# B2B2C Commerce Developer Guide

Version 53.0, Winter ‘22


@salesforcedocs

Last updated: July 28, 2021

# CONTENTS

# CHAPTER 1 B2B2C Commerce Developer Guide

Learn how to build integration packages and custom components for Salesforce B2B2C Commerce. This guide introduces the B2B2C data model, which you can augment with custom display headers, footers, and cart badges. Create integration packages to add third-party payments, taxes, and shipping rate providers.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

B2B2C Commerce is built natively on the Salesforce Customer 360 Platform, the world's most trusted commerce solution. B2B2C helps you launch stores quickly with guided setup, low-code configuration and branding, simplified data import, and shared workflows across clouds. The B2B2C Commerce platform is open and extensible. Its minimal configuration requirements and declarative programming layer let Commerce Admins and Merchandisers quickly accomplish setup tasks so that developers can focus on enabling integrations and extending platform capabilities.

Partners create personalized shopping experiences with drag-and-drop storefront theming and Einstein-enabled search, all within an integrated data model for Order Management, CMS, Sales, and Service. A rich and growing AppExchange includes integration packages from leading, third-party partners.

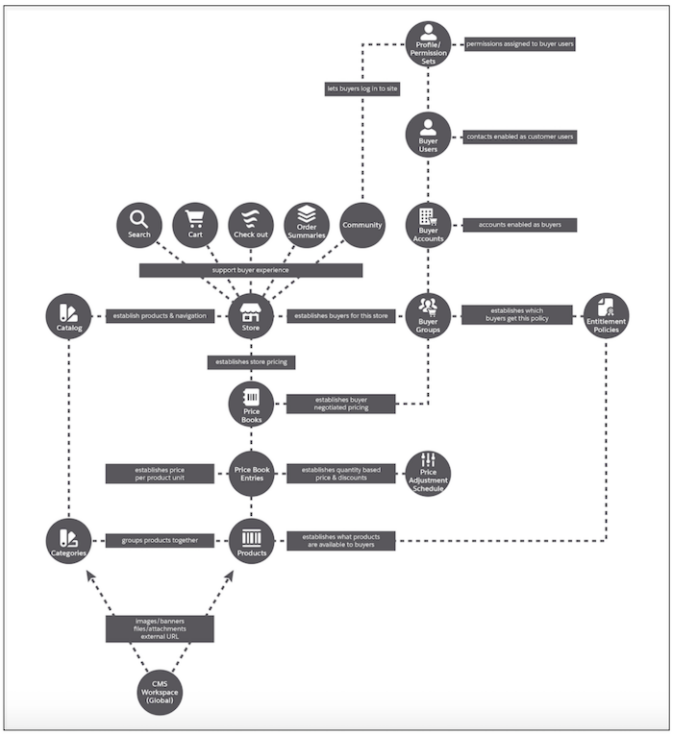
# CHAPTER 2 B2B2C Commerce Data Model

The B2B2C Commerce store template is built on a preconfigured data model. The data model supports standard and customizable business objects for a multitude of business relationships and interoperability with Lightning B2B, Salesforce Order Management, and Service Cloud.

The B2B2C data model connects the store objects. Default object relationships support a full-featured B2C webstore experience.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.




Commerce Admins and Merchandisers fill out the experience by setting up shopper groups, importing product lists, branding store pages, and linking third-party tax, shipping, and payment providers. They perform these tasks by clicking through the B2B2C Commerce template in Experience Builder with little or no coding.

Here's a summary of the default Lightning B2C data model objects.

- For a functional definition of each object, see Key Concepts.
- For an introduction to comparable data model business objects, see B2B Commerce on Lighting Data Model.

| Data Model Object | API Name | Description |
|---|---|---|
| Store | Webstore | Includes these fields: supported currencies, languages, price books, and results displayed per page, which are customizable. |
| Catalog | ProductCatalog | The Commerce Admin or Merchandiser uses B2C data import to set up. |
| Category | ProductCategory | The Commerce Admin or Merchandiser uses B2C data import to fill in a default compact layout, which includes name, catalog, category, search order, and so on. Layout is customizable. |
| Entitlement Policy | CommerceEntitlementPolicy | Filtered by BuyerGroup membership. Includes CanViewPrice and CanViewProduct fields, which are customizable. |
| Product | Product2 | The Commerce Admin or Merchandiser uses B2C data import to fill in a default compact layout, which includes a variety of customizable fields (name, family, and so on). |
| Buyer Account | BuyerAccount | The shopper's financial information, including credit and order limits, some of which pertain only to B2B. B2C Buyer Account is established when a shopper self-registers. |
| Buyer Group | BuyerGroup | One per B2B2C webstore, created by default during B2C data import. BuyerGroup name and description are customizable. |
| Buyer Group Member | BuyerGroupMember | Established when shoppers self-register or shop as guests. Buyer Account Name is customizable. |
| Price Book | PriceBook2 | Typically added during setup with your store's data import, but you can add custom fields. |

| Data Model Object | API Name | Description |
| --- | --- | --- |
| Price Book Entry | PricebookEntry | Typically preset, conforming to Price Book object when imported, but supports custom fields. |

# CHAPTER 3 B2B2C Integration Architecture

A predefined set of flows simplifies package integration for B2B2C Commerce tax, shipping, and payment providers. B2B2C Commerce integrations with external providers are asynchronous to provide a consistent, quality customer journey. The integrations are embedded into the cart and checkout experience, triggered by shopper interactions with the storefront UI.

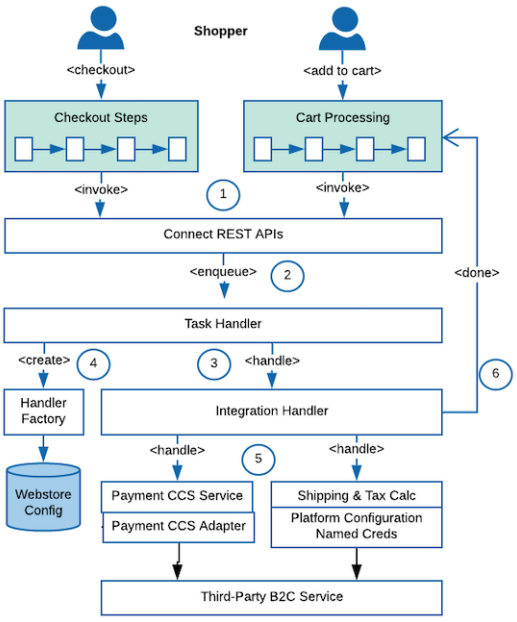
The B2B2C app supports these third-party integration points and flows.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

- **Shipping**—Calculates and writes shipping costs per delivery group.
- **Taxes**—Calculates and adds tax prices for cart items.
- **Payment**—Uses the Salesforce Payment Adapter framework to fetch tokens and authorizations (and manage exceptions, such as fraud and insufficient credit) from service providers during checkout via a Payments Gateway. The integrated Salesforce Order Management module handles additional services, including capture and refund.

Here's how the various components work together to form the integration engine.




A predefined set of flows manages shipping, inventory, pricing, and tax integrations.

1. **Cart and checkout processing**—Entering a delivery address initiates shipping charges and tax calculation. Order placement triggers payment processing. An integrated Salesforce Order Management component processes refunds, capture, and more.

2. **Connect REST API**—Service for these discrete APIs for shipping, tax, and payment integrations execute tasks asynchronously and is distinct from the Salesforce B2B Checkout Subflows implementation.
3. **Task handler**—The task handler is implemented as an MQ handler and invoked by the queue manager when the integration task is picked up for processing. The integration handler is responsible for delegating the task to the integration implementation, which the Commerce Admin or Merchandiser specifies when setting up the store.
4. **Handler Factory**—Responsible for creating an integration handler that maps to the implementation chosen by the Commerce Admin during store setup.
5. **Integration services**—Using the store's configured named credentials, a gateway conveys requests for third-party tax and shipping calculations. The CCS (Core Commerce Services) Adapter and Service Salesforce Payment Adapter request and receive authorizations, tokens, and exceptions from a third-party service via a Payments Gateway.
6. **API responses**—Successful results and exceptions with UI-facing help messages are returned to the shopper's browser.

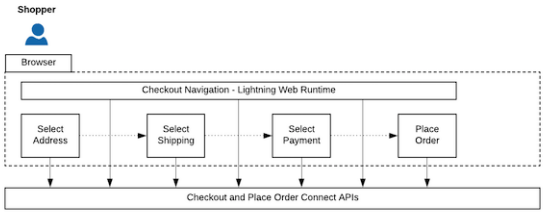
# CHAPTER 4 B2B2C Checkout API Integration

In this chapter ...



Checkout APIs execute calls to native or third-party shipping, tax—tax and shipping calculation share an API trigger—and payment integrations.

The Checkout APIs provide integration points to third-party services. Calls to the services are triggered when shoppers click Checkout or revisit a checkout session from their browser. Store components that embed the Checkout APIs are implemented in Lightning Web Runtime.




EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

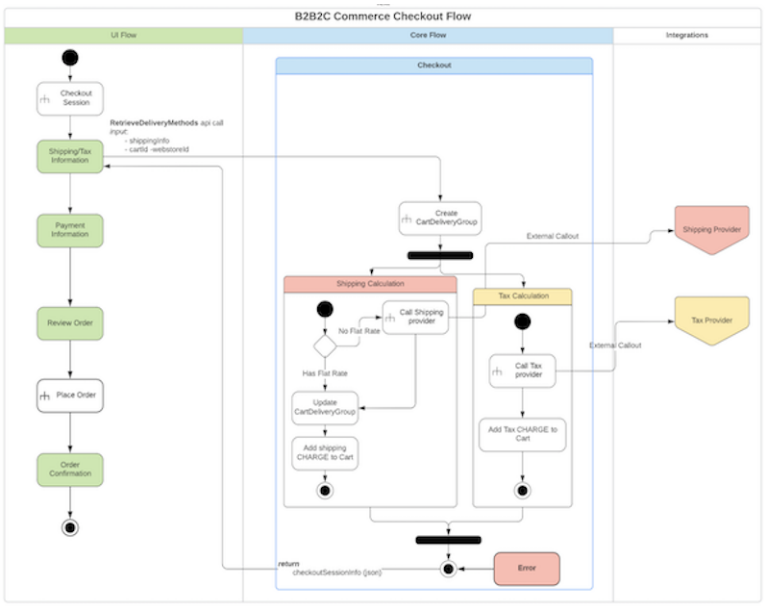
# Shipping and Tax Integration

A single API call fetches both shipping and tax costs for cart items.

Shoppers trigger an asynchronous shipping cost and tax API call to service providers when they enter a shipping address. Because shipping and tax providers require the same inputs to make their respective calculations, a single API call fetches both shipping and tax costs for cart items.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.




Here's the structure of RetrieveDeliveryMethod, the triggering shipping and tax calculation API call.

```
{
   "id" : ID : // the cart delivery group id
   "deliveryMethods" :  DeliveryMethodCollectionRepresentation :
   "deliveryAddress" : AddressRepresentation : // selected delivery address
   "cartItems" : CartItemCollectionRepresentation :
}
```

Here's a sample DeliveryGroupRepresentation.

```
{
   "id":"2Dg456789012345678AAA",
   "cartItems":{
       Total: 1,
       carItems:
       [
         {
           "cartItemId":"0a9456789012345678AAA",
           "productId":"01txx0000006i44AAA",
           "name":"shower bar",
           "listPrice":"29.95",
           "salesPrice":"20.00",
           "totalTax":"1.85",
           "totalAmount":"1",
           "totalPrice":"31.80",
           "totalAdjustmentAmount":"31.80"
```

```
          }
        ]
    },
    "deliveryMethods":{
      "total: 2",
      "items" :
      [
        {
          "id":"2Dm456789012345678AAA",
          "shippingFee":14.00,
          "currencyCode":"USD",
          "carrier":"UPS",
          "classOfService":"Next Day Shipping",
          "timeOfArrival" : "2020-11-05T13:15:30Z"
          "selected":true
        },
        {
          "id":"2Dm123789012345678EAA",
          "shippingFee":9.00,
          "currencyCode":"USD",
          "carrier":"UPS",
          "classOfService":"Three Day Shipping",
          "timeOfArrival" : "2020-11-07T16:15:30Z"
          "selected":false
        }
      ]},
    "shippingAddress":{
       "AddressType":"Shipping",
       "City":"Boston",
       "Country":"USA",
       "Id":"81Wxx0000000001EAA",
       "IsPrimary":true,
       "Name":"Home Address",
       "PostalCode":"01234",
       "State":"MA",
       "Street":"1 Milk Street"
    }
}
```

### B2B2C Shipping Reference Packages

A reference shipping integration package supports both B2B2C Commerce and Lightning B2B implementations. Use it as a template to create your own shipping calculation package.

### B2B2C Tax Reference Packages

A reference tax integration package supports both B2B2C Commerce and Lightning B2B implementations. Use it as a template to create your own shipping calculation package..

# B2B2C Shipping Reference Packages

A reference shipping integration package supports both B2B2C Commerce and Lightning B2B implementations. Use it as a template to create your own shipping calculation package.

Clone or download the package: Shipping Reference Integration Package.

This guide provides step-by-step procedures for creating and deploying a payments package to your B2B2C store. The procedure for setting up a shipping package is similar.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# B2B2C Tax Reference Packages

A reference tax integration package supports both B2B2C Commerce and Lightning B2B implementations. Use it as a template to create your own shipping calculation package..

Clone or download the package: Tax Reference Integration Package

This guide describes how to set up development environments and provides step-by-step procedures for creating and deploying a payments package to your B2B2C store. The procedure for setting up a tax package is comparable.

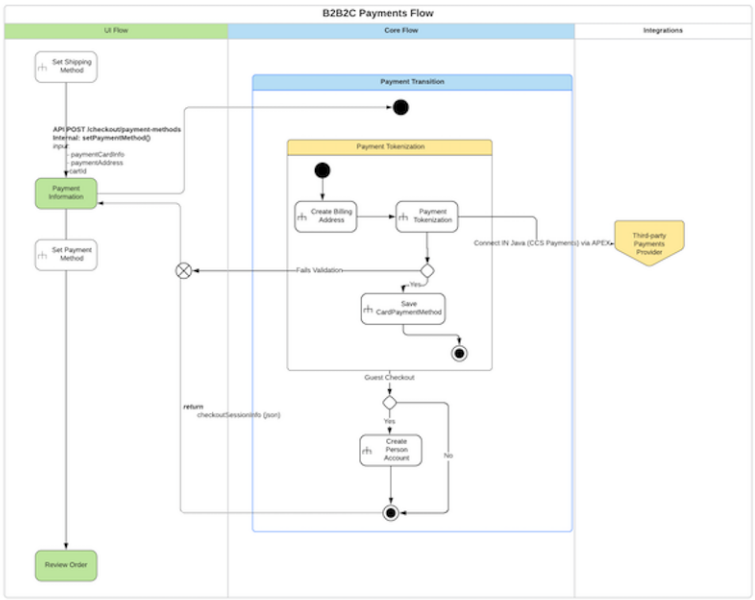
EDITIONS

Available in: Lightning communities accessed through Lightning Experience and is available in **Enterprise**, **Unlimited**, and **Developer** editions.

# Payments Integration

B2B2C Commerce, like Lightning B2B, uses the Salesforce Payment Adapter framework to fetch tokens and authorizations from service providers during checkout.

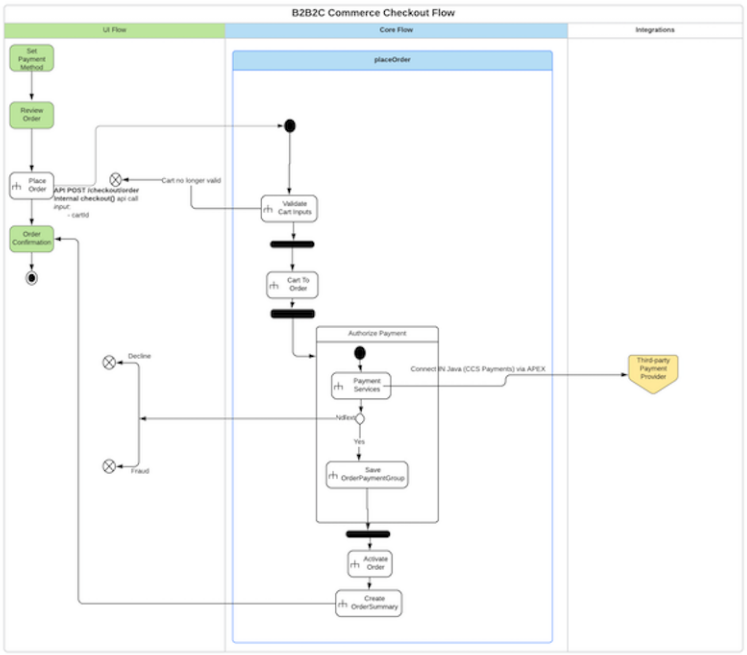
A shopper UI action triggers the SetPaymentMethod, which interacts with an integrated third-party payment provider to reserve funds.



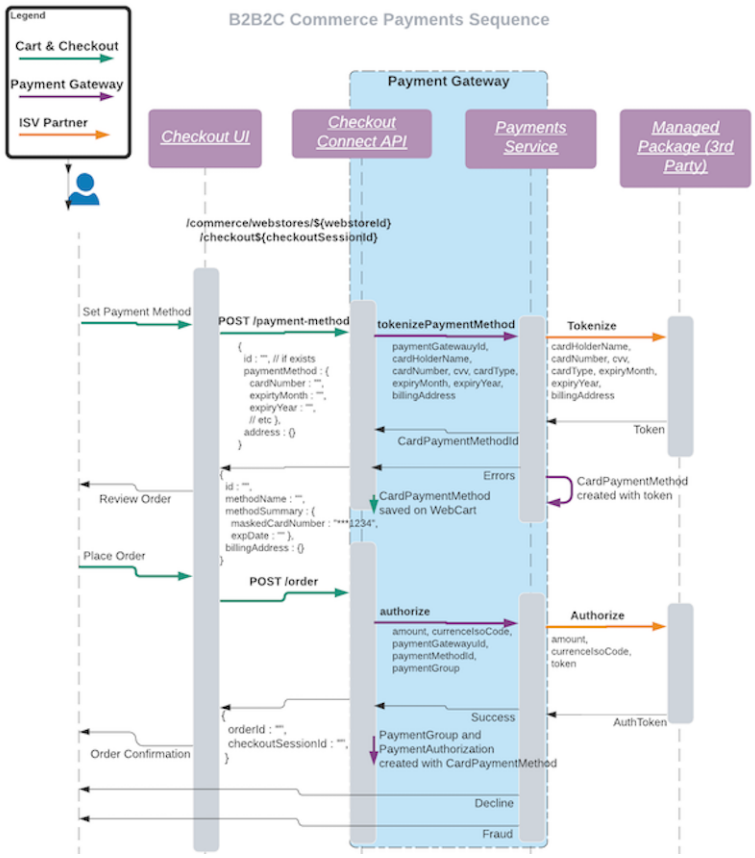

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

After the shopper reviews and approves the order, the PlaceOrder API completes the sale.




Integration packages for payment providers are interoperable across the B2B2C Commerce solution and Salesforce Order Management as well as Salesforce B2B. Here's a more detailed look at the enabling payment APIs.



Payments Flow APIs

B2B2C Commerce implements a group of payment APIs to support tokenizing shopper credit cards without storing that information natively.

Payments Gateway

The B2B2C Commerce Payments Gateway parses key object fields for tokenize, authorize, capture, refund, and other checkout API requests.

B2B2C Payments Reference Packages

A reference payments integration package supports both B2B2C Commerce and B2B Lightning implementations. Use it as a template to create your own payments package.

# Payments Flow APIs

B2B2C Commerce implements a group of payment APIs to support tokenizing shopper credit cards without storing that information natively.

Shopper actions trigger asynchronous payment API calls to a third-party service provider via the Payment Adapter framework. The APIs support tokenizing shopper credit cards so that no personal information is stored natively. Together with the licensed Salesforce Order Management (SOM) or 1C OM, the Payment Adapter framework exposes additional APIs. SOM manages the capture and refund flows and provides an integrated customer order lifecycle, including fulfillment and service.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

The following APIs initiate payments processing.

- **SetPaymentMethod** tokenizes the shopper's credit card and returns a token.
- **PlaceOrder** (returns Auth Code) validates the cart, reserves inventory, converts cart to order, authorizes payment, and activates the order in SOM. The payment authorization reserves funds from the available credit of the credit card, but it's not a payment. To the shopper, it can display as pending.

After the transaction is tokenized and the order placed, any of the following APIs and corresponding transactions can occur.


- **Authorization Reversal** releases the funds reserved by the payment authorization, removing them from "pending transactions."
- **Capture** consumes the funds reserved by the payment authorization.
- **Sale** is a transaction type where Authorization and Capture are executed as part of a single request. If successful, the order is fulfilled immediately.
- **Void** cancels the transfer of funds to the merchant account before settlement. When a payment is processed, funds are held. The balance is deducted from the customer's credit limit, but not yet transferred to the merchant account. At a later point all transactions are batched for settlement and the funds are transferred to the merchant's bank account. A transaction can be voided after purchase but before settlement.
- **Refund** is a transaction request that transfers the amount from the merchant's account to the customer's account.

For more information on these APIs, see Commerce Connect REST Payment APIs.

# Payments Gateway

The B2B2C Commerce Payments Gateway parses key object fields for tokenize, authorize, capture, refund, and other checkout API requests.

Note: Payments Gateway uses the CommercePayments Apex namespace. For more information, see the Apex Reference Guide.

The B2B2C Commerce Payments Gateway parses key object fields passed by the UI Checkout components. These fields include data used to tokenize and authorize the payment request. For example, the Payments Gateway requires a tokenize request to pass the following.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

```
cardPaymentMethod: {
    cardHolderName: <string>,
    cardNumber: <string>,
    expiryMonth: <string>,
    expiryYear: <string>,
    cvv: <string>
}
address: {
    street: <string>,
    city: <string>,
    state: <string>,
    postalCode: <string>,
    country: <string>
}
```

Authorization, capture, refund, and other requests also pass these objects. Review the B2B2C Stripe adapter samples in the Payments Reference Package on page 13 to see how the examples handle these objects for transmission to Stripe.

Examples for the Payments Adapter include:

- `tokenizeRequest → commercePayments.PaymentMethodTokenizationRequest`
- `authRequest → commercePayments.AuthorizationRequest`
- `captureRequest → commercePayments.CaptureRequest`
- `refundRequest → commercePayments.ReferencedRefundRequest`

# B2B2C Payments Reference Packages

A reference payments integration package supports both B2B2C Commerce and B2B Lightning implementations. Use it as a template to create your own payments package.

Clone or download the package: Payments Reference Integration Package

This guide describes how to set up development environments and provides step-by-step procedures for creating and deploying a payments package to your B2B2C store.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience and in **Enterprise**, **Unlimited**, and **Developer** editions.

# CHAPTER 5 B2B2C Product Import API

After creating a B2B2C store, administrators and merchandisers import product data by creating a CSV file formatted for B2B2C. They then declaratively execute the import using the B2B2C Commerce App, which calls the Product Import API. The B2B2C Product Import API populates the store with available products and associated entities, such as price books and categories.

No knowledge of the B2B2C data model is required for importing data. The Import API creates the necessary relationships in the data model.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience and in **Enterprise**, **Unlimited**, and **Developer** editions.

To support custom requirements, B2B2C Commerce exposes the Product Import API, which is a standard Connect Rest API. Its properties include IDs for entitlement policies, price books (including strikethrough price books), catalogs, and versioning for the CSV source.

For developer reference information, see **B2B2C Product Import API**.

# CHAPTER 6 Set Up a B2B2C Commerce Development Environment

In this chapter ...



Salesforce recommends the Salesforce Developer Experience (SFDX) environment for building your tax, shipping, and payment integration packages and Lightning web components (LWC). SFDX provides easy access to Salesforce extensions and GitHub repositories containing LWC templates and reference packages. SFDX also integrates with Salesforce CLI, the Visual Studio Code editor with the Salesforce Extension Pack, and a B2C plug-in to quickly deploy packages and components to scratch orgs and stores.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Install the Visual Studio Code Editor

The free Visual Studio (VS) Code editor is an ideal development environment for creating, compiling, displaying, and debugging Salesforce Lightning web components and B2B2C integration packages.

The free VS Code editor is open source and optimized for cloud and web coding. The project folder combines, compiles, and displays the output of the HTML, JavaScript, and CSS files for your component.

- Install the open source Visual Studio Code editor

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Get Salesforce Plug-ins for VS Code

The VS Code editor provides access to Salesforce plug-ins that support creating integration packages and custom B2B2C components. The extensions include features for working with development orgs (scratch orgs, sandboxes, and DE orgs), Lightning web components (LWC), and Visualforce.

For more information, see the Salesforce Extensions for VS Code Overview.

1. Open the VS Code app.
2. Navigate to **View** > **Extensions**.
3. In the search field, enter `Salesforce`.
4. To install the package, click **Salesforce Extension Pack** (v. 51.4.0 or later).

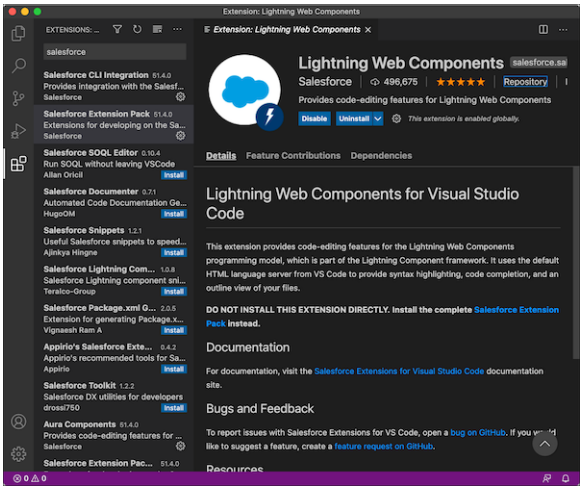
EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

The Salesforce Extension Pack includes:

- Salesforce SFDX CLI integration for VS Code
- ESLint JavaScript integration for VS Code
- LWC for VS Code, which uses the VS Code HTML server to provide code editing features for the LWC program model, including syntax highlighting, code completion, and file outlining

For an overview, see Salesforce Extensions for LWC.



# Install the SFDX CLI

The Salesforce DX (SFDX) CLI synchronizes your source code between the Salesforce orgs that you deploy to and your version control system.

Install the VS Code editor, and download the Salesforce extensions.

- Install the Salesforce SFDX CLI

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Install the SFDX Commerce B2B2C Plug-in

The SFDX Commerce B2B2C plug-in helps you perform a variety of B2B2C development tasks. Use it to set up and deploy integration packages to scratch environments for testing or to an org.

You configure a JSON file that defines store parameters. Then use the CLI to create and authorize a Dev Hub, stand up a store or sandbox, deploy an integration package, import product data, and more.

1. In the VS Code Editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter `sfdx plugins:install @salesforce/commerce`.

Note: Both SFDX and the Commerce B2B2C plug-in are automatically updated.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# CHAPTER 7 Creating B2B2C Payments Packages

In this chapter ...



Create and deploy an unmanaged payment package to your scratch org or store using an SFDX Project in the VS Code editor.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience and in **Enterprise**, **Unlimited**, and **Developer** editions.

# Enable Dev Hub

The Dev Hub org determines where scratch orgs are authorized for your package project.

Note: After you enable Dev Hub, you can't disable it.

1. Log in as System Administrator to your org.
2. From Setup, in the Quick Find box, enter *DevHub*, and select **DevHub**. If you don't see DevHub in the Setup menu, make sure that your org is one of the supported editions.
3. To enable Dev Hub, click **Enable**.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience and in **Enterprise**, **Unlimited**, and **Developer** editions.

# Create an SFDX Package Project

Use SFDX in the VS Code editor to create a project for the package.

1. In the VS Code editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *SFDX*.
3. Select **SFDX: Create Project**.
4. Name the project.

   Example: `<MyPayments|Shipping|Tax|Package>`
5. Press **Enter**.
6. Select a folder to store the project.
7. Click **Create Project**.
8. Open sfdx-project.json and change the sfdcLoginURL field to contain the login URL to your Dev Hub.

   Example: `"sfdcLoginUrl": "http://localhost.internal.salesforce.com:6109"`
9. Under MyPaymentsPackage/config, add or edit project-scratch-def.json to prepend the Commerce namespace for the GitHub containing the integration package.

   Example: `{1CommerceRepo}/config/project-scratch-def.json`

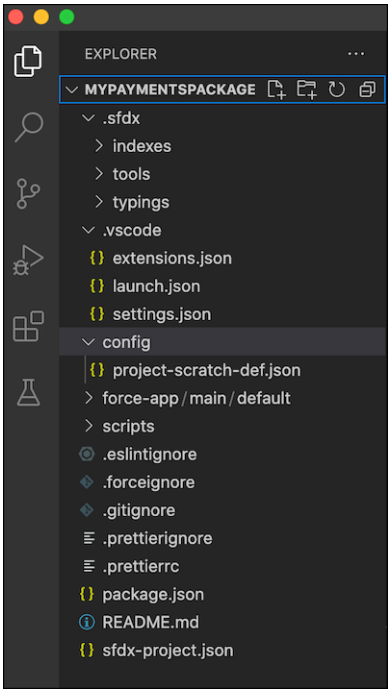
EDITIONS

Available in: Lightning communities accessed through Lightning Experience and in **Enterprise**, **Unlimited**, and **Developer** editions.

USER PERMISSIONS

To create a scratch org:
- System Administrator

Example:




# Authorize Your Dev Hub

Link your Dev Hub to the package project.

1. In the VS Code editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Scroll down to select **Authorize a Dev Hub**.
3. To enable Dev Hub, click **Enable**.
4. After prompted to confirm, optionally log in to your store with your System Administrator credentials.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Create a Scratch Org

After configuring an SFDX project and authorizing a Dev Hub, create a scratch org for testing the package. SFDX uses your project-scratch-def.json to stand up the scratch org.

1. In the VS Code editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *SFDX*.
3. Select **SFDX: Create a Default Scratch Org**.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Add Apex Adapter Classes

Create the Apex classes that serve as templates for package objects.

1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *SFDX*.
3. Select **SFDX: Create an Apex Class**.
4. Enter

   *{1commercerepo}/examples/checkout/payment-gateway-integration/Stripe/classes/StripeAdapter.cls*.
5. Press **Enter**.
6. For the directory, enter *force-app/main/default/classes*.
7. Press **Enter**.
8. Repeat steps 4 through 7 to name and choose a directory for each of the following.
   - {1commerceRepo}/examples/checkout/payment-gateway-integration/Stripe/classes/StripeAdapter.cls-meta.xml → force-app/main/default/classes/StripeAdapter.cls-meta.xml
   - {1commerceRepo}/examples/checkout/payment-gateway-integration/Stripe/classes/StripeValidationException.cls → force-app/main/default/classes/StripeValidationException.cls
   - {1commerceRepo}/examples/checkout/payment-gateway-integration/Stripe/classes/StripeValidationException.cls-meta.xml → force-app/main/default/classes/StripeValidationException.cls-meta.xml
   - {1commerceRepo}/examples/checkout/payment-gateway-integration/Stripe/classes/QueryUtils.cls → force-app/main/default/classes/QueryUtils.cls
   - {1commerceRepo}/examples/checkout/payment-gateway-integration/Stripe/classes/QueryUtils.cls-meta.xml → force-app/main/default/classes/QueryUtils.cls-meta.xml

The Apex adapter and related classes reside in your project's `force-app/main/default/classes` directory. Use the GitHub file names for the adapter, metadata, and query utilities files.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Add Reference Package Code

From the reference package in GitHub, fetch and paste code to your SFDX integration project files. Modify the code as needed.

- Copy and paste the code from the payments package (GitHub) files into the corresponding Apex CLS and meta.xml files in the `force-app/main/default/classes` directory.

  If you are creating your own unique package, modify file contents as required while retaining the file structure.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Create Named Credentials

Create named credentials to streamline authentication for the integration package.

1. In VS Code Explorer, under `force-app/main/default`, create a subdirectory called `namedCredentials`.
2. In `namedCredentials`, create a file and name it `stripe.namedCredential`.
3. Navigate to the /namedCredentials folder in the public repo of the Stripe integration payments package and copy the contents of the file:`Stripe.namedCredential`
4. Open `stripe.namedCredential`, and paste the contents copied in the previous step.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Deploying a B2B2C Package

Use SFDX, the Commerce UI, Workbench, or the SFDX B2C plug-in to deploy a package to a scratch org or store.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

Deploy a Package with SFDX
Push and deploy package source code to your Dev Hub org using SFDX commands.

Deploy a Package from the B2B2C Commerce UI
You can use the B2B2C Commerce UI to install a package into an org or store.

Deploy a Package with Workbench
You can use Workbench to install a package to your org or store.

Deploy a Package with the SFDX Commerce B2B2C Plug-in
Install a package with the SFDX Commerce B2B2C plug-in and map the gateway from the Commerce UI.

# Deploy a Package with SFDX

Push and deploy package source code to your Dev Hub org using SFDX commands.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *`SFDX: Push Source to Default Scratch Org`*.
3. To create the package, enter *`sfdx force:package:create -n PACKAGENAME -t Unlocked -r force-app —nonamespace`*.
4. To create the package version, from the command line inside the project directory, enter *`sfdx force:package:version:create -p PACKAGENAME -d force-app -v DEVHUBUSERNAME -k 123456 --skipvalidation ——wait 10`*, where `DEVHUBUSERNAME` is the name of your authorized Dev Hub.
5. To install the package, enter *`sfdx force:package:install --package <PACKAGENAME> --targetusername <jdoe@example.com>`*, where `jdoe@example.com` is the System Administrator's username.

   You can optionally install the package from the B2B2C Commerce app. To do so, for Step 4, create the package version and copy the returned installation URL, which looks like this:
   `https://login.salesforce.com/packaging/installPackage.apexp?p0=04txx0000004JDCAA2`.
   Replace the host in the URL with the host of the scratch org you're deploying to.

6. To retrieve the metadata in package format, enter `sfdx force:mdapi:retrieve -s -r ~/Documents -u MYSCRATCHORG -p PACKAGENAME`.

   Make sure that the `-u` variable is the name of the org where you installed the package in the previous step.

## Deploy a Package from the B2B2C Commerce UI

You can use the B2B2C Commerce UI to install a package into an org or store.

All package components, Apex classes, named credentials, and payment gateway providers must reside in the SFDX project.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

1. From Salesforce Setup, in the Quick Find box, enter `Packages`, and then select **Packages**.
2. Click **New** and enter the package details.
3. Click **Save**.
4. To add package components, click **Add Components**.
5. From the component type dropdown, select the the Payment Gateway adapter, Apex classes, the named credentials, and the Payment Gateway provider.
6. For each component, click **Add to Package**.
7. Click **Upload**.

   When your package is uploaded successfully, you receive an email that includes an installation link. Wait five minutes for Salesforce to activate the package.

8. To complete deployment, click the installation link in the email.
9. To retrieve the package, from the project directory of the VS Code SFDX environment, enter `sfdx force:mdapi:retrieve -s -r ~/FOLDERNAME -u MYSCRATCHORG -p PACKAGENAME`.

## Deploy a Package with Workbench

You can use Workbench to install a package to your org or store.

All package components, Apex classes, and payment gateway providers must reside in the SFDX project.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

1. Navigate to https://workbench.developerforce.com.
2. In the Environment field, select **Production**.
3. Select the latest API version.
4. Select **I agree to the terms of service**, and click **Login With Salesforce**.
5. From the Migration menu in the navigation bar, select **Deploy**.
6. Select the integration packager (MYPACKAGE.zip).
7. Select **Single Package**.
8. Click **Next**.
9. Click **Deploy**.
10. To add named credentials for the package, navigate to **Setup** > **Security** > **Named Credentials**.
11. Click **New Named Credential**.

**12.** Enter the following information, and make a note of the name and label for future tasks.

**a.** Label: *<AdapterName>_NC*

**b.** Name: *<AdapterName>_NC*

**c.** URL: *<URL for provider>*, for example, `https://api.stripe.com`

**d.** Certificate: not required

**e.** Identity Type: Select **Named Principal**

**f.** Authentication Protocol: Select **Password Authentication**

**g.** Username: *<sysasdminUsernameforOrg>*

**h.** Password: *<sysadminPasswordforOrg>*

**i.** Select **Generate Authorization Header**

**j.** Select **Allow Merge Fields in HTTP Header**

**k.** Do not select **Allow Merge Fields in HTTP Body**

**13.** Click **Save**.

**14.** To create the gateway and assign it to your org or store:

**a.** In your org or store, select **Quick Access** > **Developer Console**.

**b.** To create the gateway to the store, in the console, enter *PaymentGatewayPostInstall.run(String <storeName>)*.

# Deploy a Package with the SFDX Commerce B2B2C Plug-in

Install a package with the SFDX Commerce B2B2C plug-in and map the gateway from the Commerce UI.

All package components, Apex classes, named credentials, and payment gateway providers must reside in the SFDX project. Deploying the adapter classes requires placing the Apex source in the root B2B2C plug-in folder. The folder's location depends on how you work with the GitHub repo.

- If you're executing the Commerce B2B2C plug-in from a cloned copy of the Commerce repo, the base folder resides in the cloned directory.
- If you installed the SFDX plug-ins in your CLI and VS Code development environment, the base folder is in the root directory (for example, in UNIX, `~/.b2c`).

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

Before deploying the Apex classes, make sure that the directory containing them has this structure in the PACKAGENAME folder (for example, Stripe).

- ```
  <?xml version="1.0" encoding="UTF-8"?>
  <Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
      <members>StripeAdapter</members>
      <members>StripeValidationExeption</members>
      <name>ApexClass</name>
    </types>
    <types>
      <members>*</members>
      <name>NamedCredential</name>
    </types>
    <version>49.0</version>
  </Package>
  ```

Make sure the Classes folder includes:

- PACKAGENAMEAdapter.cls (for example, StripeAdapter.cls)
- PACKAGENAMEAdapter.cls-meta.xml (for example, StripeAdapter.cls-meta.xml)

Make sure the namedCredentials folder includes:

- PACKAGENAME.namedCredential (for example, Stripe.namedCredential)

1. In the VS Code editor, open the devhub-configuration.json file.
2. Set the store paymentAdapter variable to PACKAGENAME. For example: `"paymentAdapter": "stripe"`.
3. Open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
4. To install the adapter, enter *`sfdx b2c:payments:quickstart:setup`*.
5. To add named credentials, navigate to **Setup > Security > Named Credentials**.
6. Click **New Named Credential**.
7. Note: In substeps **a** and **b**, make a note of the name and label you enter for related tasks.

   Enter the following information:

   a. Label: *`&lt;AdapterName&gt;_NC`*
   b. Name: *`&lt;AdapterName&gt;_NC`*
   c. URL: *`&lt;URL for provider&gt;`*, for example, `https://api.stripe.com`
   d. Certificate: not required
   e. Identity Type: Select **Named Principal**
   f. Authentication Protocol: Select **Password Authentication**
   g. Username: *`&lt;sysasdminUsernameforOrg&gt;`*
   h. Password: *`&lt;sysadminPasswordforOrg&gt;`*
   i. Select **Generate Authorization Header**
   j. Select **Allow Merge Fields in HTTP Header**
   k. Do not select **Allow Merge Fields in HTTP Body**

8. Click **Save**.
9. To create the payment gateway, log in to Salesforce as an admin, and click **Lightning App Launcher**.
   a. Search for and select **Payment Gateways**.
  ```

b. Click **New**.

c. Enter a payment gateway name, for example, `Stripe`.

d. Select the Payment Gateway Provider <AdapterName>Adapter that you specified in the devhub-configuration.json file (in step 2).

e. Select the merchant credential <AdapterName>_NC.

f. Set the Status to **Active**.

10. To map the payment gateway to the store:

a. Navigate to **Store Home**.

b. Click the **Administration** tile.

c. From the left navigation menu, select **Card Payment Gateway**.

d. Click **Link Integration**.

e. Select the <AdapterName>Adapter.

f. Click **Next**.

g. Click **Confirm**.

# Listing a B2B2C Package on AppExchange

Make your integration package widely available by listing it on AppExchange. Vetted and tested shipping, tax, and payment packages are eligible to be listed on AppExchange.

Find out how to get listed on Salesforce AppExchange.

# CHAPTER 8 Creating Custom Display Components for B2B2C Commerce

In this chapter ...



The B2B2C Commerce App comes with standard components. The components are part of the store template that Commerce Admins and Merchandisers use to set up their customer experience. However, you can quickly and easily build custom components, such as headers, footers, and banners.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# B2B2C Lightning Web Components

Custom Lightning web components (LWC) are easy to build and perform well in the web browser that hosts your B2B2C store.

You build a component using core web browser building blocks: HTML, JavaScript, and CSS.

- HTML provides the structure for your component.
- JavaScript defines the core business logic, event handling, API calls to fetch page data, and related metadata.
- CSS provides the look, feel, and animation.

To build a custom component, you create an HTML file and a JavaScript file with the same name in a shared folder, also with the same name).

You then deploy the component to Experience Builder with B2B2C-specific metadata. The metadata includes crucial declarations to ensure that your component runs smoothly and safely. The Salesforce Lightning web runtime resolves, compiles, and bundles your files and constructs your component automatically.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# B2B2C Commerce APIs for Custom Components

B2B2C Commerce supports component API methods for custom branding and theming.

B2B2C Commerce includes a Commerce namespace that supports these component API methods for fetching and displaying data.

- `getAppContext()`—Retrieves the application context that doesn't rely on the current user session, such as the web store ID, or other application-related parameters.
- `getSessionContext()`—Retrieves session-related context, such as user ID and mode.
- `CartSummaryAdapter`—A wire adapter that retrieves the current cart summary

In addition, your component can use any Experience Cloud APIs, including:

- `getCommunityNavigationMenu()`—Retrieves menus for display in banner headers and footers.
- `cmsDeliveryApi`—A wire adapter that retrieves published CMS content versions for an Experience Builder site.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Create an SFDX Project

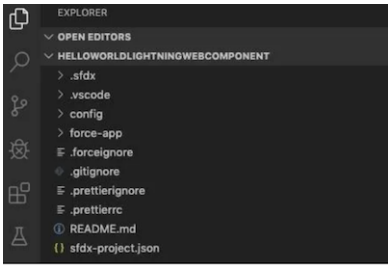
Create a Salesforce Developer Experience (SFDX) project to store your custom component files.

1. In the Visual Studio Code editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *`SFDX`*.
3. Select **SFDX: Create Project**.
4. Click **Enter**.
5. Name the project (for example, *`HelloWorldLightningWebComponent`*), and click **Enter**.
6. Select a folder to store the project.
7. Click **Create Project**.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

Example:




# Authorize an Org for an SFDX Project

Authorize the org to which SFDX deploys custom objects to streamline deployment.

1. In Visual Studio Code, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *SFDX*.
3. Select **SFDX: Authorize an Org**.
   SFDX opens the Salesforce login portal to your org in a separate browser window.
4. Log in using your Admin credentials.
5. If you're prompted to allow access, click **Allow**.
   After you authenticate in the browser, the CLI remembers your credentials. The success message looks like this:

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

Example:


# Create a Sample Lightning Web Component

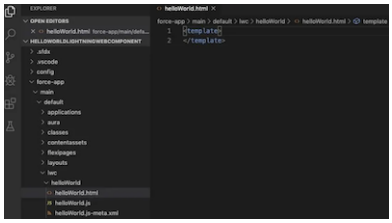
Create a sample custom component to deploy from SFDX to an authorized org.

This exercise creates a Lightning web component file structure where you can copy and paste content from a component in a GitHub repository.

1. In the Visual Studio Code editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *SFDX*.
3. Select **SFDX: Create Lightning Web Component**.
   Don't select SFDX: Create Lightning Component, which creates an Aura component.
4. Name the component (for example, *helloWorld*).
5. To accept the default `force-app/main/default/lwc` location, press Enter.
6. To view the new files in Visual Studio Code Explorer, press Enter.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.



7. Open the HTML file (for example, `helloWorld.html`), and write or copy and paste the HTML code for your project.
8. Save the file.
9. Open the JavaScript file (for example, `helloWorld.js`), and write or copy and paste the JavaScript code for your project.
10. Save the file.
11. Open the XML file (for example, `helloWorld.js-meta.xml`), and write or copy and paste the XML code for your project.

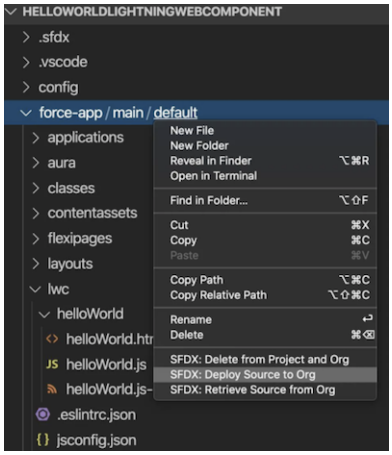
# Deploy a Custom Component to Your Org

Deploy a Lightning web component to use it on your store pages.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

1. From the component project directory in Visual Studio Code, right-click the default folder under force-app/main, and select **SFDX: Deploy Source to Org**.




2. On the Output tab of the integrated terminal, view the results of your deployment. SFDX displays a deployment status notice that includes an exit code, such as "SFDX: Deploy Source to Org ... ended with exit code 0". Exit code 0 means that the command ran successfully.

# Add a Component to Your B2B2C Store

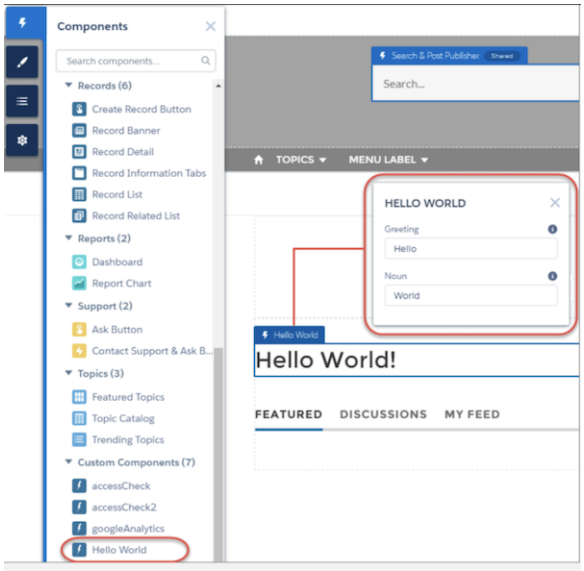
After you add a component to your canvas, you can edit its properties to customize it.

Note: Before placing a custom component, assign a custom theme layout to the page in Experience Builder.

1. In the VS Code editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *SFDX*.
3. Select **SFDX: Open Default Org**.

   Your B2B2C store opens in a separate browser.
4. In the left Navigation panel, click **Components**.

   The custom component displays under **Components** > **Custom Components**.
5. From the Custom area of the Lightning Components list, drag your Lightning web component to the page canvas.
6. To edit the component properties, select the component on the page canvas, and enter changes in the floating component property editor.




EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

# Prebuilt Custom B2B2C Components

Prebuilt custom B2B2C display components are not yet available, and we cannot provide a release date. When they become available, this topic will be updated.

Each B2B2C component includes JavaScript, CSS, HTML, and metadata files. Deploy the components to your store. Then, in Experience Builder, combine the components to create rich custom theming and shopper-friendly features, such as banners, store logos, navigation menus, search boxes, and linked lists for related products.

These components are ready to deploy components.

EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

**customCommerceFooter**

A footer for custom themed content that uses the SessionContext interface to update cached data.

After you deploy and place it on a store page canvas, the empty customCommerceFooter looks like this.

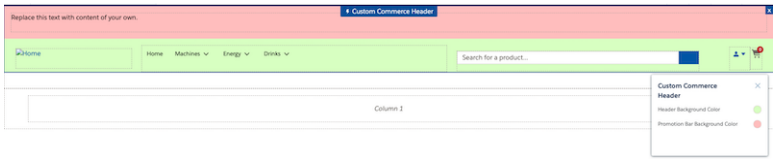

After rudimentary theming, this sample customCommerceFooter hosts a logo and link lists.


**customCommerceHeader**

A header container for placing a banner, logo, navigation, search, and cart badge. The component uses the SessionContext interface to update cached data.

After you deploy and place it on a store page canvas, the empty customCommerceHeader looks like this.


Here's a sample themed customCommerceHeader that features horizontal navigation, search, cartBadge, and logo custom components.




**cartBadge**

Use this component to extend store theming to the cart summary badge, which uses a wire adapter API to display the number of items in the cart.

**horizontalNavigation, verticalNavigation**

After deployment to a store, you can drag these menu container components into a header, footer, or another component.

**navigationMenu**

A hamburger menu container for mobile presentation.

**linksList**

A container to add links to related products, and so on.

**searchBox**

A search box container.

**storeLogo**

A container to receive and display a logo.

### Access Custom B2B2C Components

Prebuilt custom B2B2C display components are not yet available, and we cannot provide a release date. When they become available, this topic will be updated.

# Access Custom B2B2C Components

Prebuilt custom B2B2C display components are not yet available, and we cannot provide a release date. When they become available, this topic will be updated.

1. In the Visual Studio Code editor, open the Command Palette by pressing Ctrl+Shift+P (Windows) or Cmd+Shift+P (macOS).
2. Enter *SFDX*.
3. Select **SFDX: Create Lightning Web Component**.

   Don't select SFDX: Create Lightning Component, which creates an Aura component.
4. Name the component (for example, *myCustomComponents*).
5. To accept the default `force-app/main/default/lwc` location, press Enter.
6. To view the new files in Visual Studio Code Explorer, press Enter.
7. In a browser window, open the repository containing the B2B2C custom components.
8. Clone or download the repo to your SFDX project.

## EDITIONS

Available in: Lightning communities accessed through Lightning Experience in **Enterprise**, **Unlimited**, and **Developer** editions.

## USER PERMISSIONS

To create a custom component in an SFDX project:

- System Administrator

# CHAPTER 9 API End-of-Life

Salesforce is committed to supporting each API version for a minimum of three years from the date of first release. In order to mature and improve the quality and performance of the API, versions that are more than three years old might cease to be supported.

When an API version is to be deprecated, advance notice is given at least one year before support ends. Salesforce will directly notify customers using API versions planned for deprecation.