Version 23.1.0 for Salesforce Commerce Cloud - SiteGenesis

Upgrade Guide

# 

# 

# Summary

This guide describes how to upgrade your existing Afterpay SiteGenesis V2 cartridge to v23.1.0 and above.

Afterpay’s Version 23.1.0 cartridge provides the following features:

* Provides the Afterpay payment method during checkout
* Provides Afterpay Express Checkout feature
* Supports All Afterpay and Clearpay countries

## Where To Download The Cartridge

The cartridges and full documentation can be downloaded from:<https://github.com/afterpay/afterpay-salesforce-commerce-cloud>

# Table of Contents

[Summary](#_6z31qdnc5ute)

[Table of Contents](#_loz5ncc5r2xi)

[Component Overview](#_pbzy3wx36zo4)

[Upgrade Guide (for v20.0.0 and v21.0.0 cartridge users)](#_ioiq3x44klw0)

[Testing](#_bjfw6xbxekea)

[Features and Usage](#_u203niw5n203)

[Operations / Maintenance](#_b4ryz8nzy537)

[Release History](#_3vif62irv6v6)

# 

# Component Overview

## Functional Overview

Afterpay provides a payment processing gateway. Afterpay’s payment gateway connects your storefront to process transactions that allow customers to make purchases.

## Use Cases

Allows checkout using the Afterpay or Clearpay payment method at the checkout screen.

## Allows checkout using the Afterpay Express Checkout feature at the cart and product details screen.

## Limitations

The following Afterpay APIs are supported.

1. Direct Capture Payment
2. Authorize Payment
3. Create Checkout
4. Get Checkout
5. Get Configuration

The following Afterpay APIs are not supported or applicable

1. Void Authorization - Merchants will need to implement this in the event that orders can be cancelled from SFCC.
2. Capture Payment - If using our deferred payment flow the capture payment call will typically happen from an Order Management System
3. Update Shipping Courier - this will typically be implemented in the Order Management System.

## Compatibility

Available since SFCC 16.4  
SiteGenesis reference application integration examples and references version V2.

## Transaction Data Privacy

This integration requires access to the following customer data elements:

* Billing Address
* Shipping Address
* Order Details
* Customer Profile

No credit card details are stored within SFCC using this integration.

## Express Checkout

Express Checkout is a new feature which will reduce the overall checkout steps on desktop and mobile, so that your shoppers can complete their orders quickly. It enables shoppers to checkout directly from the shopping cart or product page; and use their previously provided information to Afterpay - name, shipping address, phone number, and email, to complete their orders on your website.

After adding Afterpay’s v23.1.0 cartridge, the following will be added to your site:

* “Checkout with Afterpay” buttons on the cart page
* “Buy Now with Afterpay” button on the product detail page.  
  (Optional but Highly Recommended)

When the shopper clicks on the checkout buttons, the following will occur:

* Express Checkout begins and Afterpay’s popup form appears.
* Shoppers who have used Afterpay in the past and have enabled saved shipping and payment information will be defaulted with those options and will not have to reenter that information but can change as desired.
* Price and shipping methods will be presented to the shopper (with integrated shipping option enabled).
* Upon clicking the confirmation button, the shopper will be redirected back to the merchant site for review or immediate order creation (BuyNow option enabled)

# Upgrade Guide (for v20.0.0 and v21.0.0 cartridge users)

Note: The link cartridge download will include a copy of all files that were changed from the base cartridge for SiteGenesis 105.1.0 . The edits in the integration guide for the v20.0.0 cartridge may differ slightly from the ones described in this guide. Specifically, most code blocks that were inserted will have comments surrounding them. For isml templates, the comments will generally be:

|  |
| --- |
| <iscomment>Start of Afterpay</iscomment>  ... <iscomment>End of Afterpay</iscomment> |

In Javascript sections they will be:

|  |
| --- |
| // Start of Afterpay  ... // End of Afterpay |

Thus exact line numbers in the edits below may differ slightly from your version.

This version of the cartridge does not support SiteGenesis pipelines. Please contact Afterpay directly for a cartridge containing support for pipelines.

## Download the cartridge

The current version of the Afterpay cartridge is available directly from Afterpay.

<https://github.com/afterpay/afterpay-salesforce-commerce-cloud>

The version hosted on the Salesforce Commerce Cloud Partner Marketplace is an older version and should not be used.

## Import the cartridge

Import the cartridge using UX Studio (or VSCode with the Prophet extension)

### Import the Afterpay V2.0 Cartridge into UX Studio

1. Download and unzip the cartridge.
2. In Salesforce UX Studio, change to the Digital Development view, and run the Import function.
3. In the Import window, click **General > Existing Projects into Workspace**
4. Select the location where you’ve unzipped the cartridge.
5. For SiteGenesis controllers, select the **bm\_afterpay, int\_afterpay\_core, and int\_afterpay\_sg** projects.
6. When the prompt “Do you want to attach the cartridges to a Digital Server” appears, click “Yes”. Select all the projects in the list and click “OK”.
7. If you were not prompted, return to the Project Explorer and right click the entry for your server. Click **Project References** and make sure to select all of the projects mentioned above.
8. Check to see that the new cartridge has been uploaded.

### 

## Import the site metadata

All import files can be found in the **metadata/** folder within cartridge installation.

For the Afterpay integration to work, the following object structures (metadata) needs to be imported and configured in the Business manager. Follow the below steps:

1. In the cartridge bundle find metadata/afterpay-meta-import folder.
2. Find the sites folder and inside it find the RefArch folder.
3. Rename this RefARch folder to the ID of your site. If you have multiple sites to which this meta has to be imported, then copy and paste this RefArch folder to replicate for other sites. Renames those folders with ID of corresponding sites.
4. In the cartridge bundle, inside the metadata folder compress the afterpay-meta-import folder to generate the afterpay-meta-import.zip file.
5. Go to Business Manager Menu > Administration > Site Development > Site Import & Export
6. Under Import: Upload Archive: Ensure that the radio button with label Local is enabled (Else click on the radio button to enable it)
7. Click on Choose File input field, select the afterpay-meta-import.zip file from open dialog box and click on upload button
8. After finishing the upload, from the Archives list click the radio button corresponding to afterpay-meta-import.zip and click on Import button
9. Click on the OK button of the confirmation box asking, “Are you sure that you want to import the selected archive?”

## Custom Code Changes

In order to integrate the new Afterpay cartridge, modifications need to be made to your version of the SiteGenesis base cartridge files. A 3-way visual merge tool such as Meld (<https://meldmerge.org/>) may be useful for making these changes if you’ve made considerable changes to the SiteGenesis base cartridge files for your own store.

#### File: **app\_storefront\_core/cartridge/scss/default/\_afterpay.scss**

Copy the **\_afterpay.scss** file from the **reference** folder of the v2.0 cartridge into **app\_storefront\_core/cartridge/scss/default/\_afterpay.scss**

#### File: **app\_storefront\_controllers/cartridge/controllers/COBilling.js**

In the function resetPaymentForms(), add the highlighted line into the 3 locations:

|  |
| --- |
| var status = Transaction.wrap(function () {  if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals('PayPal')) {  app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();  app.getForm('billing').object.paymentMethods.bml.clearFormElement();   cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD));  cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_BML));  // removes Afterpay PaymentInstrument  require('int\_afterpay\_sg/cartridge/scripts/checkout/afterpayHandlers.js').handleChangedPaymentInstrument();  } else if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_CREDIT\_CARD)) {  app.getForm('billing').object.paymentMethods.bml.clearFormElement();   cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_BML));  cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));   // removes Afterpay PaymentInstrument  require('int\_afterpay\_sg/cartridge/scripts/checkout/afterpayHandlers.js').handleChangedPaymentInstrument();  } else if (app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value.equals(PaymentInstrument.METHOD\_BML)) {  app.getForm('billing').object.paymentMethods.creditCard.clearFormElement();   if (!app.getForm('billing').object.paymentMethods.bml.ssn.valid) {  return false;  }   cart.removePaymentInstruments(cart.getPaymentInstruments(PaymentInstrument.METHOD\_CREDIT\_CARD));  cart.removePaymentInstruments(cart.getPaymentInstruments('PayPal'));   // removes Afterpay PaymentInstrument  require('int\_afterpay\_sg/cartridge/scripts/checkout/afterpayHandlers.js').handleChangedPaymentInstrument();  }  return true;  }); |

In the start() function, paste the highlighted code:

|  |
| --- |
| pageMeta.update({  pageTitle: Resource.msg('billing.meta.pagetitle', 'checkout', 'SiteGenesis Checkout')  });  // Start of Afterpay  require('int\_afterpay\_sg/cartridge/scripts/checkout/afterpayHandlers.js').handleBillingStart();  // End of Afterpay   returnToForm(cart, params); |

Also, in the same file, add highlighted lines to handlePaymentSection function:

|  |
| --- |
| function handlePaymentSelection(cart) {  var afterpayBrandUtilities = AfterpayUtilities.brandUtilities;  var selectedPaymentMethodID = app.getForm('billing').object.paymentMethods.selectedPaymentMethodID.value;  var result;    if (empty(selectedPaymentMethodID)) {  if (cart.getTotalGrossPrice() > 0) {  result = {  error: true  };  } else {  result = {  ok: true  };  }  }    if ((selectedPaymentMethodID == 'AFTERPAY' || selectedPaymentMethodID == 'CLEARPAY') && !afterpayBrandUtilities.isAfterpayApplicable()) {  result = {  error: true  };  }    // skip the payment handling if the whole payment was made using gift cert  if (selectedPaymentMethodID.equals(PaymentInstrument.METHOD\_GIFT\_CERTIFICATE)) {  result = {  ok: true  };  }    if (empty(PaymentMgr.getPaymentMethod(selectedPaymentMethodID).paymentProcessor)) {  result = {  error: true,  MissingPaymentProcessor: true  };  }  if (!result) {  result = app.getModel('PaymentProcessor').handle(cart.object, selectedPaymentMethodID);  }  return result; } |

Also, in the same file, add highlighted lines to billing function:

|  |
| --- |
| function billing() {    app.getForm('billing').handleAction({  applyCoupon: function () {  var couponCode = request.httpParameterMap.couponCode.stringValue || request.httpParameterMap.dwfrm\_billing\_couponCode.stringValue;    // TODO what happened to this start node?  app.getController('Cart').AddCoupon(couponCode);    handleCoupon();  return;  },  creditCardSelect: function () {  updateCreditCardSelection();  return;  },  paymentSelect: function () {  // ToDo - pass parameter ?  publicStart();  return;  },  redeemGiftCert: function () {  var status = redeemGiftCertificate(app.getForm('billing').object.giftCertCode.htmlValue);  if (!status.isError()) {  returnToForm(app.getModel('Cart').get(), {  NewGCPaymentInstrument: status.getDetail('NewGCPaymentInstrument')  });  } else {  returnToForm(app.getModel('Cart').get());  }    return;  },  save: function () {  Transaction.wrap(function () {  var cart = app.getModel('Cart').get();  var resetPaymentFormsResult = resetPaymentForms();  var validateBillingResult = validateBilling();  var handleBillingAddressResult = handleBillingAddress(cart);  var handlePaymentSelectionResult = handlePaymentSelection(cart);    if (!resetPaymentFormsResult || !validateBillingResult || !handleBillingAddressResult || // Performs validation steps, based upon the entered billing address  // and address options.  handlePaymentSelectionResult.error) {// Performs payment method specific checks, such as credit card verification.  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.  if (!handlePaymentSelectionResult.redirect) {  app.getController('COSummary').Start();  }  return;  }  });  },  selectAddress: function () {  updateAddressDetails();  return;  }  }); } |

#### File: **app\_storefront\_controllers/cartridge/controllers/COSummary.js**

In the start() function, remove the line:

|  |
| --- |
| app.getView(viewContext).render('checkout/summary/summary'); |

And paste in the following:

|  |
| --- |
| require("\*/cartridge/scripts/checkout/ afterpaySGCheckoutHelpers.js").disableSummaryForAfterpay(cart, viewContext); |

#### File: **app\_storefront\_controllers/cartridge/controllers/COPlaceOrder.js**

In the handlePayments() function, paste the following:

|  |
| --- |
| if (authorizationResult.not\_supported || authorizationResult.error) {  return {  error: true  }; } |

With:

|  |
| --- |
| if (authorizationResult.not\_supported || authorizationResult.error) {  return {  authorizeError : authorizationResult.authorizationResponse,  error: true  }; } |

Replace the code block:

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

with:

|  |
| --- |
| if (handlePaymentsResult.error) {  return Transaction.wrap(function () {  OrderMgr.failOrder(order);  return {  error: true,  afterpayOrderAuthorizeError: handlePaymentsResult.authorizeError,  PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')  };  }); |

#### File: **app\_storefront\_controllers/cartridge/controllers/Currency.js**

|  |
| --- |
| function setSessionCurrency() {  var afterpayUtilities = require('\*/cartridge/scripts/util/afterpayUtilities');  var currencyMnemonic = request.httpParameterMap.currencyMnemonic.value;  var locale = request.httpParameterMap.locale.value;  var Response = require('\*/cartridge/scripts/util/Response');  var currency;   if (currencyMnemonic) {  currency = Currency.getCurrency(currencyMnemonic);  if (currency) {  session.setCurrency(currency);   Transaction.wrap(function () {  var currentCart = Cart.get();  if (currentCart) {  currentCart.updateCurrency();  currentCart.calculate();  }  });  }  }   if (afterpayUtilities.sitePreferencesUtilities.isAfterpayEnabled() && locale) {  afterpayUtilities.brandUtilities.initBrand(locale);  }   Response.renderJSON({  success: true  }); } |

#### File: **app\_storefront\_controllers/cartridge/controllers/COShipping.js**

Near the bottom of function selectShippingMethod(), paste the highlighted code :

|  |
| --- |
| Transaction.wrap(function () {  cart.updateShipmentShippingMethod(cart.getDefaultShipment().getID(), request.httpParameterMap.shippingMethodID.stringValue, null, applicableShippingMethods);  cart.calculate();  });  // Start of Afterpay  require('int\_afterpay\_sg/cartridge/scripts/checkout/afterpayHandlers.js').handleShippingMethodUpdate();  // End of Afterpay  app.getView({  Basket: cart.object  }).render('checkout/shipping/selectshippingmethodjson'); |

#### File: **app\_storefront\_core/cartridge/templates/default/components/header/htmlhead\_UI.isml**

Locate the following code:

|  |
| --- |
| <!--[if lt IE 9]> <script src="${URLUtils.staticURL('/js/lib/html5.js')}"></script> <![endif]--> |

And add this code after it:

|  |
| --- |
| <isinclude url="${URLUtils.url('Afterpay-IncludeAfterpayLibrary')}" /> |

#### File: **app\_storefront\_core/cartridge/templates/default/checkout/billing/billing.isml**

Locate the

|  |
| --- |
| <**iscomment**> +++++++++++++++++++++billing address</**iscomment**> |

and paste the code below after the comment as seen in the screenshot:

|  |
| --- |
| <**iscomment**>Start of Afterpay</**iscomment**> <**isset** name="afterpayError"  value="${!empty(pdict.AfterpayApiError) ? pdict.AfterpayApiError :  (!empty(request.httpParameterMap.get('afterpay'))  .stringValue ? request.httpParameterMap.get('afterpay').stringValue : null)}"  scope="page" />  <**isif** condition="${!empty(afterpayError)}">  <**div** class="error-form">  <**isprint** value="${afterpayError}" encoding="off" />  </**div**> </**isif**>  <**iscomment**>End of Afterpay</**iscomment**> |

#### File: **app\_storefront\_core/cartridge/templates/default/checkout/billing/paymentmethods.isml**

After the line:

|  |
| --- |
| <**isinclude** template="util/modules"/> |

Paste the following code as show in the screenshot:

|  |
| --- |
| <iscomment>Start of Afterpay</iscomment> isscript>  var AfterpayUtilities = require("\*/cartridge/scripts/util/afterpayUtilities.js");  var brandUtilities = AfterpayUtilities.brandUtilities;   brandUtilities.initBrand(pdict.CurrentRequest.locale);   var afterpayApplicable = brandUtilities.isAfterpayApplicable();  var apBrand = brandUtilities.getBrand();  var basketObject = dw.order.BasketMgr.getCurrentBasket();  var orderGrandTotal = basketObject.totalGrossPrice;  var thresholdUtilities = apBrandUtilities.getApiVersionDependentClass('\*/cartridge/scripts/util/thresholdUtilities');  var withinThreshold = thresholdUtilities.checkThreshold(orderGrandTotal);   afterpayApplicable = afterpayApplicable && withinThreshold.status && !withinThreshold.belowThreshold;   var afterpayPaymentName = ''; </isscript>  <iscomment>End of Afterpay</iscomment> |

After the line:

|  |
| --- |
| <**isif** condition="${paymentMethodType.value.equals(dw.order.PaymentInstrument.METHOD\_GIFT\_CERTIFICATE)}"><**iscontinue**/></**isif**> |

Paste the following:

|  |
| --- |
| <iscomment>Start of Afterpay</iscomment> <iscomment>Ignore the payment method AFTERPAY\_PBI if Afterpay payment method is disabled in configuration</iscomment> <isif condition="${(paymentMethodType.value.equals('AFTERPAY') || paymentMethodType.value.equals('CLEARPAY')) && !afterpayApplicable}">  <iscontinue/> </isif>  <isif condition="${(paymentMethodType.value.equals('AFTERPAY') || paymentMethodType.value.equals('CLEARPAY'))}">  <isset name="afterpayPayment" value="${paymentMethodType.value}" scope="page"/> </isif>  <iscomment>End of Afterpay</iscomment> |

Find and comment out the following code block:

|  |
| --- |
| <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> |

And replace with the following:

|  |
| --- |
| <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}">  <isset name="paymentObject" value="${paymentMethodType.object}" scope="page"/>  <isif condition="${empty(paymentObject.image)}">  <isprint value="${Resource.msg(paymentMethodType.label,'forms',null)}"/>  <iselse/>  <img src="${paymentObject.image.URL}" class="payment-image ${radioID.toLowerCase()}-image" alt="${Resource.msg('billing.logo', apBrand, null)}"/>  </isif>  </label> </div> |

Locate the following block of code:

|  |
| --- |
| <**iscomment**> Custom processor -------------------------------------------------------------- </**iscomment**> |

And paste the following below that code block, as seen in the screenshot:

|  |
| --- |
| iscomment>  Afterpay Pay Over Time  -------------------------------------------------------------- </iscomment> <iscomment>Start of Afterpay</iscomment> <isif condition="${afterpayApplicable && !empty(afterpayPayment)}">  <div class="payment-method <isif condition="${!empty(pdict.selectedPaymentID) && pdict.selectedPaymentID == afterpayPayment}"> payment-method-expanded </isif>" data-method="${afterpayPayment}">  <isafterpaymessage  classname="checkout-afterpay-message"  totalprice="${orderGrandTotal}"  afterpaybrand="${apBrand}"  />  </div> </isif>  <iscomment>End of Afterpay</iscomment> |

#### File: **app\_storefront\_core/cartridge/templates/default/product/components/pricing.isml**

Locate this code:

|  |
| --- |
| <**isinclude** template="product/components/standardprice"/> |

And paste the code below:

|  |
| --- |
| <**iscomment**>Start of Afterpay</**iscomment**> <**isinclude** template="util/modules"> <**iscomment**>End of Afterpay</**iscomment**> |

At the very bottom of the file, right before the last <isif>, paste the following:

|  |
| --- |
| <isscript>  var priceForAp;   if (pdict.ProductSetStandardPrice || pdict.ProductSetSalesPrice) {  priceForAp = pdict.ProductSetSalesPrice < pdict.ProductSetStandardPrice ? pdict.ProductSetSalesPrice : pdict.ProductSetStandardPrice;  } else {  priceForAp = pdict.Product.priceModel.isPriceRange() > 0 ? pdict.Product.priceModel.minPrice : (SalesPrice ? SalesPrice : StandardPrice);  }  var sitePreferences = require('\*/cartridge/scripts/util/afterpayUtilities').sitePreferencesUtilities;  var afterpayPDPEnable = sitePreferences.isDisplayPdpInfo(); </isscript> <isif condition="${afterpayPDPEnable}">  <isinclude url="${URLUtils.url('Afterpay-RenderMessage',  'totalprice', priceForAp,  'classname', 'pdp-afterpay-message',  'quickview', isQuickView,  'afterpayBrand', pdict.CurrentSession.privacy.afterpayBrand  )}" /> </isif> |

#### File: **app\_storefront\_core/cartridge/templates/default/product/producttile.isml**

At the top of the file, right after the </isif>, paste the following:

|  |
| --- |
| <**iscomment**>Start of Afterpay</**iscomment**> <**isinclude** template="util/modules"></**isinclude**> <**iscomment**>End of Afterpay</**iscomment**> |

Around line 244 (after the above changes), comment out the following code:

|  |
| --- |
| prices.push(price); if (extraPrice) {prices.push(extraPrice);} |

Replace it by pasting in the following::

|  |
| --- |
| // Start of Afterpay if (price && price.value) {prices.push(price);} if (extraPrice && extraPrice.value) {prices.push(extraPrice);} // End of Afterpay |

Around line 254 (after the above changes), right before the line:

|  |
| --- |
| <**iscomment**>Promotion</**iscomment**> |

Paste in the following as shown:

|  |
| --- |
| <iscomment>Start of Afterpay</iscomment> <iscomment>Afterpay message</iscomment> <isscript>  var AfterpayUtilities = require('\*/cartridge/scripts/util/afterpayUtilities');  var BrandUtilities = AfterpayUtilities.brandUtilities;  var priceForAp = pdict.CurrentHttpParameterMap.pricerange.stringValue == 'true'? pdict.CurrentHttpParameterMap.minprice : prices[prices.length-1].value;   BrandUtilities.initBrand(pdict.CurrentRequest.locale);  var sitePreferences = AfterpayUtilities.sitePreferencesUtilities;  var afterpayPLPEnable = sitePreferences.isDisplayPlpInfo();  </isscript>  <isif condition="${afterpayPLPEnable}">   <isinclude url="${URLUtils.url('Afterpay-RenderMessage',  'totalprice', priceForAp,  'classname', 'plp-afterpay-message',  'afterpayBrand', pdict.CurrentSession.privacy.afterpayBrand  )}"  /> </isif><iscomment>End of Afterpay</iscomment> |

#### File: **app\_storefront\_core/cartridge/templates/default/checkout/cart/cart.isml**

Near the top of the file, After

|  |
| --- |
| <isinclude template="util/reporting/ReportBasket.isml" /> |

paste the following as shown:

|  |
| --- |
| <iscomment>Start of Afterpay Express</iscomment> <isif condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('enableAfterpay')  && dw.system.Site.getCurrent().getCustomPreferenceValue('apEnableExpressCheckout')}">  <isafterpayexpresscheckout basket="${pdict.Basket}">  <iscomment>  If the delivery option is changed, do ajax call to server to check whether all items in  cart are being picked up in-store. Call Afterpay Express widget with "pickup" setting.  </iscomment>  <script type="text/javascript">  document.addEventListener("DOMContentLoaded", function() {  console.log("afterpay express js loaded");  initAfterpay({pickupflag: $('#afterpay-express-storepickup').val() === "true"});  initializeDeliveryOptionChangeListener();  });  </script>  </isif> <iscomment>End of Afterpay Express</iscomment> |

Near line 52, under the <button> element, paste the following code:

|  |
| --- |
| <iscomment>Start of Afterpay Express</iscomment> <isif condition="${afterpayExpressCheckoutEnabled && !disableAfterpayPaymentMethod}">  <isscript>  var AfterpayUtilities = require('\*/cartridge/scripts/util/afterpayUtilities');  var BrandUtilities = AfterpayUtilities.brandUtilities;  var countrycode = BrandUtilities.getCountryCode();  var checkoutButtonClass = 'afterpay-checkout-button';  if(countrycode === 'GB'){  checkoutButtonClass = 'clearpay-checkout-button';  }  </isscript>  <div>  <isif condition="${isExpressCheckoutFinalize}">  <a href="${URLUtils.url('AfterpayExpress-ContinueFinalize')}">  <input type="button" class="button-fancy-large ${checkoutButtonClass}" id="afterpay-continue-button" />  </a>  <iselse/>  <input type="button" class="button-fancy-large ${checkoutButtonClass}" id="afterpay-express-button" data-afterpay-entry-point="cart" />  </isif>  </div> </isif> <iscomment>End of Afterpay Express</iscomment> |

Locate the following <div>:

|  |
| --- |
| <**div** class="cart-actions"> |

Right above it, paste the following:

|  |
| --- |
| <iscomment>Start of Afterpay</iscomment> <isscript>  var BrandUtilities = require('\*/cartridge/scripts/util/afterpayUtilities').brandUtilities;  BrandUtilities.initBrand(pdict.CurrentRequest.locale); </isscript>  <div class="learn-more">   <isinclude url="${URLUtils.url('Afterpay-RenderMessage',  'totalprice', orderTotalValue,  'classname', 'cart-afterpay-message',  'afterpayBrand', pdict.CurrentSession.privacy.afterpayBrand  )}"  /> </div> <iscomment>End of Afterpay</iscomment> |

Near line 910, under the <button> element, paste the following code:

|  |
| --- |
| <iscomment>Start of Afterpay Express</iscomment> <isif condition="${afterpayExpressCheckoutEnabled && !disableAfterpayPaymentMethod}">  <div>  <isif condition="${isExpressCheckoutFinalize}">  <a href="${URLUtils.url('AfterpayExpress-ContinueFinalize')}">  <input type="button" class="button-fancy-large ${checkoutButtonClass}" id="afterpay-continue-button" />  </a>  <iselse/>  <input type="button" class="button-fancy-large ${checkoutButtonClass}" id="afterpay-express-button" data-afterpay-entry-point="cart" />  </isif>  </div>  </isif> <iscomment>End of Afterpay Express</iscomment> |

#### File: **app\_storefront\_core/cartridge/templates/default/util/modules.isml**

At the very bottom of the file, paste the following:

|  |
| --- |
| <**iscomment**>Start of Afterpay</**iscomment**> <**iscomment**> Render the Afterpay modules </**iscomment**> <**isinclude** template="util/modulesafterpay" /> <**iscomment**>End of Afterpay</**iscomment**> |

#### File: **app\_storefront\_core/cartridge/templates/default/checkout/summary/summary.isml**

Starting on line 221, comment out the following button:

|  |
| --- |
| <**button** class="button-fancy-large" type="submit" name="submit" value="${Resource.msg('global.submitorder','locale',null)}">  ${Resource.msg('global.submitorder','locale',null)} </**button**> |

Right underneath, paste the following code. This snippet contains the commented out button as well for clarity.

|  |
| --- |
| <isafterpaywidget /> <isif condition="${afterpayExpressCheckoutEnabled && isExpressCheckoutFinalize}"> <script type="text/javascript"> document.addEventListener("DOMContentLoaded", function() {  createAfterpayWidget();  let elem = document.getElementById('afterpay-express-placeorder-button');  elem.addEventListener('click', function() {  let finalizeUrl = document.getElementById('afterpay-express-url-placeorder').value;  window.location.href=finalizeUrl+'?checksum='+afterpayWidget.paymentScheduleChecksum;  }); }); </script> <isscript> var brandUtilities = require("\*/cartridge/scripts/util/afterpayUtilities.js").brandUtilities; var countrycode = brandUtilities.getCountryCode(); var placeOrderButtonClass = 'afterpay-placeorder-button'; if(countrycode === 'GB'){  placeOrderButtonClass = 'clearpay-placeorder-button'; } </isscript> <isif condition="${afterpayExpressCheckoutEnabled && isExpressCheckoutFinalize && afterpayAmount.value > 0}"> <div id="afterpay-widget-container"></div> </isif> <div id="afterpay-placeorder" class="afterpay-widget-hideuntilready afterpay-summary-button-container"> <input type="button" id="afterpay-express-placeorder-button" class="${placeOrderButtonClass} afterpay-summary">  &nbsp; </input> </div> <iselse/>  <button class="button-fancy-large" type="submit" name="submit" value="${Resource.msg('global.submitorder','locale',null)}"> ${Resource.msg('global.submitorder','locale',null)}  </button> </isif> <iscomment>End of Afterpay</iscomment> |

#### **app\_storefront\_core/cartridge/scripts/util/Resource.ds**

Find the line ResourceHelper.getPreferences = function(pageContext) {

Add the highlighted code inside the method:

|  |
| --- |
| ResourceHelper.getPreferences = function(pageContext) {  var cookieHintAsset = ContentMgr.getContent('cookie\_hint');  var consentTrackingHintAsset = ContentMgr.getContent('consent\_tracking\_hint');  var AfterpayUtilities = require('\*/cartridge/scripts/util/afterpayUtilities.js');  var brandUtilities = AfterpayUtilities.brandUtilities;  return {  LISTING\_INFINITE\_SCROLL: (Site.getCurrent().getCustomPreferenceValue('enableInfiniteScroll') ? true : false),  LISTING\_REFINE\_SORT: true,  STORE\_PICKUP: Site.getCurrent().getCustomPreferenceValue('enableStorePickUp'),  COOKIE\_HINT: (cookieHintAsset && cookieHintAsset.online) || false,  CONSENT\_TRACKING\_HINT: (consentTrackingHintAsset && consentTrackingHintAsset.online) || false,  CHECK\_TLS: Site.getCurrent().getCustomPreferenceValue('checkTLS'),  AFTERPAY\_PORTAL: brandUtilities.getBrandSettings().javaScriptUrl  }; } |

#### File: **app\_storefront\_core/cartridge/templates/default/product/productcontent.isml**

Starting on line 222 it looks like SiteGenesis has a slight bug, which we will fix. The old line was:

|  |
| --- |
| <**input** type="hidden" name="cgid" id="cgid" value="{pdict.CurrentHttpParameterMap.cgid.value}" /> |

Let’s go ahead and add the $ in the value attribute:

|  |
| --- |
| <**input** type="hidden" name="cgid" id="cgid" value="${pdict.CurrentHttpParameterMap.cgid.value}" /> |

Right below this change, paste the code below:

|  |
| --- |
| <**iscomment**>Start of Afterpay</**iscomment**> <**isafterpayexpresscheckout** pagetype="product\_detail"/> <**isif** condition="${afterpayExpressCheckoutPdpEnabled}"> <**script** type="text/javascript">  **var** continueFinalize = **function**() {  **var** finurl = $('#afterpayurl-continuefinalize').val() + "?cartAction=add&pid=" + $('#pid').val() + "&Quantity=" + $('#Quantity').val();  window.location.href = finurl;  }  $(document).ajaxComplete(**function**() {  **if** (document.querySelector("#afterpay-express-link-button")) {  initAfterpay({pickupflag: false, target: "#afterpay-express-link-button", productIdSelector: "#pid", productQuantitySelector: "#Quantity"});  }  });  $(document).ready(**function**() {  **if** (document.querySelector("#afterpay-express-link-button")) {  initAfterpay({pickupflag: false, target: "#afterpay-express-link-button", productIdSelector: "#pid", productQuantitySelector: "#Quantity"});  }  }); </**script**> </**isif**> <**iscomment**>End of Afterpay</**iscomment**> |

Right after line 245 of the change above, paste the following:

|  |
| --- |
| <iscomment>Start of Afterpay</iscomment> <isif condition="${afterpayExpressCheckoutEnabled && afterpayExpressCheckoutPdpEnabled && !disableAfterpayPaymentMethod}"> <iscomment> When calling initAfterpay, we pass in the selectors for the pid and Quantity input fields for the item. When Afterpay Express popup starts, it will grab the values using those selectors, and pass those in as input to CreateToken. The AfterpayExpress-CreateToken controller will add the product into the cart by itself. </iscomment> <isscript>  var brandUtilities = require("\*/cartridge/scripts/util/afterpayUtilities.js").brandUtilities;  var countrycode = brandUtilities.getCountryCode();  var payNowButtonClass = 'afterpay-paynow-button';  if(countrycode === 'GB'){  payNowButtonClass = 'clearpay-paynow-button';  } </isscript> <div class="afterpay-paynow-pdp"> <isif condition="${isExpressCheckoutFinalize}"> <input id="afterpay-express-continuefinalize-button" type="button" class="${payNowButtonClass}" onclick="continueFinalize();return false;"></input> <iselse/> <input id="afterpay-express-link-button" type="button" class="${payNowButtonClass}"></input> </isif> </div> </isif> <iscomment>End of Afterpay</iscomment> |

#### File: **app\_storefront\_core/cartridge/js/pages/checkout/billing.js**

In the $addCoupon.on() event handler, change the line:

|  |
| --- |
| **if** (data.success && data.baskettotal === 0) { |

To:

|  |
| --- |
| **if** (data.success) { |

Optionally, paste the following comments to indicate the reason for the change.

|  |
| --- |
| // Afterpay - fix for apparent bug since page should reload if coupon applied regardless of baskettotal. Otherwise // order totals on page are not updated and no indication is given whether coupon has been applied |

*As the comment indicates, SiteGenesis has a small bug on the billing page. If a coupon is applied, the order totals are not updated and there’s no indication on the page whether the coupon was applied or not. The data is only updated if the page is refreshed.*

#### File: **app\_storefront\_core/cartridge/js/app.js**

Near the bottom of the initializeEvents() function, paste the following:

|  |
| --- |
| // Start of Afterpay $(document).on('click', '.afterpay-link', **function** (e) {  e.preventDefault();  dialog.open({  url: $(e.target).attr('href')  });  }); // End of Afterpay |

#### File: **app\_storefront\_core/cartridge/js/countries.js**

|  |
| --- |
| 'use strict';  exports.init = function init () {  $('.country-selector .current-country').on('click', function () {  $('.country-selector .selector').toggleClass('active');  $(this).toggleClass('selector-active');  });  // set currency first before reload  $('.country-selector .selector .locale').on('click', function (e) {  e.preventDefault();  var url = this.href;  var currency = this.getAttribute('data-currency');  var locale = this.getAttribute('data-locale');  $.ajax({  dataType: 'json',  url: Urls.setSessionCurrency,  data: {  format: 'ajax',  currencyMnemonic: currency,  locale: locale  }  })  .done(function (response) {  if (!response.success) {  throw new Error('Unable to set currency');  }  window.location.href = url;  });  }); }; |

#### File: **app\_storefront\_core/cartridge/js/bonus-products-view.js**

Near the top of the file, paste the following:

|  |
| --- |
| function createOrder() {  var url = util.ajaxUrl(Urls.createAfterpayOrder);   ajax.getJson({  url: url,  callback: function (data) {  if (data) {  checkAuthorize(data);  }  }  }); }  function checkAuthorize(data) {  if (window.AfterPay) {  AfterPay.init();  AfterPay.display({token: data.token});  } else {  setTimeout(checkAuthorize.bind(null, data), 3000);  } }  function initScript() {  var script = document.createElement('script');  script.src = SitePreferences.AFTERPAY\_PORTAL;  script.async = true;  document.body.appendChild(script); }  function initSubmitOrderEvent() {  var $form = $('.checkout-billing');   var $paymentMethods = $form.find('input[name$="\_selectedPaymentMethodID"]');  var $btnContinue = $form.find('button[name$="\_billing\_save"]');   $btnContinue.on('click', function(e) {  var methods = ['AFTERPAY', 'CLEARPAY'];  if (methods.indexOf($paymentMethods.filter(':checked').val()) !== -1) {  e.preventDefault();  createOrder();  return false;  }  });  var $declined = $('.ap-declined'),  $form = $('.submit-order');   if ($declined.length > 0) {  $form.on('submit', function (e) {  e.preventDefault();  window.location = window.Urls.billing;  });  } else {  $form.off('submit').on('submit', function (e) {  return true;  });  } }   exports.init = function () {  initScript();  initSubmitOrderEvent(); };  module.exports.initSubmitOrderEvent = function () {  initSubmitOrderEvent(); }; |

#### File: **app\_storefront\_core/cartridge/scss/default/style.scss**

At the end of the file, paste the following as shown in the screenshot:

|  |
| --- |
| // Start of Afterpay @**import** "afterpay"; // End of Afterpay |

## Building Client Resources

After making changes to the base SiteGenesis files, you will need to rebuild client-side JS and CSS. This is typically done by running this from the top-level SiteGenesis directory.

|  |
| --- |
| $ npm run build |

## Upload Modified Cartridge

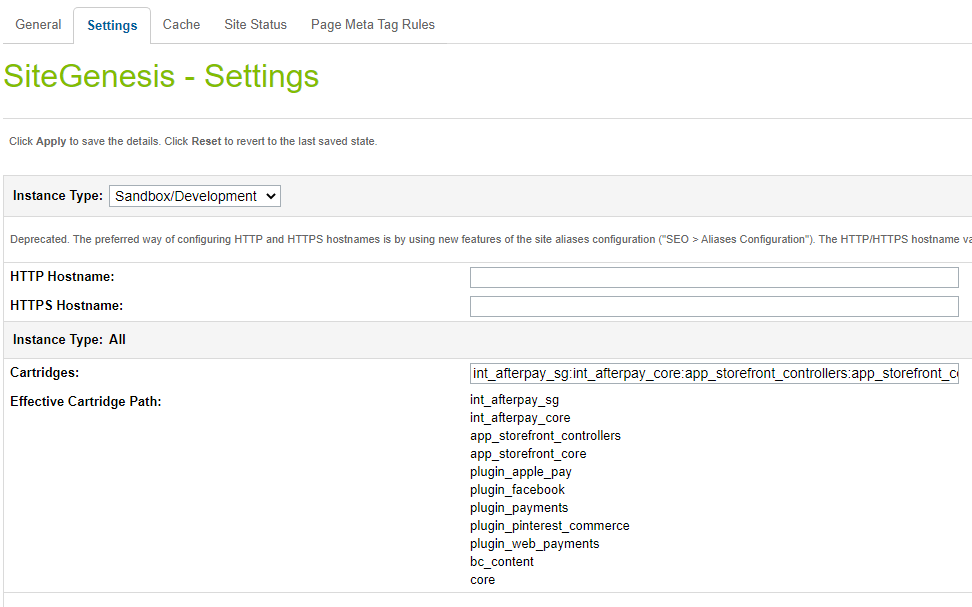
Use UX Studio (or VSCode with the Prophet extension) to upload the modified cartridge to your store.

## Activate the Cartridge in Business Manager

### Set Cartridge Path

Before the Afterpay functionality can become available to SFRA, the cartridge names have to be added to the cartridge path of the Site’s settings. In order to do this, follow the below instructions:

1. Log into Business Manager
2. Navigate to Administration > Sites > Manage Sites.
3. Click on the target site name and Click on the Settings tab.
4. In the textbox Cartridges add **“int\_afterpay\_sg:int\_afterpay\_core”** in front of Sitegenesis base cartridges.



1. Click Apply.
2. To activate the cartridge for the Sandbox/Development/Production instances repeat steps 4 and 5 after selecting the appropriate instance from the Instance Type dropdown menu.
3. Repeat steps 3 to 6 for each site that is to use Afterpay.

### 

### Set Afterpay Custom Site Preferences

After installing the cartridge, the following new configuration options will be available in the **Business Manager**.

In the Business Manager, go to **Merchant Tools > Site Preferences > Custom Preferences > Integration\_Afterpay**

If you currently have v21.0.0 cartridge, you will already have the express checkout options configured.

|  |  |  |
| --- | --- | --- |
| Site Preference | Description | Default |
| Brand settings (apBrandSettings) | Afterpay / Clearpay Brand related settings  in JSON format where   * brand – afterpay/clearpay, depending on region; * service – the field is used to identify the service with Afterpay/Clearpay. * javaScriptUrl – The endpoint where the javascript library is obtained from.   Sandbox:  Afterpay (+UK) <https://portal.sandbox.afterpay.com/afterpay.js>  Clearpay <https://merchant-tools.sandbox.clearpay.com/clearpay.js>  Production:  Afterpay (+UK) <https://portal.afterpay.com/afterpay.js>  Clearpay <https://merchant-tools.clearpay.com/clearpay.js>   * learnMoreUrl – the URL that links to more information about Afterpay that users can click to view. | <default value for RefArch site. Other countries examples could be found in RefArchGlobal site meta>  {  "US": {  "brand": "afterpay",  "service": "afterpay.service.US",  "javaScriptUrl": "https://portal.sandbox.afterpay.com/afterpay.js",  "learnMoreUrl": "https://www.afterpay.com/purchase-payment-agreement"  }  } |
| Enable Express Checkout  (apEnableExpressCheckout) | Enables the Express Checkout feature. | Yes |
| Enable BuyNow  (apEnableExpressCheckoutBuyNow) | Enables the BuyNow mode. Skips review page on merchant checkout. (Note: Only used for Integrated Shipping) | Yes |
| Express Checkout Shipping Strategy  (apExpressCheckoutShippingStrategy) | Select either Integrated Shipping (Afterpay Checkout prompts for shipping method), or deferred (merchant site handles shipping method selection) | Integrated |
| Description Express Checkout will use for In-Store Pickup  (apStorePickupDescription) | prompt to use in Afterpay Express Checkout for in-store pickup orders | Available for next-day pickup |
| Enable Express Checkout on Product Details Page  (apEnableExpressCheckoutPdp) | enable Express Checkout directly from the product detail page | No |

### 

### When Afterpay is turned on, by default Express Checkout will be enabled, Integrated Shipping will be enabled, and the BuyNow option will be enabled. Make sure to change these settings to the ones you wish to use on your site.

### When Express Checkout is disabled, the Afterpay checkout button will only appear on the checkout page.

### Set Afterpay Payment Methods

Merchant Tools > Ordering > Import & Export > Upload the “payment-methods.xml” from the metadata folder > Import the xml. A payment method with the names AFTERPAY and CLEARPAY are available.

### Set Afterpay Payment Processors

Merchant Tools > Ordering > Payment Processors > Click New Give a unique ID for the processor: “*AFTERPAY\_CREDIT*” Description: **“***Pay By Installment” and click Apply*

### Set SFCC(Demandware) Service

Administration > Operations > Services > Credentials > afterpay.http.defaultcredentials > Details > Credentials tab and select required service and enter the account details.

|  |  |
| --- | --- |
| URL | **Australia & New Zealand:**  Sandbox  https://api-sandbox.afterpay.com/v2/  Production  https://api.afterpay.com/v2/  **United States & Canada:**  Sandbox  https://api.us-sandbox.afterpay.com/v2/  Production  https://api.us.afterpay.com/v2/  **United Kingdom:**  Sandbox  https://api.eu-sandbox.afterpay.com/v2/  Production  https://api.eu.afterpay.com/v2/  **EU:**  Sandbox  https://api.sandbox.clearpay.com/v2/  Production  https://api.clearpay.com/v2/ |
| User | Enter the Merchant ID provided by Afterpay |
| Password | Enter the Secret Key provided by Afterpay |

### Setting up Job Schedule

This cartridge supports an Afterpay order status of PENDING by running a Job Schedule against all Afterpay Orders that are in a PENDING state. This is not required at this point as Aferpay will only respond with APPROVED or DECLINED and if the order is DECLINED the checkout cannot complete.

By default, Job Step is set for SiteGenesis site. You may change this to your site.

# Testing

## Checkout Flow

Once the cartridge is installed and integrated based on the instructions, please try to place an order on your sandbox to test the functionality. Even if you’ve enabled Express Checkout, place a non-Express-Checkout order by putting items in the cart and proceeding to the checkout page before placing an order using Afterpay.

* After you click on Place order then you should be redirected to Afterpay Payment Page. If you are not registered, then you will be asked to create an account



## Express Checkout Flow

If you’ve enabled Express Checkout, please try to place an order on your sandbox to test the functionality. Use the Express Checkout buttons on the cart page and verify that Afterpay’s Express Checkout popup appears and proceed through the checkout.

# Features and Usage

## Afterpay Payment on Checkout

When the shopper selects the Afterpay payment method at checkout and clicks the **Place Order** button, they will be redirected to Afterpay servers to handle the payment. After the payment is complete, the shopper is redirected back to the confirmation or order review page on your store

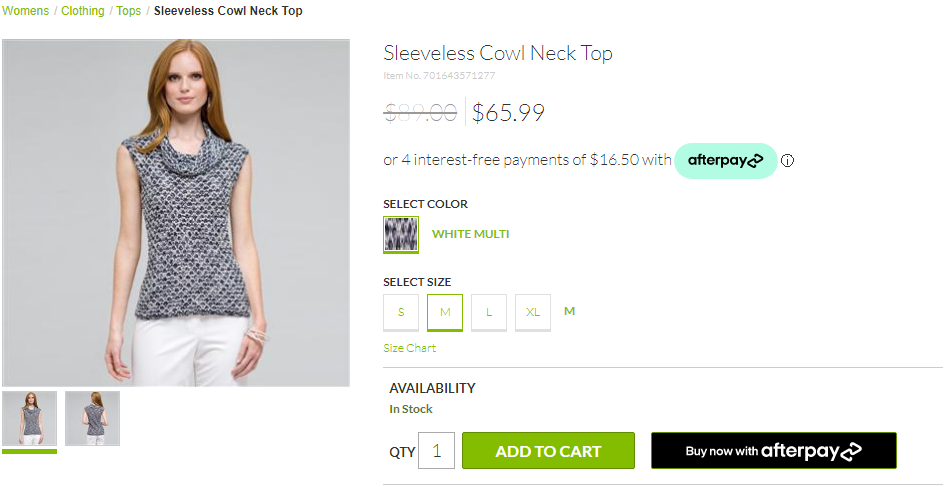
## 

## 

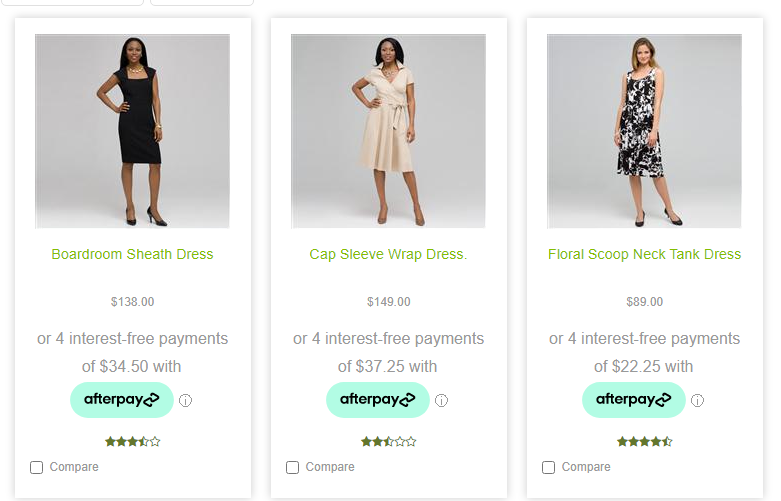
## Afterpay Product Messaging

Afterpay adds messaging on various pages in the store.

Messaging on Product Details Page:



Messaging on Product List Page:

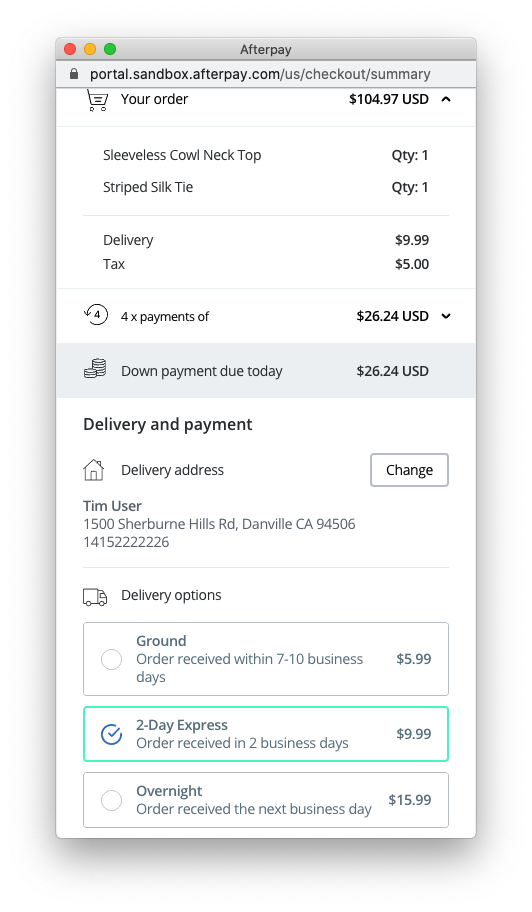
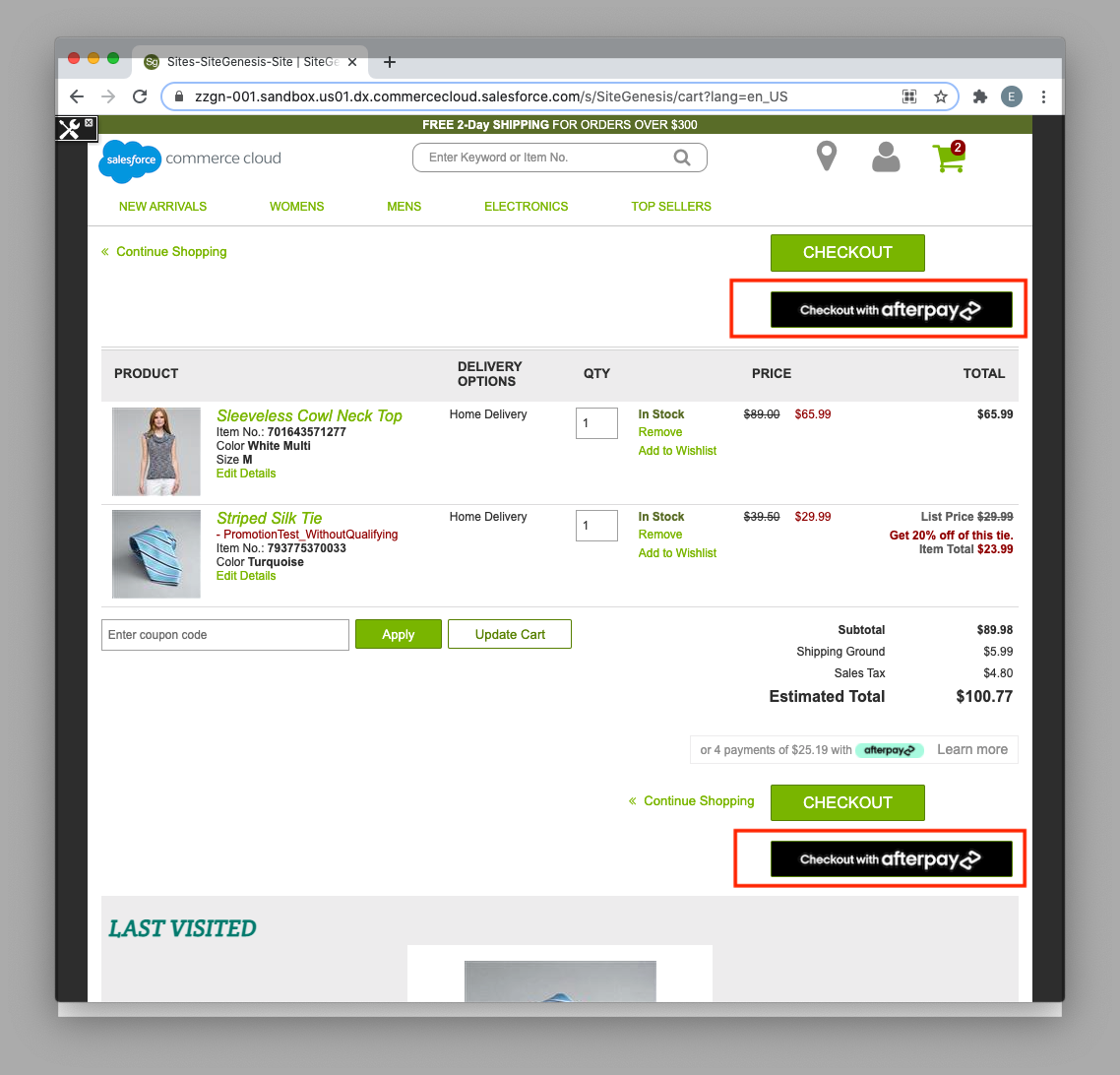


## 

## Express Checkout

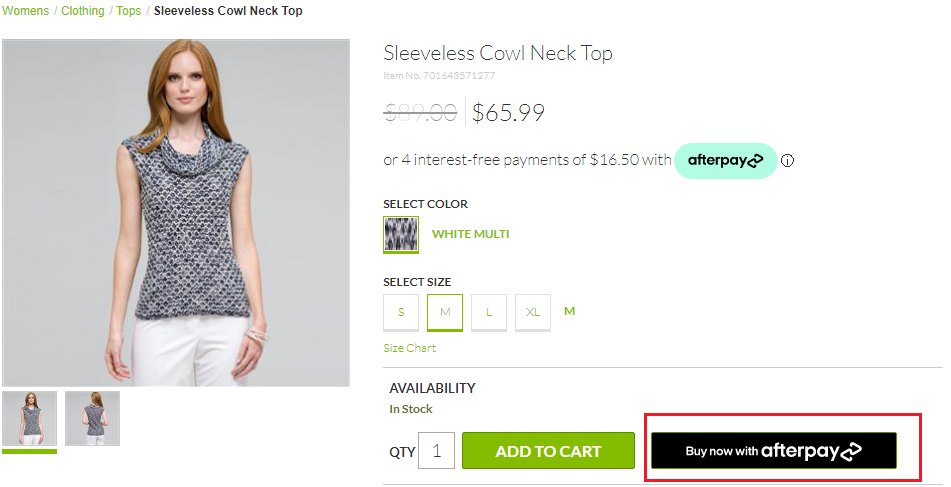
### Checkout from Cart Page

When the Express Checkout feature is enabled, new buttons will appear on the cart page.

  
  
When the shopper presses either of these buttons, it will launch Afterpay Express Checkout. A popup window will appear and depending on whether the shopper has preset default checkout settings with Afterpay, they will be prompted for payment information and a shipping address. Assuming the **integrated shipping** option has been selected by the merchant, Afterpay will call a webservice, in the background, on the merchant store to get available shipping options and pricing, and display that to the shopper in the same popup window. If the BuyNow option has been selected by the merchant, the shopper can simply click “Buy Now” and will be redirected back to the merchant site, where an order will be immediately created and payment auth or capture performed (see [API Docs for Payment Flows](https://developers.afterpay.com/afterpay-online/docs/api-calls-and-payment-flows)).

### Checkout From Product Details Page

When the apEnableExpressCheckoutPdp setting is enabled, the product details page will display a “Buy now with Afterpay” button, whenever the item is eligible to be added to the cart and when the item is below the maximum threshold amount for Afterpay (typically $2000 USD).



When this button is clicked, the current item and quantity will immediately be added to the cart, and Afterpay Express Checkout will start with all the cart contents.

### Integrated Shipping

This feature improves the user experience by integrating your shipping options directly into the Afterpay Express checkout. It streamlines the checkout process and can be combined with the BuyNow option to create a one-stop checkout.  
  
To enable Integrated Shipping, in **Business Manager**, make sure to set the custom preference **apExpressCheckoutShippingStrategy** to **integrated**.

During Afterpay Express Checkout, once the shopper’s delivery address is known, and whenever the shopper changes their delivery address, the Afterpay popup will send a request to the merchant server to supply valid shipping options and costs for that address to the popup. The shopper makes the selection in the Afterpay popup, and the selected shipping option is passed back to the merchant site for order creation. This avoids the need for having the shopper make the shipping method selection on the merchant site.



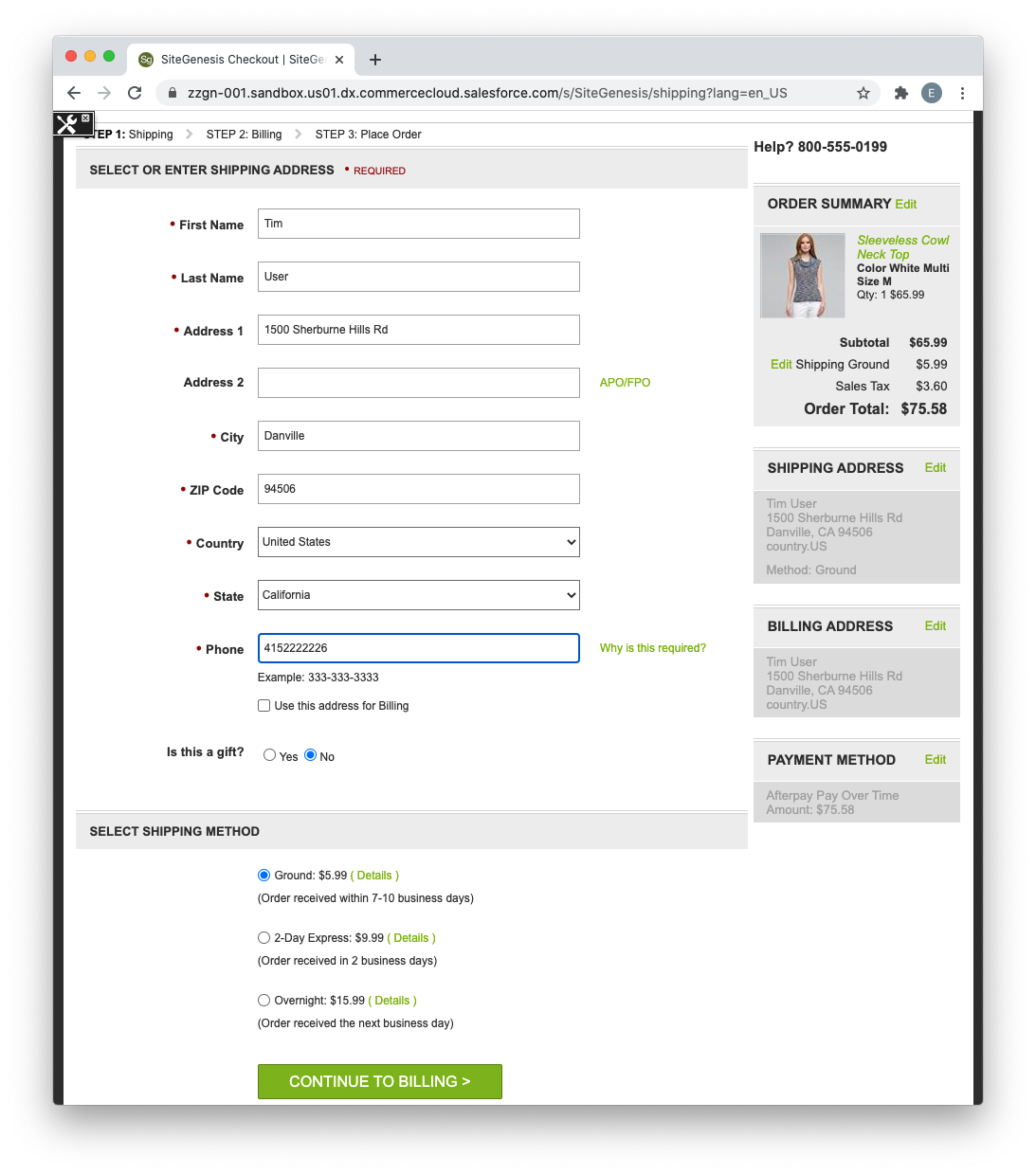
### In-Store Pickup and Integrated Shipping

In-Store pickup is supported out of the box by **Integrated Shipping**. All items in the cart must be picked up from the same store, and there cannot be a mix of in-store pickup items and home delivery items in the cart. When those conditions are met, when the Afterpay checkout button is pressed (**from the cart page only**), the store information is passed to Afterpay in the **Create Token** request and the shopper will be presented with an in-store pickup interface.  
  


If the Afterpay checkout button is pressed from the product details page, the checkout will assume the order is for home delivery.

### Deferred Shipping

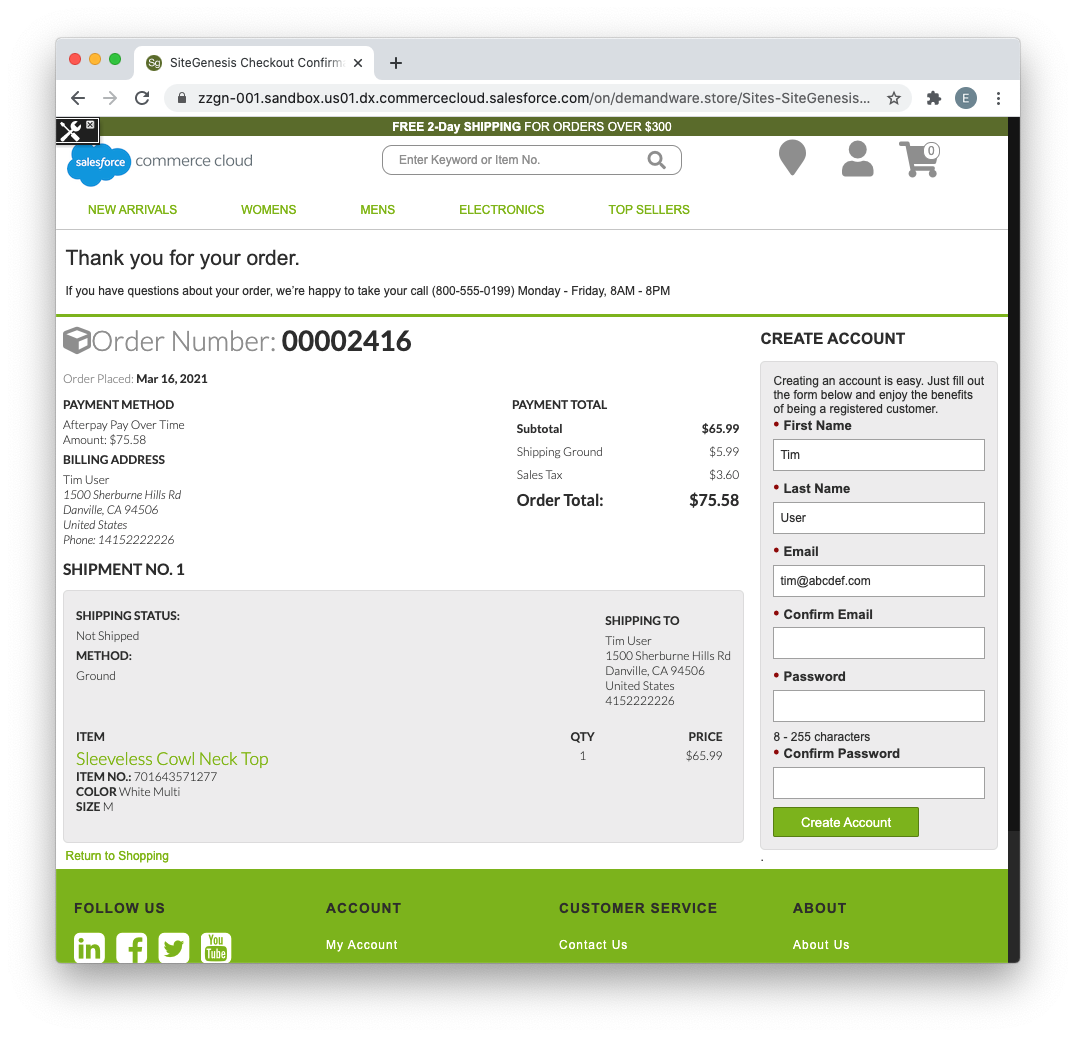
In order to handle shipping method selection on your site instead of via Afterpay’s checkout popup window, please enable the **Deferred Shipping** feature. This allows you to use the Afterpay Express Checkout to simplify the shopper’s checkout process, and quickly get the shopper’s name, email and shipping address and skip having to enter them in your checkout forms. After the shopper has gone through the Afterpay checkout process, they will be redirected back to the shipping page on your site to proceed with checkout.



To enable Integrated Shipping, in Business Manager, make sure to set the custom preference **apExpressCheckoutShippingStrategy** to **integrated**.

### BuyNow (Immediate capture) Feature

When the Express Checkout is complete, you may either capture payment immediately or continue the checkout on your site. If you are capturing immediately, set the **apEnableExpressCheckoutBuyNow** custom preference setting to **Yes** - this shows the user a “Buy Now” button at the end of their Afterpay checkout. Upon clicking, control is returned to your site for payment capture, and an order summary will be displayed.



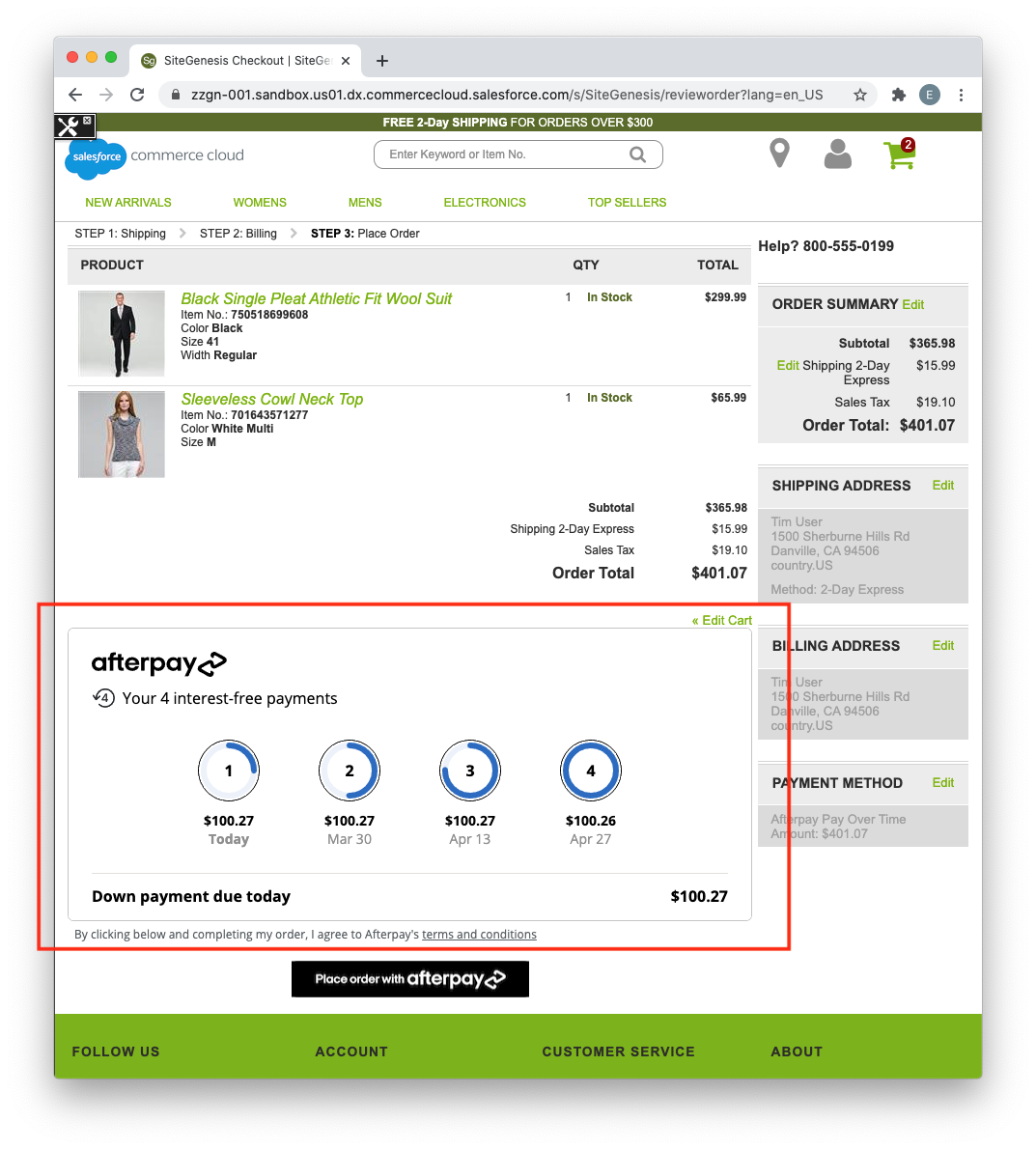
**Note:** The BuyNow feature only applies to the Integrated Shipping option. For Deferred Shipping control needs to return to your site for shipping method selection, so a BuyNow feature would not make sense.

### Afterpay Widget

The Afterpay Widget is not a new feature in and of itself but is an essential component that needs to be placed on your site if the order value is going to change after the Afterpay Express checkout has finished. This will be the case unless the **Integrated Shipping** and **BuyNow** features are enabled.  
  
It serves several purposes:

1. To ensure the user has seen and agreed to an accurate payment schedule.
2. To verify that the final amount captured matches what the user has seen.
3. To provide information to the merchant such as the first payment amount for display purposes.
4. To inform the merchant of any barriers to purchase, for instance the user having gone over their payment limit with Afterpay.

The Afterpay Widget will be displayed on the final checkout summary screen (for deferred shipping):



It will also be displayed on the billing screen if the shopper has already been through the Afterpay checkout process and are now back on your site to complete the checkout (see Express Checkout Finalization Flow below). It is only displayed when Afterpay is the active payment instrument for the cart.

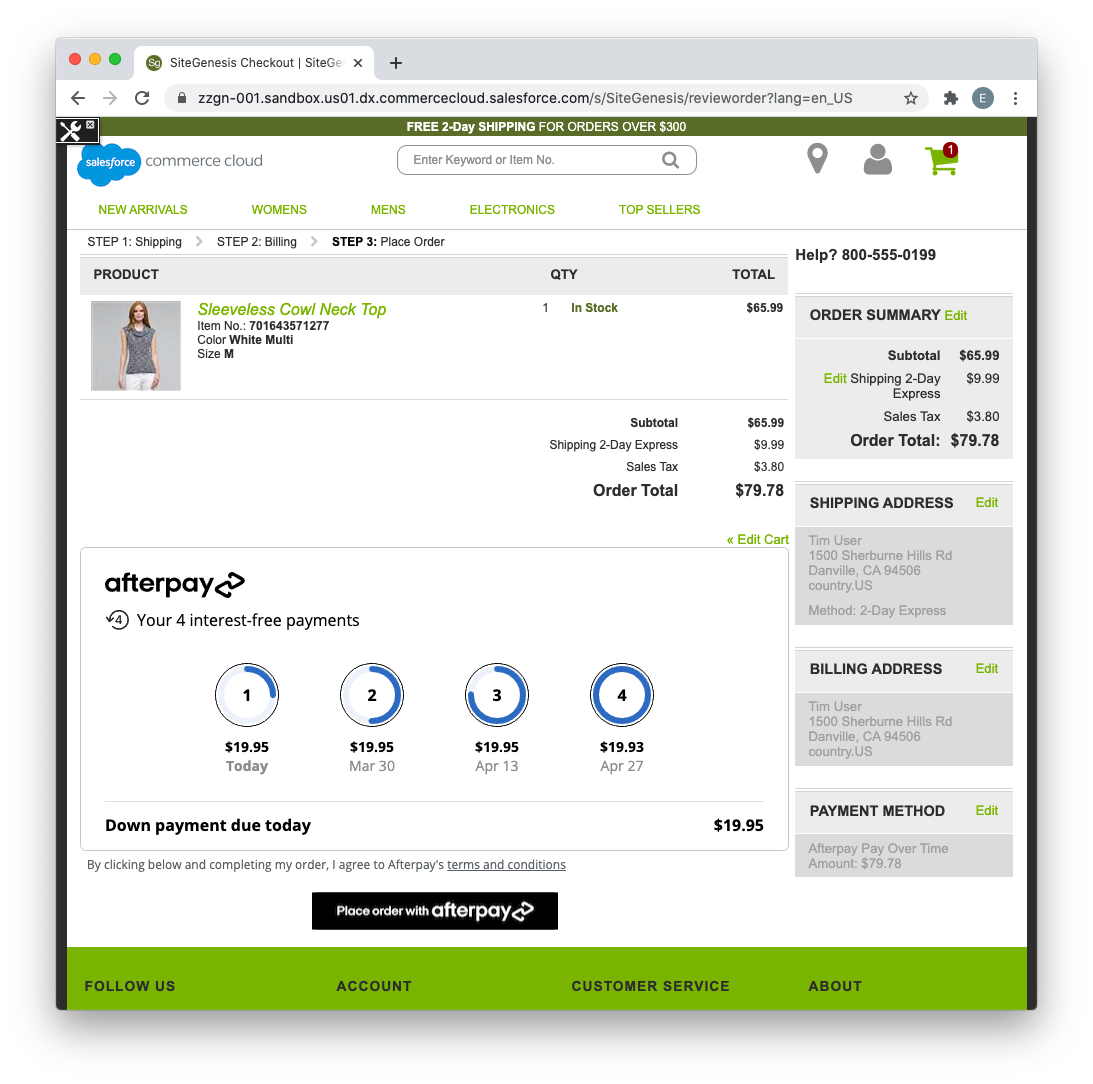


The **Place order with Afterpay** button will always be placed below the Afterpay Widget to insure that the shopper has seen the widget before placing the order.

### Express Checkout Finalization Flow (ECFF)

Unless you are using the **Integrated Shipping** feature with the **BuyNow** option, the shopper will be redirected back to your site after the Afterpay checkout. During this phase, in SiteGenesis, the shopper can still make arbitrary changes to their cart (e.g. add/remove items, apply coupons and gift certificates, change shipping address and methods, etc) . However, Afterpay’s link cartridge will remember (using session state) that the user is currently in the Express Checkout Finalization Flow. This will affect the behavior of the Afterpay checkout buttons:

* When the “Checkout with Afterpay” button is clicked, instead of redirecting to Afterpay to checkout again and generate a new token, the button will take the shopper to the cart summary page, where the Afterpay Widget will be displayed, along with the “Place Order with Afterpay” button.



* When the “Buy now with Afterpay” button is clicked on the product details page, the product will be added to the cart and the shopper will be directed to the summary page as well. The default shipping method used in the rest of the order will be used for the newly added product as well. The Afterpay Widget will display the total order amount, and upon clicking the “Place Order with Afterpay” button, the order will be created and payment will be captured.

**Exiting from Express Checkout Finalization Flow:**

There are several conditions which will cause the shopper to exit from ECFF:

* If the user explicitly chooses another payment method on the billing page, and clicks submit, Afterpay will be removed as a payment method.
* If the user navigates to any page which normally contains an Afterpay Widget (the billing page and summary page) but does not have Afterpay set as a payment instrument for the cart, they will no longer be in ECFF.
* After the shopper clicks **Place Order with Afterpay**, regardless of whether the transaction was successful or not, they will no longer be in ECFF. The **Place Order with Afterpay** button will also fail if the user is not in “Express Checkout Finalization Flow”, but this should not happen unless the shopper opens multiple browser windows and clicks on the button in an old window when the session state has already changed.
* The user’s browser session expired

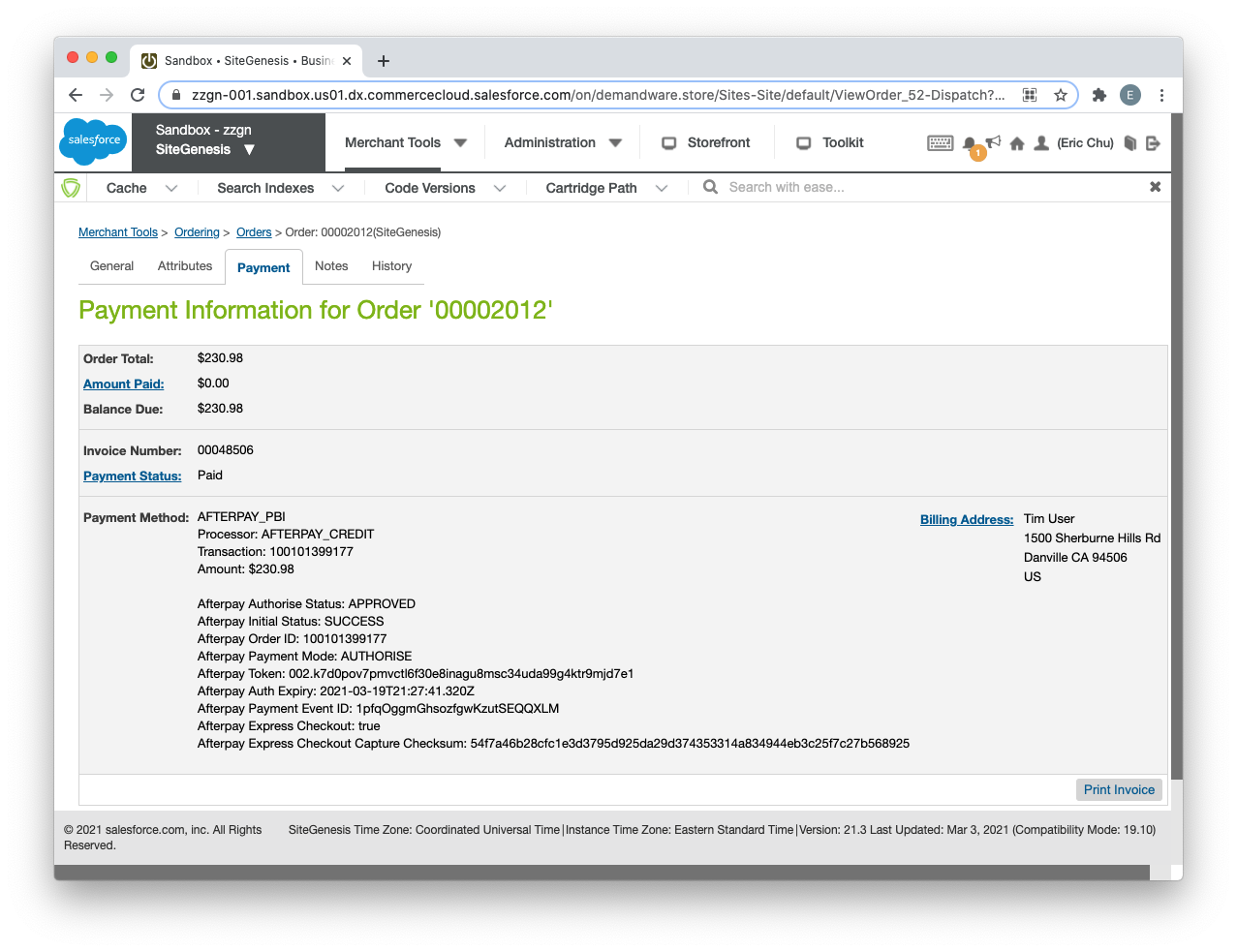
After exiting from the Express Checkout FInalization Flow, clicking on “Checkout with Afterpay”, or “Buy Now with Afterpay” buttons will start the Afterpay checkout process again.

### 

### Express Checkout Behavior

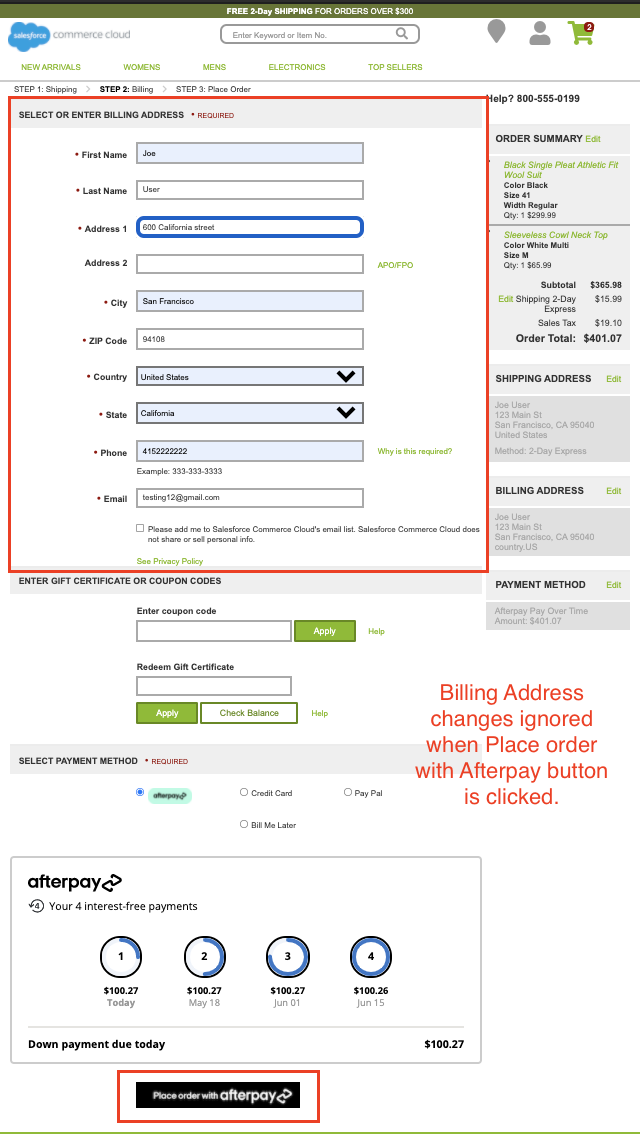
### Order Fields

Orders created via Afterpay Express Checkout will look the same as non-Express Checkout orders, with the following exceptions:

* The Billing Address field will display the shipping address. Since Afterpay is handling the checkout, after the express checkout popup closes, the shipping address is returned from Afterpay and used to populate both the shipping address and billing address in the cart. Thus, when the order is created, it will be the shipping address unless the shopper has specifically modified the billing address before finalizing the transaction.
* **Afterpay Express Checkout** - will be true for express checkout orders
* **Afterpay Express Checkout Capture Checksum** - this is a checksum generated by the Afterpay Widget, and indicates the total amount and payment schedule the shopper was shown before placing the order  
  

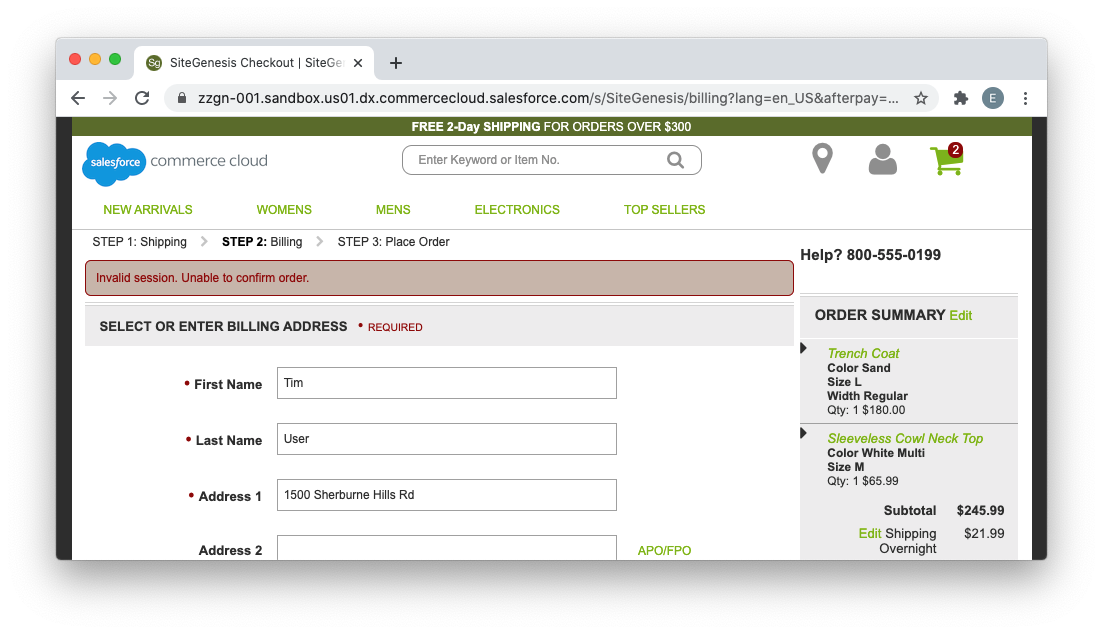
### Billing Page Address Changes During ECFF

On the Billing Page, if the billing address is changed right before the shopper clicks “Place order with Afterpay” (during ECFF), the changed data will be ignored since the form is not actually submitted to the server. This should be unimportant since Afterpay is handling the payment process and the billing address fields would typically be prepopulated with the shipping address.



### Error Behavior

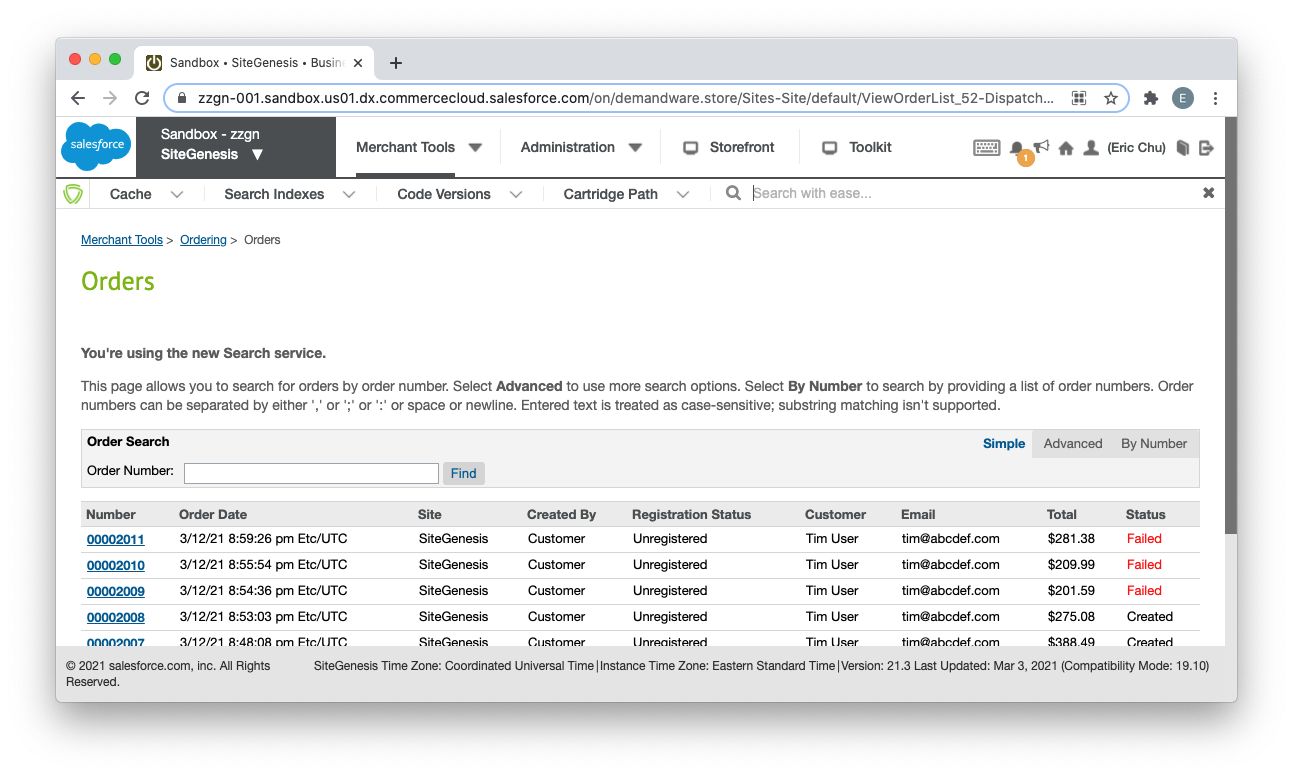
When there are payment-related errors, the shopper will be redirected back to the billing page and an error message will be displayed.

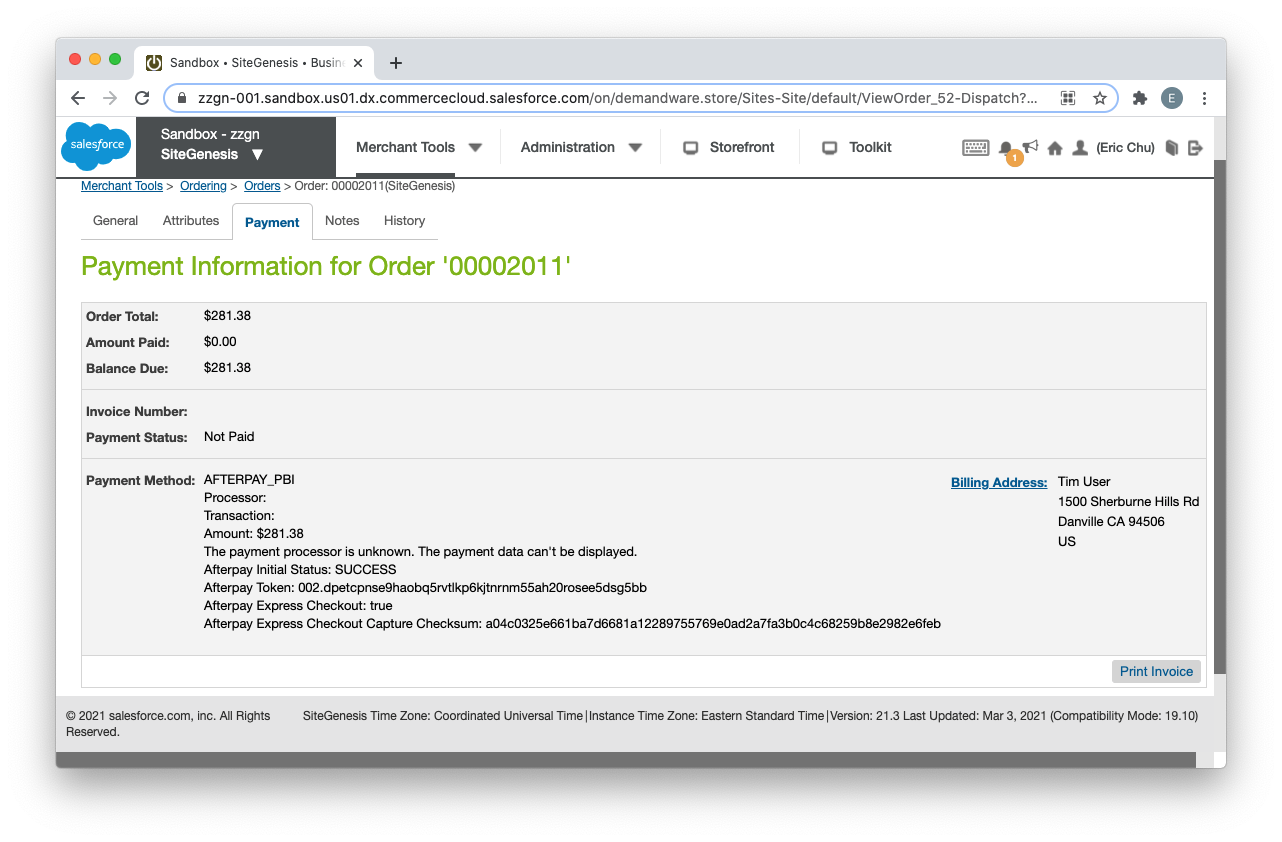


When there is a payment-related error, an order will generally already have been created by SFCC. This can occur in the following scenarios:

* Capture was declined
* Shopper modified the cart in the short period between when SFCC created the order, and before the payment capture via Afterpay.
* The merchant site was unable to communicate with Afterpay’s payment servers

The order will have a “Failed” status in these cases, and the payment status will be “Not Paid”.



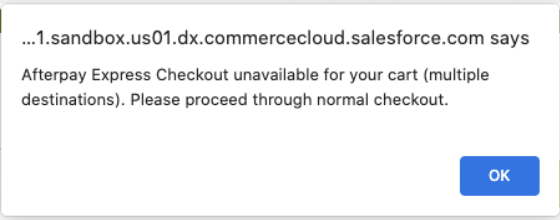
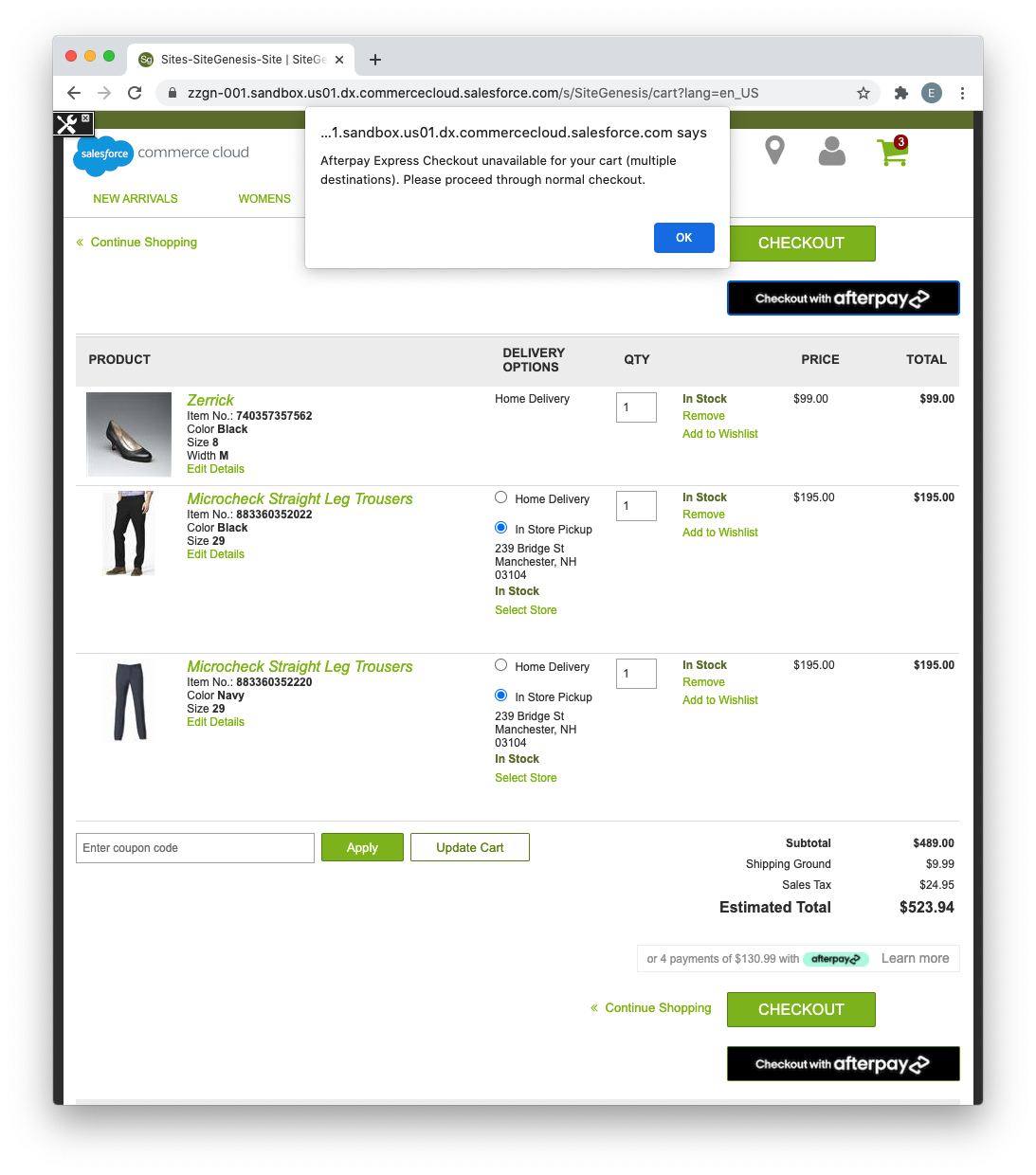
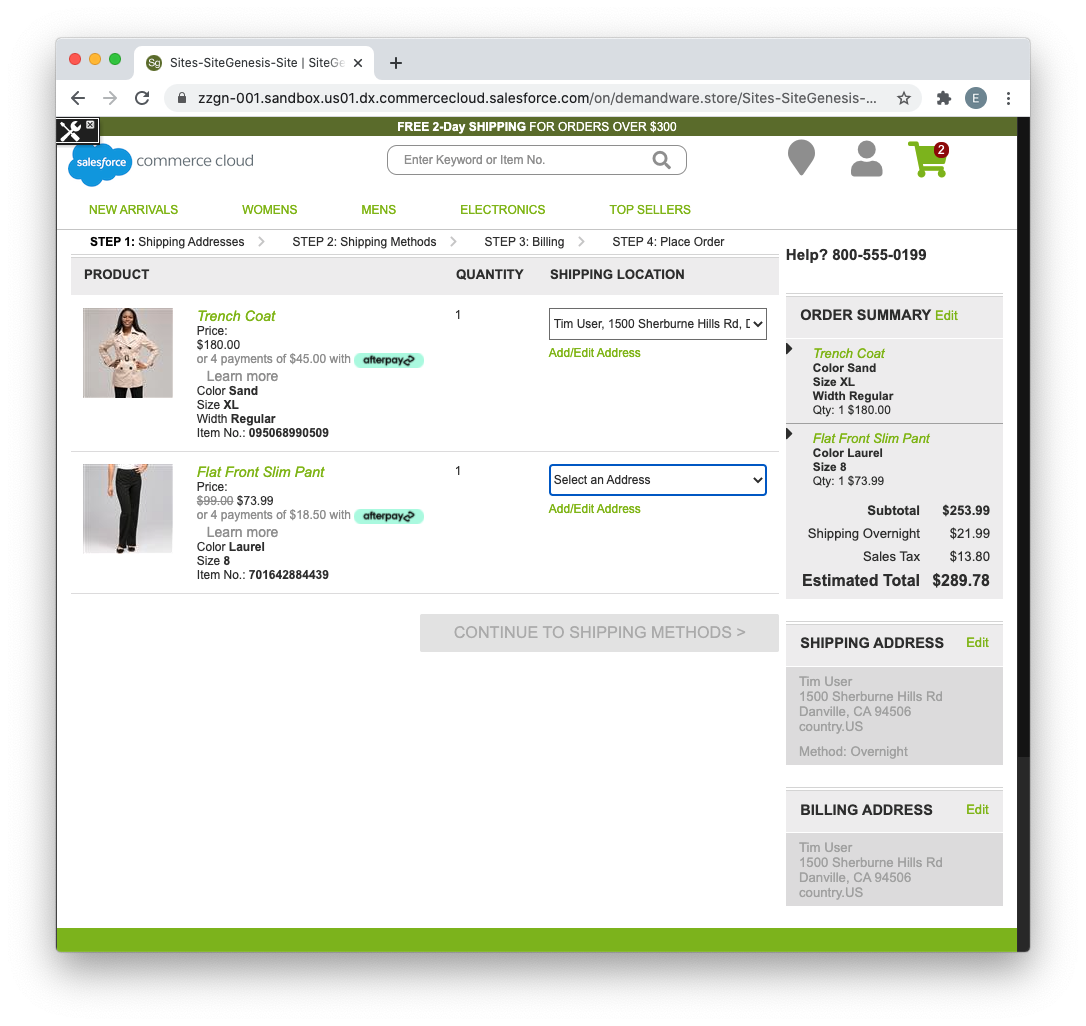


### ECFF Behavior

Because the shopper can make arbitrary changes to their billing address and shipping address during ECFF, occasionally the shipping and billing address on-file at Afterpay may not match the information created in the order. However, this should not matter since you (the merchant), will always have the correct shipping address, and Afterpay will always have the correct billing address.

### Express Checkout Limitations

### No Multiple Shipping Destinations Prior to Afterpay Express Checkout

Express Checkout will only work when all items in the cart are set for **Home Delivery** OR all items in the cart are set for **In-Store Pickup** from the same store. If neither of these are true, and the shopper clicks the “Checkout with Afterpay” button, an error message will appear indicating that Afterpay is unavailable for the current cart.  
  
Afterpay checkout buttons will not appear on SiteGenesis multiship pages:  


The shopper can still pay with Afterpay using the standard checkout process (selecting Afterpay on the billing page), but will not be able to use express checkout features.

Even though the shopper will not be able to use Express Checkout in these instances, the exception is if they’ve already gone through the Afterpay Express Checkout, and are now in the ECFF on your site. They will be able to add items to cart, ship to multiple locations, and/or pick up from multiple stores. The order can still be created and payment captured once they click on the “Pay now with Afterpay” button. At this point, since Afterpay is only capturing payment, the shipping limitations no longer matter.

### Billing Address is not Passed to Merchant

If a shopper uses Afterpay Express Checkout, the billing address is typically not passed back to your site from Afterpay. In non-BuyNow checkout flows, the billing address will instead be populated with the shipping address.

### Other Limitations

* Currently, there is no support for a “Buy Now with Afterpay” from a wishlist page.
* Currently, there is no support for a “Buy Now with Afterpay” button on a product bundle page.
* For in-store pickup orders, Afterpay Express checkout does not support adding a message to the store. If finalization happens at the merchant site, the shopper could go back to the shipping screen and add the message.

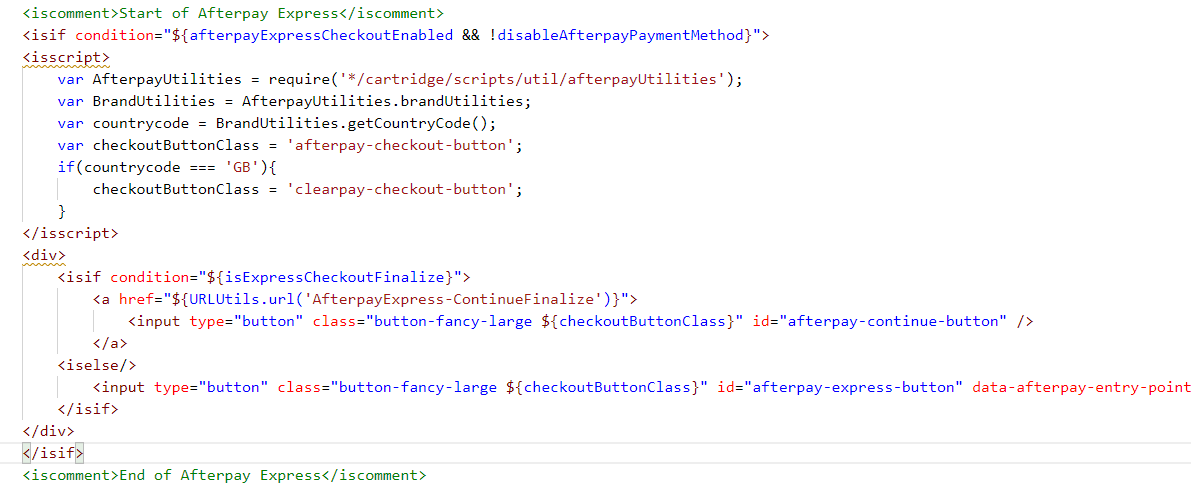
### Express Checkout Implementation Details

### Overview

Particularly for sites that have made substantive changes to the base SiteGenesis store, we’ve provided some details on how certain features were implemented. This would make it easier to understand how your site’s customizations may potentially require some modifications to Afterpay’s cartridge to work properly.

### Checkout Button Behavior during ECFF

In the isml templates, buttons which behave differently during ECFF will have an <isif condition=”${isExpressCheckoutFinalize}”> to control the button behavior. In **app\_storefront\_core/cartridge/templates/default/checkout/cart/cart.isml**, during ECFF, a button will be displayed that upon clicking, will run the “AfterpayExpress-ContinueFinalize'' endpoint. This is a endpoint that will navigate to the cart summary page. When not in ECFF, the button has the (id=”afterpay-express-button”), which has been set as the target for launching the Afterpay Express Checkout popup.



If your site is designed in such a way that ECFF wouldn’t make sense and you’d prefer the user go through Express Checkout again whenever they click the “Checkout with Afterpay” or “Buy now with Afterpay” buttons, this is behavior that’s easily changeable in the template.

### Checkout Buttons on Product Details Page

The product details template (**app\_storefront\_core/cartridge/templates/default/product/productcontent.isml**), contains this section:



The client-side Javascript is required since the “Buy Now with Afterpay” button is located in a section of the page which is dynamically generated via an ajax call. Thus we need both event handlers to make sure that when the button is loaded, initAfterpay() will be called, which sets the onclick handler on that button.

initAfterpay() takes an object as input which contains settings to control the Express Checkout behavior. For the product details page, we need to pass in a productIdSelector, which is a css selector for the <input> element containing the pid of the product to buy, and a productQuantitySelector, which is a css selector for the quantity. When these are passed in, the Express Checkout will first add the item to the cart before the normal checkout process.

### Client-side Javascript on cart checkout page

In SiteGenesis, there is a radio button which can be used to toggle between in-store pickup and home delivery when those choices are both available for a product. Since the toggle only does an ajax call and does not cause a full page reload, the express checkout button will need to know when the in-store pickup vs home delivery state has potentially changed. In order for the express checkout popup to correctly detect that state, we currently have a javascript snippet on the page to detect the radio button click, and quickly do an ajax call to the merchant site to determine whether the checkout qualifies for in-store pickup.  
  
In **app\_storefront\_core/cartridge/templates/default/checkout/cart/cart.isml**:



In **cartridges/int\_afterpay\_core/cartridge/static/default/js/afterpayExpress.js**:

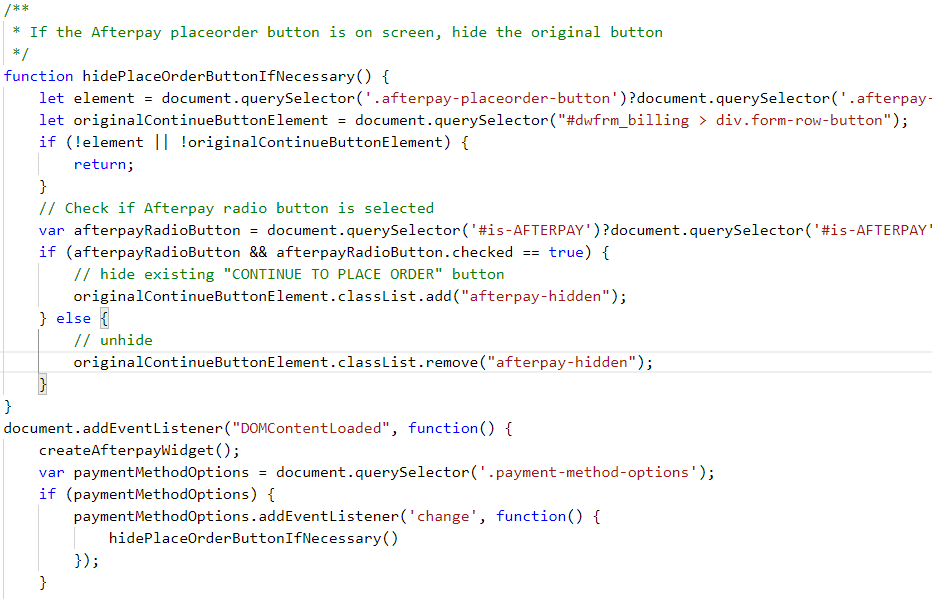


On your site, if there are similar actions which may affect cart-eligibility for in-store pickups which do not cause a page reload, you may need to call reinitializeAfterpayPopup() as well.

### Client-Side Javascript on Billing Page

There is some client-side Javascript on the **Billing Page** which is used to toggle between the “Place order with Afterpay” button, and the default “Continue to Place Order” button. During Express Checkout Finalization Flow, if Afterpay is still the payment method, we display the “Place order with Afterpay” button (and display the Afterpay Widget).

In **app\_storefront\_core/cartridge/templates/default/checkout/billing/paymentmethods.isml**:

****

Whenever the Afterpay button (selector **.afterpay-placeorder-button**) appears on the screen, the default button (with selector **#dwfrm\_billing > div.form-row-button**) will be hidden.  
  
If you’ve made modifications to the billing page, please keep this in mind as you may need to customize the behavior to fit your site.

# 

# Operations / Maintenance

## Data Storage

Some additional custom attributes are stored against system objects. The following table lists these attributes and their uses, other than Site Preferences (which are listed under section 3.3.2).

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Attribute | Type | Description |
| OrganizationPreferences | flowIPAddresses (flowIPAddresses) | Text | IP addresses |
| Order | Afterpay Payment Method (apIsAfterpayOrder) | Boolean | Flags the order as an Afterpay order |
| PaymentTransaction | Afterpay Token (aToken) | String | The token used to place order in Afterpay |
| PaymentTransaction | Afterpay Initial Status  (apInitialStatus) | String | The pre-approval status first returned |
| PaymentTransaction | Afterpay Payment Mode (apPaymentMode) | String | The payment mode used  PaymentTransaction |
| PaymentTransaction | Afterpay Direct Payment  (apDirectPayment Status) | String | The status of Direct Payment, if this payment mode is used |
| PaymentTransaction | Afterpay Authorise Status  (apAuthoriseStatus ) | String | The status of Authorise, if  this payment mode is used |
| PaymentTransaction | Afterpay Order ID (apPaymentD) | String | The Payment ID generated by Afterpay |
| PaymentTransaction | Afterpay Refund ID (apRefundID) | String | Refund ID |
| PaymentTransaction | Auth Expiry (apAuthExpiry) | String | The time the Afterpay Order Auth Expires |
| PaymentTransaction | Payment Event ID (apPaymentEventID ) | String | The unique ID for the Afterpay Auth event |
| PaymentTransaction | Afterpay Express Checkout (apExpressCheckout) | Boolean | Flags the order as an express checkout order |
| PaymentTransaction | Afterpay Express Checkout Checksum (apExpressCheckoutChecksum) | String | Checksum from the Afterpay Widget displayed before the order was created. |

## Availability

If the Afterpay service is unavailable, the shopper will not be able to checkout using this payment method.

The service availability can be tracked in SFCC using **Administration > Operations > Service Status**.

## Support

For support, please contact your Afterpay account contact.

# 

# User Guide

## Roles, Responsibilities

Integration of this cartridge will typically be done by a SFCC developer.

Afterpay will provide access keys for be used with the API.

## Business Manager Refund Transaction

For the Afterpay integration to work, the following needs to be configured in the Business manager.

1. Upload the cartridge bm\_afterpay and int\_afterpay\_core into server.
2. Go to Business Manager Menu > Administration > Manage Sites > Business Manager > Settings > Cartridges input field add cartridge name “bm\_afterpay: int\_afterpay\_core.
3. Administration > Organization > Roles > Administrator > Business manager modules > Select context (select your site) > enable the Afterpay transactions management

# 

# Release History

|  |  |  |
| --- | --- | --- |
| Version | Date | Changes |
| 23.1.0 | 28 Feb 2022 | Minor bug fixes, code improvements |
| 23.0.1 | 18 Nov 2021 | Security improvements, Minor bug fixes |
| 23.0.0 | 15 Sep 2021 | Afterpay JS library, Multilingual |
| 21.1.0 | 30 Apr 2021 | Express Checkout, V2 API |
| 20.1.0 | 10 Jan 2020 | No changes |
| 18.3 | 07 May 2018 | Controller version |
| 18.3 | 03 Sep 2016 | Initial release |