Ability to manage payment methods offered by ACI via BM
* Supports both synchronous and asynchronous payment methods
* Ability to capture payment immediately during order placement or to authorise payment first and capture payment later
* Code plugins for refund / capture / reversal operations
* Ability to save cards to account and use them later during checkout
* Allows business users to modify the ACI payment widget styling
* Allows business users to modify the JavaScript functionality that is provided with the cartridge
* Ability to specify custom risk parameters to be sent to ACI
* Ability to hold orders which needs manual review in pending status and release or cancel them later based on review

# Compatibility

Available since Commerce Cloud Platform Release 24.4, Site Genesis 105.2.1. The cartridge was tested against compatibility mode 22.7

The cartridge is available for installations on storefronts that support both SFRA and Site Genesis implementations.

# Installation

## Cartridges

Cartridges relevant for controller version are int\_aci\_controllers and int\_aci\_core

## BM Imports

### Cartridge path

Add the cartridges shown below to the beginning of the cartridge path

|  |  |
| --- | --- |
| Site Cartridge Path | int\_aci\_controllers:int\_aci\_core |
| BM Cartridge Path | int\_aci\_core |

### Metadata Import

Locate the folder site-template in the installation package and perform following steps.

* Review the contents in the site-template folder.
* Change the site ID under path ***site-template\sites*** to your site ID.
* Change the library ID under path ***site-template\libraries*** to your library ID
* Archive the folder to ‘site-template.zip’ and import the file via Site Import & Export

### Services Import

A new service, “http.aciworldwide.checkout”, has been created as part of the integration.

Import File: ***site\_template\services.xml***

Please set the urls as shown below based on the instance where the cartridge is being integrated

|  |  |
| --- | --- |
| **Environment** | **URL** |
| Non-production instances | <https://test.oppwa.com/v1> |
| Production instances | <https://oppwa.com/> |

### Payment Processor

A new payment processor named ACI has been created as part of the cartridge.

### Jobs Import

Following jobs have been developed as part of cartridge.

|  |  |
| --- | --- |
| **Job Name** | **Job Description** |
| ACI Pending Transactions | The job will periodically check the status of all pending transactions in ACI and update the order export status or cancel the order depending on the transaction status in ACI.  *NOTE: Please change the site-id to the correct site id before importing. As this is scheduled job, please check and update the frequency as per merchant requirements.* |

Import File: ***site\_template\jobs.xml***

### Content Library Import

*NOTE: Please change the library id in the file to the correct library id before importing.*

# Configurations

## Site Preferences

Two site preference groups have been created for this cartridge

### ACI

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Enabled/Disabled | Flag to enable or disable ACI payment method |
| ACI API Version | Version of ACI API. Defaulted to v1. This value needs to be changed when there is a version change in ACI API |
| ACI Bearer Token | Authentication token provided by ACI |
| Allow customers to save card during checkout? | Checking this flag will allow customers to save cards during checkout |
| Capture on order confirmation? | Determines if the payment needs to be captured during order placement or to only pre-auth a payment during order placement. |
| ACI Entity ID | Entity ID for ACI authentication |
| ACI ForceResultCode | Set this value to force ACI to return specific error codes for example, 800.100.100. This is only for testing purposes and should not be set in LIVE environment |
| ACI Transaction Mode | Set this value to "INTERNAL" or EXTERNAL in all non-production instances and to "LIVE" in production instance. |
| Summary page in checkout flow? | Flag to indicate if the merchant uses a summary page. Set this value to true if your site has a summary page in checkout flow. |
| ACI Widget URL | ACI CopyPay Widget URL, for example - <https://test.oppwa.com/v1> for test environment |
| ACI Notification Emails | Notification emails in case of failure while reversing transaction in place order flow. Multiple emails IDs will be comma separated |
| ACS transaction ID | This field contains a universally unique transaction identifier assigned by the ACS to identify a single transaction. |
| Authentication Type | The type of authentication that was requested by the ACS. |
| 3DS Requestor Challenge Indicator | Indicates whether a challenge is requested for this transaction. |
| ACS Challenge mandate indicator | Indication of if a challenge is required for the transaction to be authorized due to local/regional mandates or other variable. |
| Directory Server transaction ID | Transaction ID assigned by the directory server |
| Exemption flag | Flags the transaction as exemption during authorization. Possible values: |
| Transaction status reason | Provides information on why the Transaction Status field has the specified value. |
| 3D Secure version | Version of 3D (examples: 1.0.2, 2.1.0, 2.2.0) |

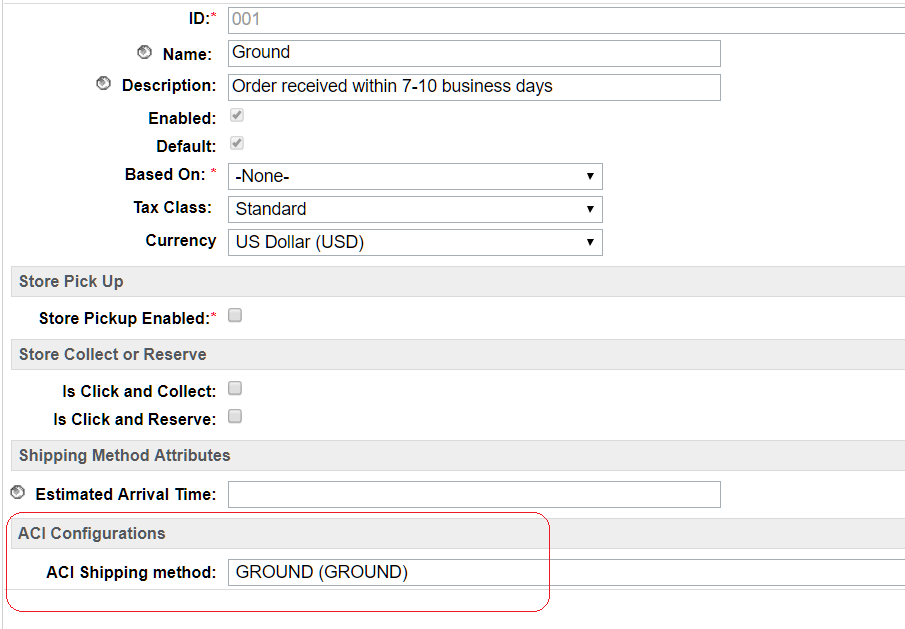
### ACI Merchant Configurations

Merchant configurations can be set to override the information that is shown on the cardholder statement. Setting this is not mandatory and it needs to be set only if the details about the client needs to be overridden in cardholder statement.

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Merchant Country | The country of the merchant. This can be configured to override the merchant country shown in the card holder statement in ACI |
| ACI Merchant City | The city of the merchant. This can be configured to override the merchant city of the card holder statement in ACI |
| ACI Merchant Category code | The merchants's category code. This can be configured to override the merchant category code shown in the card holder statement in ACI |
| ACI Merchant Name | Name of the merchant. This can be configured to override the merchant name of the card holder statement in ACI |
| ACI Merchant Phone | The merchants's phone number. This can be configured to override the merchant phone number shown in the card holder statement in ACI |
| ACI Merchant PostCode | The postal code or zip code of the merchant. This can be configured to override the merchant postcard of the card holder statement in ACI |
| ACI Merchant State | The county, state or region of the merchant. This field can be used to override the information that is shown on the cardholder statement |
| ACI Merchant Street | The building number and/or street name of the merchant. This can be configured to override the merchant address of the card holder statement in ACI |

## Configure Shipping Method Custom Attribute

A custom attribute, ACI\_ShippingMethod, of type “Enum of Strings” is added to Shipping Method object. Each of the shipping methods configured in the site needs to be mapped to one of the values in the list as this is required in the API call to ACI as mentioned in ACI documentation - <https://docs.oppwa.com/reference/parameters#shipping-address>

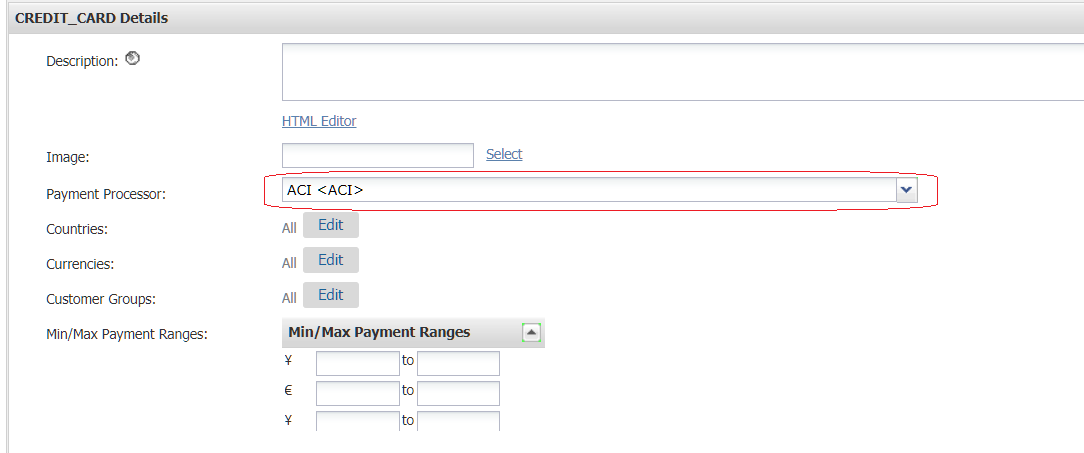


## Configuring Payment Methods

### Configuring Credit Cards

Please find below the steps to configure CREDIT\_CARD payment method for ACI.

* Change the payment processor of the payment method to ACI

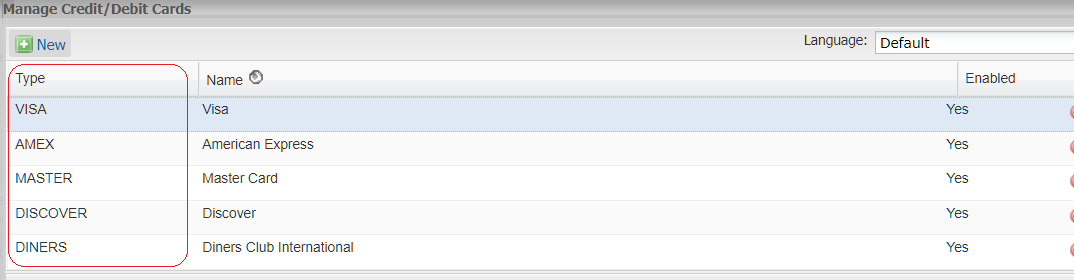


#### Configure card types

The card type of credit or debit cards must match the Brand ID specified in the ACI documentation - <https://docs.oppwa.com/tutorials/integration-guide>

Please follow below steps to configure the card types supported by ACI.

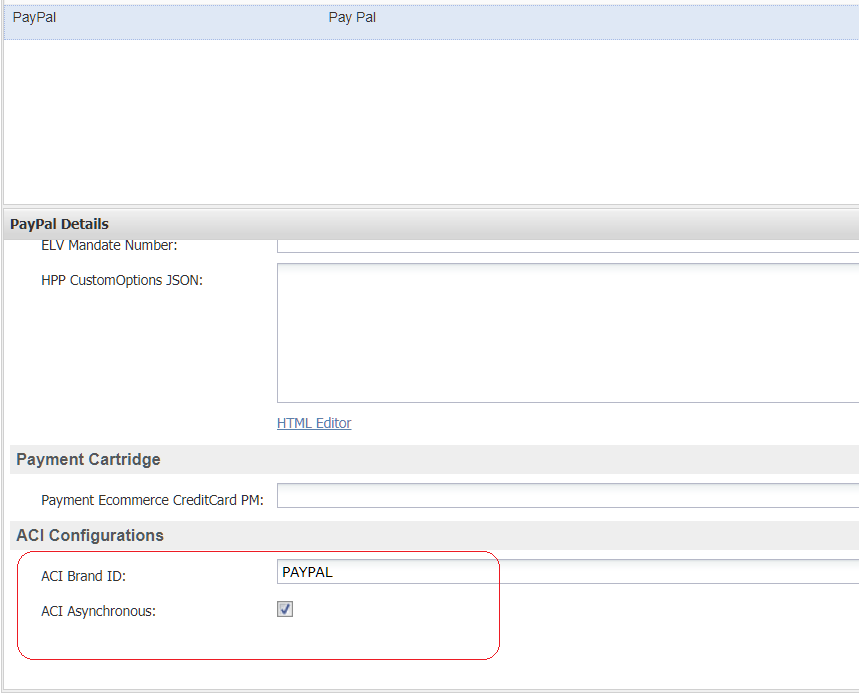
* Update the card type of existing cards to the Brand ID specified in ACI documentation - <https://docs.oppwa.com/tutorials/integration-guide>
* While adding new cards, ensure the card type matches the Brand ID required by ACI
* All asynchronous card types must be configured as separate individual payment methods



### Configure PayPal

Please follow the below steps to configure PayPal payment method to integrate with ACI

* Change the payment processor of the payment method to ACI
* Update the ACI Brand ID to PAYPAL
* Check the ACI Asynchronous flag to mark the payment method as asynchronous



### Configuring additional payment methods

Please follow the below steps to configure additional payment methods offered by ACI

* Change the payment processor of the payment method to ACI
* Update the ACI Brand ID to the brand id specified in ACI documentation. Please refer <https://docs.oppwa.com/reference/brands-reference> for more details.
* Update the ACI Asynchronous flag, if the payment method is asynchronous. Please refer <https://docs.oppwa.com/reference/brands-reference> for more details.

### Available payment methods

Please follow the below steps to configure additional payment methods offered by ACI

* Credit Card
* PayPal

Please follow the below documentation to configure additional payment methods offered by ACI.

ACI documentation - <https://docs.oppwa.com/tutorials/integration-guide>

# Storefront Cartridge Changes

## Controller Changes

### COBilling.js

**Changes to redirect to ACI Hosted payment page**

Following changes applies to save action in the billing function.

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| var Transaction = require('dw/system/Transaction');  var URLUtils = require('dw/web/URLUtils');  var Countries = require('app\_storefront\_core/cartridge/scripts/util/Countries');  /\* Script Modules \*/  var app = require('~/cartridge/scripts/app'); | var Transaction = require('dw/system/Transaction');  var URLUtils = require('dw/web/URLUtils');  var Countries = require('app\_storefront\_core/cartridge/scripts/util/Countries');  // ACI\_19.1.0: START  var Site = require('dw/system/Site');  // ACI\_19.1.0: END  /\* Script Modules \*/  var app = require('~/cartridge/scripts/app');  var guard = require('~/cartridge/scripts/guard'); |
| // 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; | // Mark step as fulfilled  app.getForm('billing').object.fulfilled.value = true;  // ACI\_19.1.0: START  var paymentMethodID = app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value;  var paymentProcessor = '';  if(paymentMethodID) {  var paymentProcessor = PaymentMgr.getPaymentMethod(paymentMethodID).paymentProcessor.ID;  }  if (Site.getCurrent().getCustomPreferenceValue('ACI\_ENABLED')) {  var ACIHelper = require('\*/cartridge/scripts/aci/aciHelper');  if (ACIHelper.isACIPaymentMethod(paymentMethodID)) {  // Redirect to stand alone ACI hosted page in case of ACI payments  session.privacy.paymentMethodID = paymentMethodID;  response.redirect(URLUtils.url('ACI-ShowCheckoutWidget'));  } else {  // A successful billing page will jump to the next checkout step.  app.getController('COSummary').Start();  }  } else if(paymentProcessor==='ACI' || !paymentMethodID){  publicStart();  } else {  // A successful billing page will jump to the next checkout step.  app.getController('COSummary').Start();  }  // ACI\_19.1.0: END  return; |

The changed file will look like:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### COSummary.js

**Changes to handle redirect from ACI payment page**

Add below function to COSummary.js file.

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| None | //ACI\_19.1.0: START  /\*\*  \* Handles redirect from ACI payment widget page  \*  \*/  function redirectFromACI() {  var ACI = require('\*/cartridge/controllers/ACI');  var ACIResponse = ACI.HandleACIRedirect();  if(ACIResponse.error) {  response.redirect(ACIResponse.errorRedirectURL);  return;  } else {  submit();  }  }  //ACI\_19.1.0: END |
| /\*  \* Web exposed methods  \*/  /\*\* @see module:controllers/COSummary~Start \*/  exports.Start = guard.ensure(['https'], start);  /\*\* @see module:controllers/COSummary~Submit \*/  exports.Submit = guard.ensure(['https', 'post', 'csrf'], submit); | /\*  \* Web exposed methods  \*/  /\*\* @see module:controllers/COSummary~Start \*/  exports.Start = guard.ensure(['https'], start);  /\*\* @see module:controllers/COSummary~Submit \*/  exports.Submit = guard.ensure(['https', 'post', 'csrf'], submit);  //ACI\_19.1.0: START  /\*\* @see module:controllers/COSummary~Submit \*/  exports.ACIHostedPageSubmit = guard.ensure(['https'], submit);  /\*\* @see module:controllers/COSummary~handleACIPaymentResponse \*/  exports.ACIRedirect = guard.ensure(['https'], redirectFromACI);  //ACI\_19.1.0: END |

The changed file will look like:



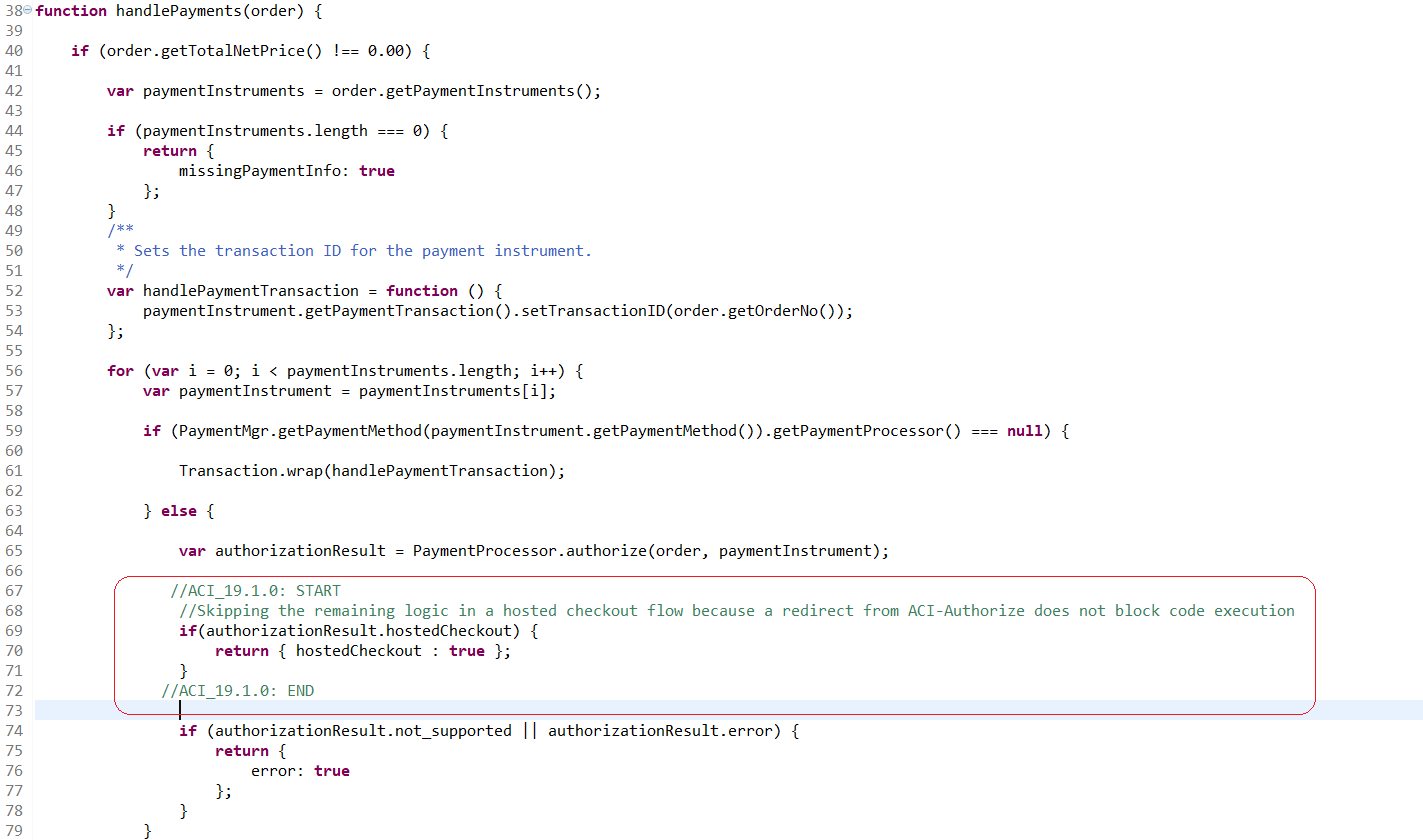
### COPlaceOrder.js

**Changes to skip execution after redirecting flow to ACI**

Add below code to function handlePayments () after authorize call.

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| if (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor() === null) {  Transaction.wrap(handlePaymentTransaction);  } else {  var authorizationResult = PaymentProcessor.authorize(order, paymentInstrument);  if (authorizationResult.not\_supported || authorizationResult.error) {  return {  error: true  };  }  } | if (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor() === null) {  Transaction.wrap(handlePaymentTransaction);  } else {  var authorizationResult = PaymentProcessor.authorize(order, paymentInstrument);    //ACI\_19.1.0: START  //Skipping the remaining logic in a hosted checkout flow because a redirect from ACI-Authorize does not block code execution  if(authorizationResult.hostedCheckout) {  return { hostedCheckout : true };  }  //ACI\_19.1.0: END    if (authorizationResult.not\_supported || authorizationResult.error) {  return {  error: true  };  }  } |

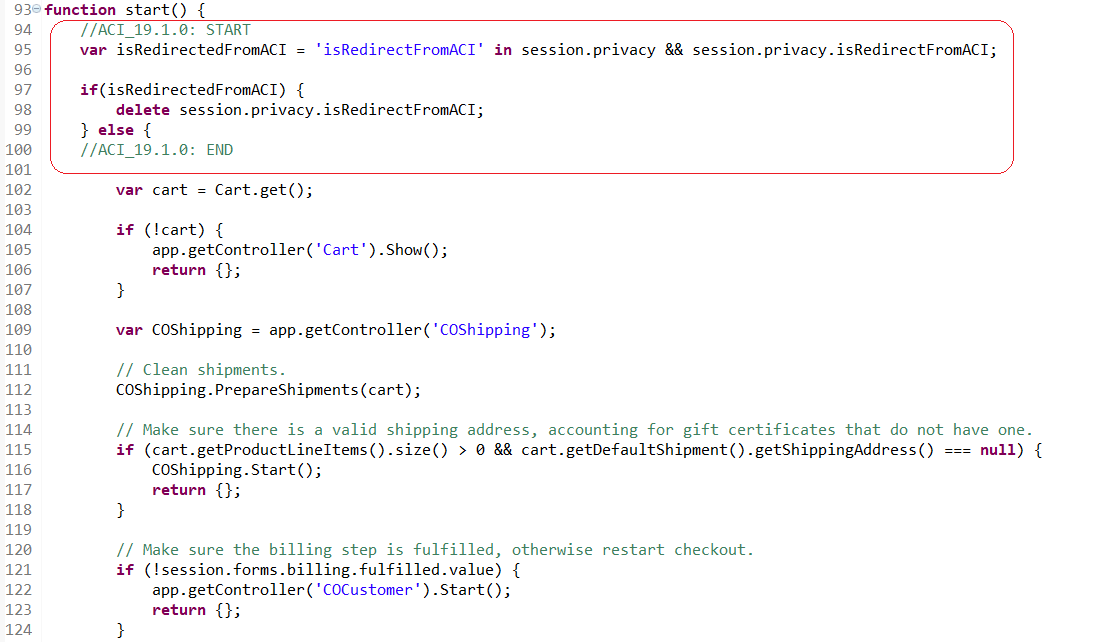
The changed file will look like:



**Changes to handle redirect from ACI**

Make below changes to start () function.

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| function start() {  var cart = Cart.get();  if (!cart) {  app.getController('Cart').Show();  return {};  } | function start() {  //ACI\_19.1.0: START  var isRedirectedFromACI = 'isRedirectFromACI' in session.privacy && session.privacy.isRedirectFromACI;    if(isRedirectedFromACI) {  delete session.privacy.isRedirectFromACI;  } else {  //ACI\_19.1.0: END    var cart = Cart.get();    if (!cart) {  app.getController('Cart').Show();  return {};  } |

The changed file:

**Changes to not create new order after redirect from ACI**

Make below changes to start () function

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| /\* API Includes \*/  var OrderMgr = require('dw/order/OrderMgr');  var PaymentMgr = require('dw/order/PaymentMgr');  var Status = require('dw/system/Status');  var Transaction = require('dw/system/Transaction');  /\* Script Modules \*/  var app = require('~/cartridge/scripts/app');  var guard = require('~/cartridge/scripts/guard'); | /\* API Includes \*/  var OrderMgr = require('dw/order/OrderMgr');  var PaymentMgr = require('dw/order/PaymentMgr');  var Status = require('dw/system/Status');  var Transaction = require('dw/system/Transaction');  // ACI\_19.1.0: START  var Site = require('dw/system/Site');  //ACI\_19.1.0: END  /\* Script Modules \*/  var app = require('~/cartridge/scripts/app');  var guard = require('~/cartridge/scripts/guard'); |
| // Handle used addresses and credit cards.  var saveCCResult = COBilling.SaveCreditCard();  if (!saveCCResult) {  return {  error: true,  PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')  };  } | if (!Site.getCurrent().getCustomPreferenceValue('ACI\_ENABLED')) {  // Handle used addresses and credit cards.  var saveCCResult = COBilling.SaveCreditCard();  if (!saveCCResult) {  return {  error: true,  PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')  };  }  } |
| // Creates a new order. This will internally ReserveInventoryForOrder and will create a new Order with status  // 'Created'.  var order = cart.createOrder();  if (!order) {  app.getController('Cart').Show();  return {};  } | //ACI\_19.1.0: START  var order;    if (isRedirectedFromACI){  //If redirected from ACI no need to create new order as order is already created in previous flow  var orderNo = session.privacy.orderNo;  delete session.privacy.orderNo;    order = OrderMgr.getOrder(orderNo);  } else {  // Creates a new order. This will internally ReserveInventoryForOrder and will create a new Order with status  // 'Created'.  order = cart.createOrder();  }  //ACI\_19.1.0: END  if (!order) {  app.getController('Cart').Show();  return {};  } |

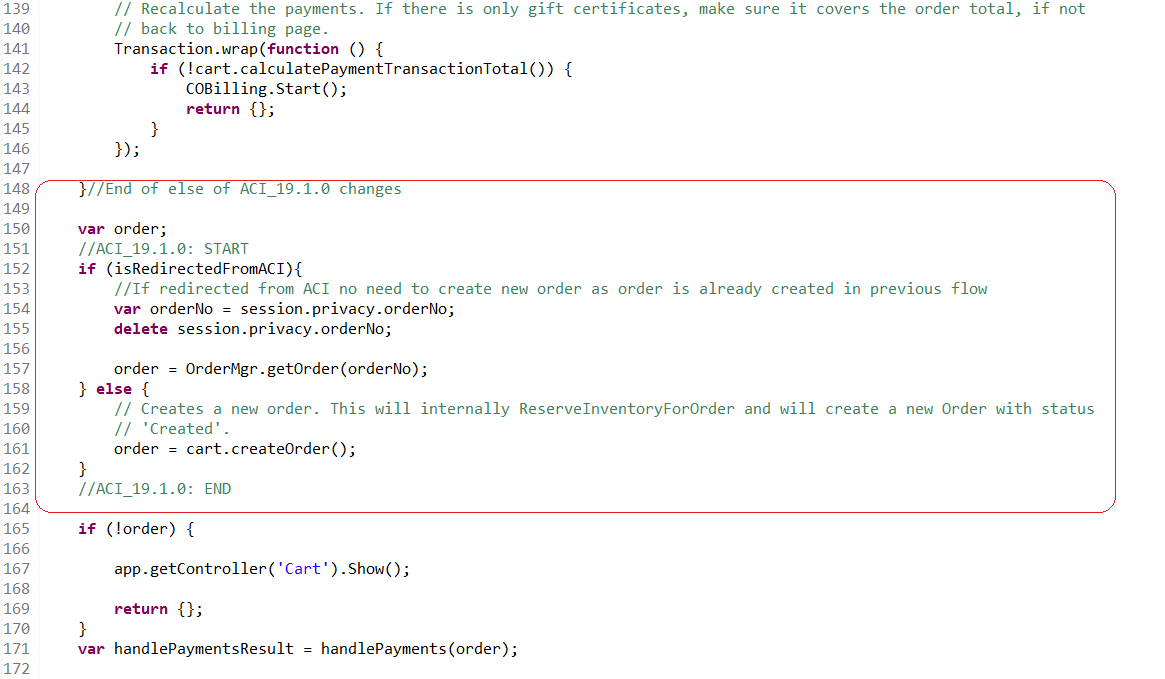
Changed file:

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

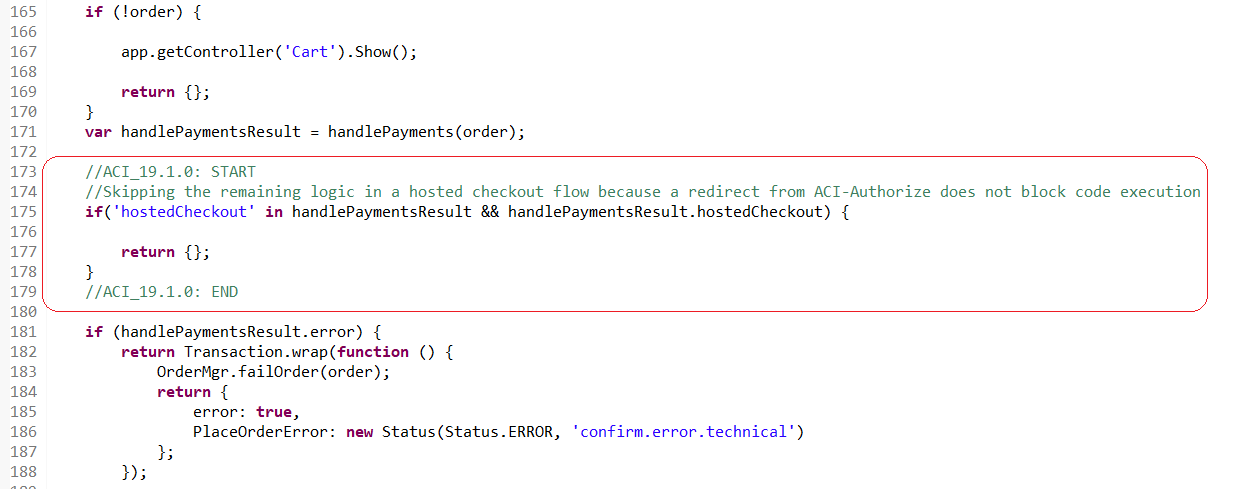


**Changes to skip execution after redirecting flow to ACI**

Add below code in start function after the call to handlePayments function

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| var handlePaymentsResult = handlePayments(order);  if (handlePaymentsResult.error) {  return Transaction.wrap(function () {  OrderMgr.failOrder(order);  return {  error: true,  PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')  };  });  } | var handlePaymentsResult = handlePayments(order);  //ACI\_19.1.0: START  //Skipping the remaining logic in a hosted checkout flow because a redirect from ACI-Authorize does not block code execution  if('hostedCheckout' in handlePaymentsResult && handlePaymentsResult.hostedCheckout) {    return {};  }  //ACI\_19.1.0: END  if (handlePaymentsResult.error) {  return Transaction.wrap(function () {  OrderMgr.failOrder(order);  return {  error: true,  PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')  };  });  } |

Changed file:

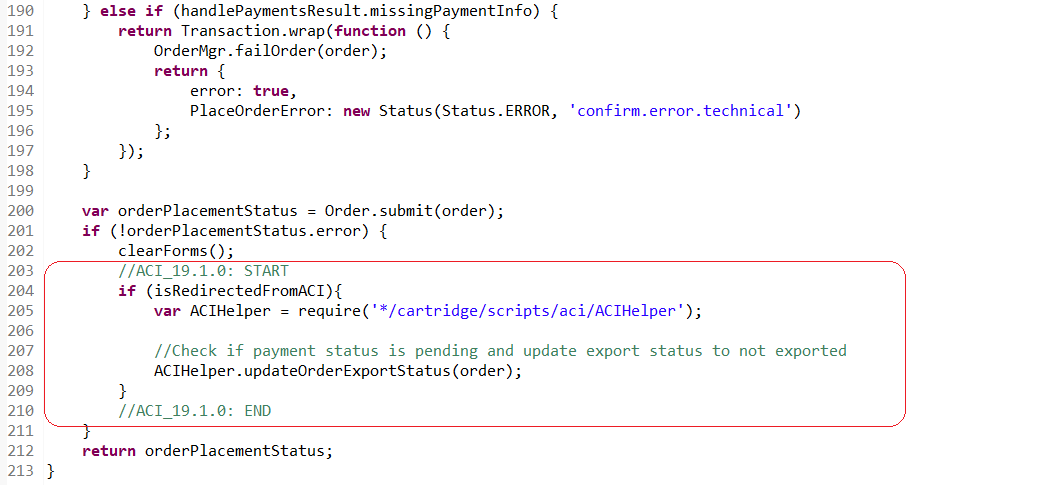


**Changes to update order export status in case of pending transactions**

Add below line in start function inside the “if” block after Order.submit (order) function call

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| var orderPlacementStatus = Order.submit(order);  if (!orderPlacementStatus.error) {  clearForms();  }  return orderPlacementStatus;  } | var orderPlacementStatus = Order.submit(order);  if (!orderPlacementStatus.error) {  clearForms();  //ACI\_19.1.0: START  if (isRedirectedFromACI){  var ACIHelper = require('\*/cartridge/scripts/aci/aciHelper');  //Check if payment status is pending and update export status to not exported  ACIHelper.updateOrderExportStatus(order);  }  //ACI\_19.1.0: END  }  return orderPlacementStatus;  } |

Changed file:



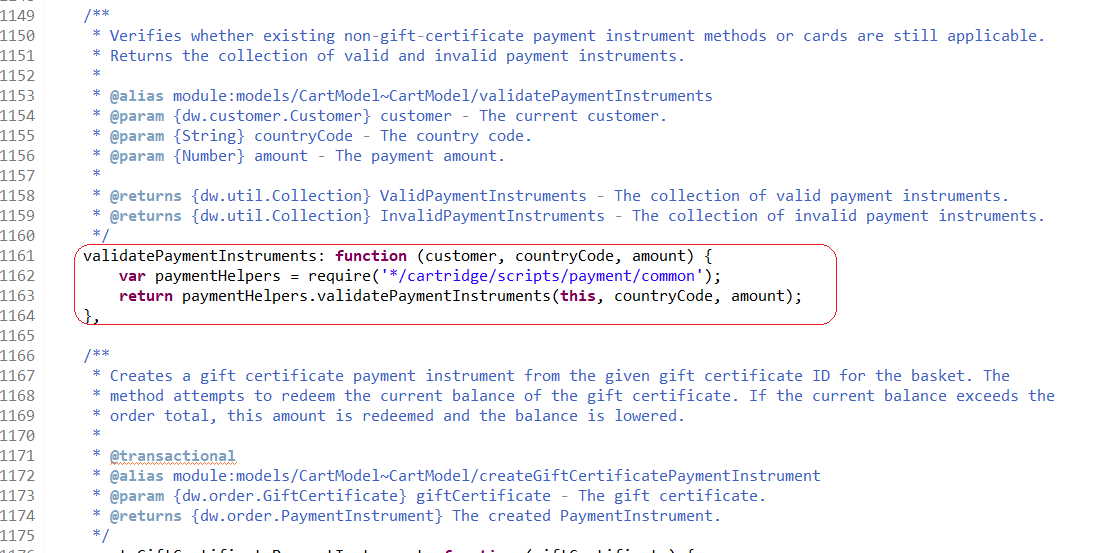
### CartModel.js

**Changes to add ACI specific validation**

Make changes to function validatePaymentInstruments () as shown below.

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| validatePaymentInstruments: function (customer, countryCode, amount) {  var paymentHelpers = require('~/cartridge/scripts/payment/common');  return paymentHelpers.validatePaymentInstruments(this, countryCode, amount);  }, | validatePaymentInstruments: function (customer, countryCode, amount) {  var paymentHelpers = require('\*/cartridge/scripts/payment/common');  return paymentHelpers.validatePaymentInstruments(this, countryCode, amount);  }, |

Changed file:



## ISML Changes

### billing.isml

Following changes needs to be added to billing.isml to display error messages in case of any issues in ACI payment. Add the changes before form declaration

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| <isif condition="${!pdict.CurrentForms.multishipping.entered.value}">  <ischeckoutprogressindicator step="2" multishipping="false" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'false' : 'true'}"/>  <iselse/>  <ischeckoutprogressindicator step="3" multishipping="true" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'false' : 'true'}"/>  </isif>  <form action="${URLUtils.continueURL()}" method="post" id="${pdict.CurrentForms.billing.htmlName}" class="checkout-billing address form-horizontal"> | <isif condition="${!pdict.CurrentForms.multishipping.entered.value}">  <ischeckoutprogressindicator step="2" multishipping="false" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'false' : 'true'}"/>  <iselse/>  <ischeckoutprogressindicator step="3" multishipping="true" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'false' : 'true'}"/>  </isif>  <iscomment> ACI\_19.1.0: START For showing aci errors</iscomment>  <isif condition="${!dw.system.Site.getCurrent().getCustomPreferenceValue('ACI\_ENABLED')}">  <isinclude template="aci/util/errormessages" />  </isif>  <iscomment> ACI\_19.1.0: END</iscomment>  <form action="${URLUtils.continueURL()}" method="post" id="${pdict.CurrentForms.billing.htmlName}" class="checkout-billing address form-horizontal"> |

Changed file:

A screenshot of a computer code

Description automatically generated

### paymentmethods.isml

|  |  |
| --- | --- |
| **OOB Code** | **Custom Code** |
| <div class="payment-method-options form-indent">  <isloop items="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.options}" var="paymentMethodType">  <iscomment>Ignore GIFT\_CERTIFICATE method, GCs are handled separately before other payment methods.</iscomment>  <isif condition="${paymentMethodType.value.equals(dw.order.PaymentInstrument.METHOD\_GIFT\_CERTIFICATE)}"><iscontinue/></isif>  <div class="form-row label-inline">  <isset name="radioID" value="${paymentMethodType.value}" scope="page"/>  <div class="field-wrapper">  <input id="is-${radioID}" type="radio" class="input-radio" name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlName}" value="${paymentMethodType.htmlValue}" <isif condition="${paymentMethodType.value == pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlValue}">checked="checked"</isif> />  </div>  <label for="is-${radioID}"><isprint value="${Resource.msg(paymentMethodType.label,'forms',null)}"/></label>  </div>  </isloop>  </div> | <div class="payment-method-options form-indent">  <isloop items="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.options}" var="paymentMethodType">  <iscomment>Ignore GIFT\_CERTIFICATE method, GCs are handled separately before other payment methods.</iscomment>  <isif condition="${paymentMethodType.value.equals(dw.order.PaymentInstrument.METHOD\_GIFT\_CERTIFICATE)}"><iscontinue/></isif>  <iscomment> ACI\_19.1.0: START </iscomment>  <isset name="paymentProcessor" value="${(dw.order.PaymentMgr.getPaymentMethod(paymentMethodType.value).paymentProcessor.ID)}" scope="page" />  <isif condition="${!paymentProcessor || !(paymentProcessor.equals('ACI'))}">  <div class="form-row label-inline">  <isset name="radioID" value="${paymentMethodType.value}" scope="page"/>  <div class="field-wrapper">  <input id="is-${radioID}" type="radio" class="input-radio"  name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlName}"  value="${paymentMethodType.htmlValue}"  <isif condition="${paymentMethodType.value == pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlValue}">  checked="checked"</isif> />  </div>  <label for="is-${radioID}"><isprint value="${Resource.msg(paymentMethodType.label,'forms',null)}"/></label>  </div>  <iselseif condition="${paymentProcessor.equals('ACI') && dw.system.Site.getCurrent().getCustomPreferenceValue('ACI\_ENABLED')}">  <div class="form-row label-inline">  <isset name="radioID" value="${paymentMethodType.value}" scope="page"/>  <div class="field-wrapper">  <input id="is-${radioID}" type="radio" class="input-radio"  name="${pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlName}"  value="${paymentMethodType.htmlValue}"  <isif condition="${paymentMethodType.value == pdict.CurrentForms.billing.paymentMethods.selectedPaymentMethodID.htmlValue}">  checked="checked"</isif> />  </div>  <label for="is-${radioID}"><isprint value="${Resource.msg(paymentMethodType.label,'forms',null)}"/></label>  </div>  </isif>  <iscomment> ACI\_19.1.0: END </iscomment>  </isloop>  </div> |
| <iscomment>  Credit card block  --------------------------------------------------------------  </iscomment>  <div class="payment-method <isif condition="${empty(pdict.selectedPaymentID) || pdict.selectedPaymentID=='CREDIT\_CARD'}">payment-method-expanded</isif>" data-method="CREDIT\_CARD"> | <iscomment>  Credit card block  --------------------------------------------------------------</iscomment>  <iscomment> ACI\_19.1.0: START Changed data-method of Credit card to ACI\_CREDIT\_CARD</iscomment>  <isset name="creditCardProcessor" value="${dw.order.PaymentMgr.getPaymentMethod('CREDIT\_CARD').paymentProcessor.ID}" scope="page" />  <isif condition="${creditCardProcessor.equals('ACI') && dw.system.Site.getCurrent().getCustomPreferenceValue('ACI\_ENABLED')}">  <div class="payment-method  <isif condition="${empty(pdict.selectedPaymentID) || pdict.selectedPaymentID=='CREDIT\_CARD'}">payment-method-expanded</isif>" data-method="Custom">  </div>  <iselseif condition="${!creditCardProcessor || !(creditCardProcessor.equals('ACI'))}"/>  <iscomment> ACI\_19.1.0: END </iscomment>  <div class="payment-method <isif condition="${empty(pdict.selectedPaymentID) || pdict.selectedPaymentID=='CREDIT\_CARD'}">payment-method-expanded</isif>" data-method="CREDIT\_CARD"> |
| <isif condition="${pdict.CurrentCustomer.authenticated}">  <isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.saveCard}" type="checkbox"/>  </isif>  </div>  <iscomment>  Bill me later | <isif condition="${pdict.CurrentCustomer.authenticated}">  <isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.saveCard}" type="checkbox"/>  </isif>  </div>  <iscomment> ACI\_19.1.0: START Closes the if-else loop checking if ACI is enabled </iscomment>  </isif>  <iscomment> ACI\_19.1.0: END </iscomment>  <iscomment>  Bill me later |
| <iscomment>  Custom processor  --------------------------------------------------------------  </iscomment>  <div class="payment-method <isif condition="${!empty(pdict.selectedPaymentID) && pdict.selectedPaymentID=='PayPal'}">payment-method-expanded</isif>" data-method="Custom">  <!-- Your custom payment method implementation goes here. -->  ${Resource.msg('billing.custompaymentmethod','checkout',null)}  </div>  </fieldset> | <iscomment>  Custom processor  --------------------------------------------------------------  </iscomment>  <iscomment> ACI\_19.1.0: START Display this code only when ACI is disabled</iscomment>  <isset name="paypalProcessor" value="${dw.order.PaymentMgr.getPaymentMethod('PAYPAL').paymentProcessor.ID}" scope="page" />  <isif condition="${!paypalProcessor || !(paypalProcessor.equals('ACI'))}">  <div class="payment-method  <isif condition="${!empty(pdict.selectedPaymentID) && pdict.selectedPaymentID=='PayPal'}">payment-method-expanded</isif>" data-method="Custom">  <!-- Your custom payment method implementation goes here. -->  ${Resource.msg('billing.custompaymentmethod','checkout',null)}  </div>  </isif>  <iscomment> ACI\_19.1.0: END </iscomment> ></fieldset> |

Changed file:

A screenshot of a computer program

Description automatically generated

A close-up of a text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

# Cartridge Extensions

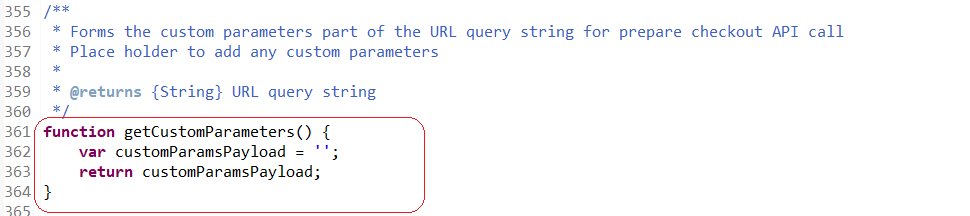
## Sending additional parameters to ACI

### Custom Parameters collected from browser

Any custom parameters collected from browser must be prefixed with “SHOPPER\_” before passing to ACI. For example, to send a custom parameter “taxType”, it must be prefixed as “SHOPPER\_taxType” and send in the form as “customParameters[SHOPPER\_taxType]” to ACI

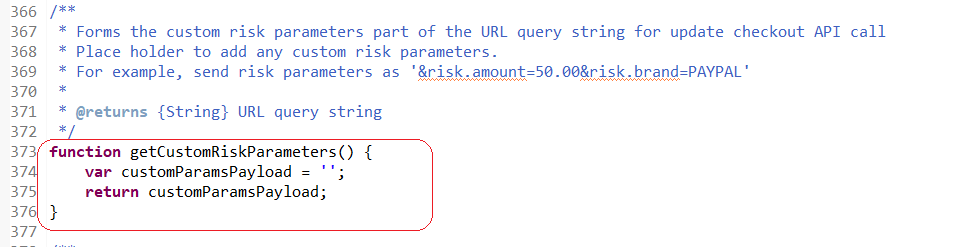
### Custom Risk Parameters

Script ACIServiceWrapper.js in path int\_aci\_core/cartridge/scripts/aci wraps all the changes required to make a call to the ACI APIs. Function getCustomRiskParameters () can be modified to send any additional risk parameters configured by merchants to ACI.



### Additional custom parameters

Function getCustomParameters () in script ACIServiceWrapper.js can be modified to send any additional parameters to ACI.



## Backoffice operations

The cartridge ‘**int\_ACI\_SG\_test**’ has implementation for the following API methods – Capture, Refund and Reversals. They are optional and requires custom integration. The implementation is available in script ***TransactionHelper.js*** in path ***int\_aci\_core/cartridge/scripts/aci/***

When site cartridge path has int\_ACI\_SG\_test  configured within it, then the back office test page will be accessible via controller ACITransactionSimulator-Test. But if site cartridge path does not have int\_ACI\_SG\_test configured within it, then the back office test page will not be accessible via controller ACITransactionSimulator-Test. An OOB error page would be thrown. Also when site cartridge path has int\_ACI\_SG\_test  configured within it, and the SFCC instance in Production, then the back office test page will not be accessible via controller ACITransactionSimulator-Test. Customer will be redirected to Error page.

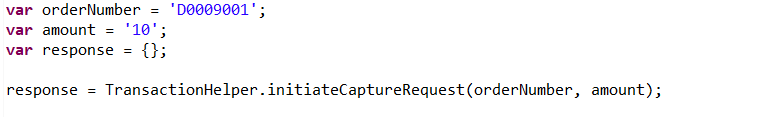
### Capture

The implementation supports full capture and partial captures. The function ***TransactionHelper.initiateCaptureRequest*** has the code required to handle capture requests. The function accepts the order number and the amount to be captured as arguments.

The function returns the below error codes:

|  |  |  |
| --- | --- | --- |
| **Error Code** | **Reason** | **Handling Error** |
| UNABLE\_TO\_FIND\_ORDER | Returned when not able to find an order corresponding to the order number passed in the argument. | Retry the operation with correct order number |
| ACI\_CAPTURE\_ERROR | Returned in one of the following scenarios:   * Invalid Capture Amount – amount to be captured exceeds order total plus already captured amount * ACI Service unavailable * ACI Service returned error response | Check the error log and ACI service logs and take appropriate action. |
| UNABLE\_TO\_FIND\_TRANSACTION | Returned when there is no transaction ID associated with the order | Manually capture the payment via ACI console |
| GENERAL\_CAPTURE\_ERROR | Returned in case of any exceptions other than above errors | Check the error log and ACI service logs and take appropriate action. |

Sample code to invoke capture request:



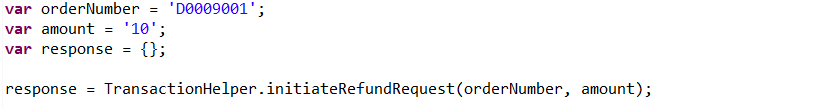
### Refund

The implementation supports full refund and partial refunds. The function ***TransactionHelper.initiateRefundRequest*** has the code required to handle refund requests. The function accepts the order number and the amount to be refunded as arguments.

The function returns the below error codes:

|  |  |  |
| --- | --- | --- |
| **Error Code** | **Reason** | **Handling Error** |
| UNABLE\_TO\_FIND\_ORDER | Returned when not able to find an order corresponding to the order number passed in the argument. | Retry the operation with correct order number |
| ACI\_REFUND\_ERROR | Returned in one of the following scenarios:   * Invalid refund amount – amount to be refunded exceeds order total plus already refunded amount * ACI Service unavailable * ACI Service returned error response | Check the error log and ACI service logs and take appropriate action. |
| UNABLE\_TO\_FIND\_TRANSACTION | Returned when there is no transaction ID associated with the order | Manually refund the payment via ACI console |
| GENERAL\_REFUND\_ERROR | Returned in case of any exceptions other than above errors | Check the error log and ACI service logs and take appropriate action. |

Sample code to invoke refund request:



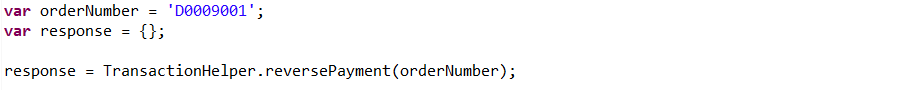
### Reversal

The implementation supports full reversal. The function ***TransactionHelper.reversePayment*** has the code required to handle reversal requests. The function accepts the order number as argument.

The function returns the below error codes:

|  |  |  |
| --- | --- | --- |
| **Error Code** | **Reason** | **Handling Error** |
| UNABLE\_TO\_FIND\_ORDER | Returned when not able to find an order corresponding to the order number passed in the argument. | Retry the operation with correct order number |
| ACI\_REVERSAL\_ERROR | Returned in one of the following scenarios:   * ACI Service unavailable * ACI Service returned error response | Check the error log and ACI service logs and take appropriate action. |
| UNABLE\_TO\_FIND\_TRANSACTION | Returned when there is no transaction ID associated with the order | Manually reverse the payment via ACI console |
| GENERAL\_REVERSAL\_ERROR | Returned in case of any exceptions other than above errors | Check the error log and ACI service logs and take appropriate action. |

Sample code to invoke reversal request:



## Widget Customization

### Extending Java script functionality

Script ***aciwidget.js*** in path ***int\_aci\_controllers\cartridge\static\default\js\aciwidget.js*** contains the java script functionality required for COPYPAY payment widget. The script can be modified if required to add or change any widget specific script changes explained in <https://docs.oppwa.com/tutorials/integration-guide/advanced-options>

### Modify widget styling

CSS file, ***aciwidget-style.css***, in path ***int\_aci\_controllers\cartridge\static\default\css\aciwidget-style.css*** can be modified to make any changes to the widget styling as explained in <https://docs.oppwa.com/tutorials/integration-guide/customisation#cssreference>

# Operations, Maintenance

## Availability

In case of problems with the connection to ACI Payments please contact the ACI Payments support department.

Please supply as much information as possible (Merchant account, time, order number, ACI Widget ID etc.).

## Failover / Recovery Process

In case the ACI service is unavailable, the user will not be able to checkout using ACI payment methods.

The service availability can be tracked in SFCC using the Service Status.

## Support

In case of problems with the integration, missing features, etc. please contact the ACI Payments support department or your ACI account manager.

# Appendix

## System Extensions

In addition to the changes mentioned above, the below system extensions were done as part of the cartridge.

### Basket

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Checkout ID | Unique identifier generated by ACI for the CopyAndPay widget for the customer's checkout journey |

### Order

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Checkout ID | Unique identifier generated by ACI for the CopyAndPay widget for the customer's checkout journey |
| ACI Payment Response | Saves the response returned as part of ACI Payment Status API call |
| Is ACI payment in pending status | Marks the order as pending if the transaction response from ACI is in pending status or requires manual review |

### PaymentMethod

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Brand ID | Indicates the brand id of the ACI payment method. Must match the value on ACI documentation. Please refer https://docs.oppwa.com/tutorials/integration-guide for more details |
| ACI Asynchronous | Indicates if the payment method is synchronous or asynchronous |

### PaymentTransaction

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Transaction Status Flow | Holds the transaction status flow for ACI payment transactions |

### ShippingMethod

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Shipping method | Parameter to be passed in the ACI PrepareCheckout request |

The following provides the list of possible values for the custom attribute under Shipping Method as mentioned in ACI documentation in <https://docs.oppwa.com/reference/parameters#shipping-address> :

* LOWEST\_COST
* CARRIER\_DESIGNATED\_BY\_CUSTOMER
* ELECTRONIC\_DELIVERY
* GROUND
* INTERNATIONAL
* MILITARY
* NEXT\_DAY\_OVERNIGHT
* OTHER
* STORE\_PICKUP
* SAME\_DAY\_SERVICE
* TWO\_DAY\_SERVICE
* THREE\_DAY\_SERVICE

### CustomerPaymentInstrument

|  |  |
| --- | --- |
| **Name** | **Purpose** |
| ACI Registration ID | Registration ID to be used for saving customer cards to account |