Digital Experience for Communications

Developing Open Storefront Framework Applications for Commerce Experience

January 2024

Digital Experience for Communications
Developing Open Storefront Framework Applications for Commerce Experience

January 2024

F77907-11

Copyright © 2024, Oracle and/or its affiliates.

Author: Tejaswi Tatavarthi

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Get Help

There are a number of ways to learn more about your product and interact with Oracle and other users.

Get Help in the Applications

Use help icons ⓘ to access help in the application. If you don't see any help icons on your page, click your user image or name in the global header and select Show Help Icons.

Get Support

You can get support at *My Oracle Support*. For accessible support, visit *Oracle Accessibility Learning and Support*.

Get Training

Increase your knowledge of Oracle Cloud by taking courses at *Oracle University*.

Join Our Community

Use *Cloud Customer Connect* to get information from industry experts at Oracle and in the partner community. You can join forums to connect with other customers, post questions, suggest *ideas* for product enhancements, and watch events.

Learn About Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program*. Videos included in this guide are provided as a media alternative for text-based topics also available in this guide.

Share Your Feedback

We welcome your feedback about Oracle Applications user assistance. If you need clarification, find an error, or just want to tell us what you found helpful, we'd like to hear from you.

You can email your feedback to *oracle_fusion_applications_help_ww_grp@oracle.com*.

Thanks for helping us improve our user assistance!

1 About This Guide

Audience and Scope

This guide is intended for developers who want to use the CX Commerce Open Storefront Framework to build storefront applications for Oracle Digital Experience for Communications.

This guide describes how you can set up the developer environment and create and deploy applications for communications-related products. It also explains how to create open storefront framework widgets for communications business scenarios. It assumes you're familiar with the CX Commerce Open Storefront Framework and have read the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

The Developing Open Storefront Framework Applications for Oracle CX Commerce guide contains essential information about developing and deploying storefront applications using the CX Commerce Open Storefront Framework. Most of the content in that guide is relevant for building storefront applications for Oracle Digital Experience for Communications. However, the procedures and settings that are specific to Oracle Digital Experience for Communications are available only in this guide. So, use this guide as a supplement to the latest version of the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

Related Guides

Refer to the guides in this table for more information about the tasks covered in this guide.

Guide	Description
Oracle CX Commerce Developing Open Storefront Framework Applications	Provides information on using the Open Storefront Framework to develop Oracle CX Commerce websites.
Oracle Digital Experience for Communications Implementing Buying Experience	Describes how to enable the omnichannel buying and ordering experience for end users or customers.
REST API for Oracle Digital Experience for Communications Buying Experience	Describes how to set up your instance of Care Experience in conjunction with the activities that you perform to set up Oracle CX Sales and Oracle Fusion Service.
REST API for Oracle CX Industries Framework	Describes how you can use Oracle REST APIs to view and manage data stored in Oracle CX Industries Framework.

2 Implementation Overview

Understand the Open Storefront and Self-Care Application

Use the communications open storefront and self-care reference application to develop and deliver user experiences for your storefront and self-care applications faster. The application is built on the Communications Open Storefront Framework (OSF), which is an extension of the CX Commerce OSF.

With this cloud-native application, you can develop more responsive and rich user experiences and accelerate new deployments using:

- Framework's rich library of reference components
- Buying Experience and Self-Care core functionality
- TM Forum（TMF）Open APIs
- Industry-standard tools and resources provided by CX Commerce

This chapter provides an overview of the Communications OSF and the additional components required for developing communications storefront and self-care applications.

Understand the Communications Open Storefront Framework Architecture

Use the Communications OSF architecture to plan and set up your development environment for rapid development and deployment of your storefront and self-care applications. You can develop and run applications in your development environment locally and then deploy them to the Oracle environment

The Communications OSF architecture includes the CX Commerce OSF components. For information on the CX Commerce OSF components, see the Understand the Open Storefront Framework chapter in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

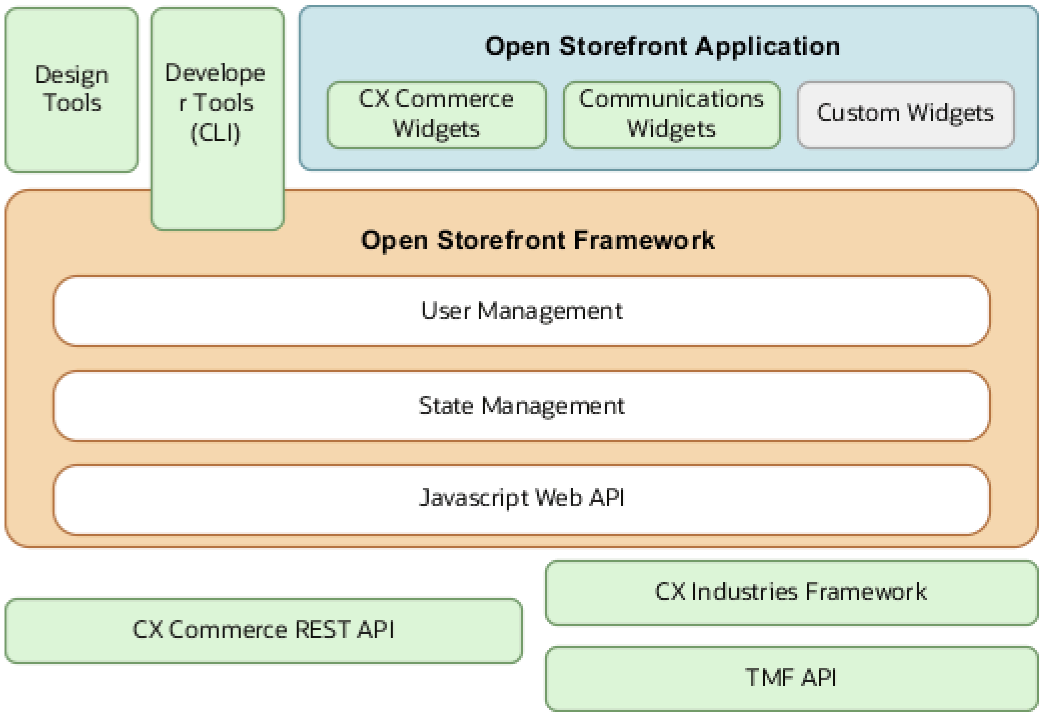
In addition to these components, use the following Communications OSF components to build storefront and self-care applications for Oracle Digital Experience for Communications:

- Communications widgets: The communications-specific widgets that you can use in conjunction with core CX Commerce widgets to build your storefront and self-care applications. Widgets work together with plug-ins and other components to help you create your website user interface.
- TMF APIs: The TMF open APIs through which the widgets make calls to the storefront server. Communications widgets call the TMF APIs through the CX Industries Framework, which routes the requests to the corresponding TMF API. Refer to the following guides on My Oracle Support at https://support.oracle.com:
  - Rest API Guide for Buying Experience for information on the TMF APIs used by storefront applications

  - Implementing CX Industries Framework for information on CX Industries Framework

- Communications registry: The read-only communications NPM registry (@oracle-dx4c-buying:registry) that you can use to get all the communications-specific open storefront packages and their dependencies for your environment. You can download these OSF packages and their dependencies as needed. You can also develop new packages and dependencies if required.

This image illustrates the communications Storefront and Self-care OSF architecture.



Related Topics

3 Get Started

Install the Reference Application

Install the Open Storefront and Self-Care reference application to develop and deliver user experiences faster.

Here's how you can install the reference application:

1. Download the Commerce Storefront and Self-Care Reference for Buying Experience file from Oracle Digital Experience for Communications Buying Experience (Doc ID 2730574.1) on My Oracle Support at https://support.oracle.com.
2. Decompress the downloaded file and open the README.
3. Follow the instructions in the README to run the installation scripts for installing and deploying the following:
    - Install Communications Server-Side Extensions.
    - Create a SAML Application in Oracle Identity Cloud Service to support single-sign on between Identity Cloud Service and Oracle Commerce Cloud.
    - Upload the key images used in the Storefront and Self-care application to Oracle Content Cloud and Experience.
    - Install the Communications Site-Settings extension and update the site setting values.
    - Update the CX Industries Framework settings to support this application. Deploy the Storefront and Self-care application on Commerce Cloud.

Install Required Software

Before you can set up your development environment, you must install certain third-party software tools.

In addition to software supplied by Oracle, a storefront development environment requires the following:

- Node.js – Install the most recent LTS (long-term support) version of Node.js 16.x on your system. (Do not install a different major version number.) Node.js 16.x is available at:
  `https://nodejs.org/dist/latest-v16.x`

  The Node.js installation also includes npm.
- Yarn – Install the latest stable version of Yarn 1.x on your system. (Do not install Yarn 2.x). Yarn 1.x is available at:
  `https://legacy.yarnpkg.com/docs/install`

After you install the software, perform the setup tasks in the sequence in which the topics are covered in this chapter.

Related Topics

Configure npm to Access Commerce and Communications Packages

You must configure the CX Commerce npm to access the CX Commerce packages hosted in the CX Commerce Open Storefront Framework registry.

To configure npm to access the @oracle-cx-commerce registry, enter the following command:

```
npm config set @oracle-cx-commerce:registry https://oracle-cx-commerce-repository.occa.ocs.oraclecloud.com
```

To configure the CX Commerce npm, see the Configure npm to access Commerce packages topic in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

Also, configure the communications npm by using the @oracle-dx4c-buying scope hosted in the @dx4c-buying registry. This registry is different from the primary npm registry and open storefront framework registry. To configure npm to access the @dx4c-buying registry, enter the following command:

```
npm config set @oracle-dx4c-buying:registry https://oracle-cx-dx4c-repository.occa.ocs.oraclecloud.com
```

Set Up a Workspace

Here's how you create a workspace from an accelerator template. You can also create a workspace from a deployed application. For information on using the deployed application, see the Set Up a Workspace topic in the Developing Open Storefront Framework Applications for Oracle CX

Create a Workspace from an Accelerator Template

You can use one of the core CX Commerce accelerator templates or the dx4c-store template for creating a workspace. dx4c-store is a more complete application that includes reference implementations of the communications widgets and other communications-specific components.

To create the workspace, enter the following command:

```
npx @oracle-dx4c-buying/cli-init create-workspace <workspace_dir> --template <template_name>
```

<workspace-dir> is the absolute or relative path to the top-level directory of the workspace. This directory must not already exist or it should be empty.

For example:

```
npx @oracle-dx4c-buying/cli-init create-workspace my-dx4c-workspace --template dx4c-store
```

This command uses the cli-init tool in the registry to run the create-workspace command. The command creates a subdirectory named my-dx4c-workspace that contains the files and directories from the dx4c-store application.

The workspace top-level directory contains a number of files and directories, including the following:

- A package.json file that designates the workspace as a private package, specifies package dependencies, and defines several scripts.
- A packages/apps directory with a child directory containing the selected application (for example, packages/apps/dx4c-store).

You can also specify a different package name and Node.js module for your application when you create a workspace by using the --appPackageName value. For example:

```
npx @oracle-dx4c-buying/cli-init create-workspace another-workspace --template dx4c-store --appPackageName
 @my-module/my-dx4c-storefront
```

This command creates a new workspace with an application named, `my-dx4c-storefront`, in a module named, `@my-module`.

Once you have created a workspace, you can create your application by modifying or replacing the application code and configuration in the workspace.

Install Package Dependencies

Install both the CX Commerce and communications open storefront framework packages and their dependencies.

To do this, go to the workspace directory and enter this command:

```
yarn install
```

This creates and populates the node_modules directory under your workspace directory. Note that the `node_modules/@oracle-dx4c-buying` subdirectory contains a symbolic link to the application directory (such as dx4c-store) in your workspace's `packages/apps` directory. For information on the warnings you see during installation, see Set Up a Workspace in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

Configure the Workspace to Access Your Commerce Server

Configure your local workspace to access a Commerce administration server environment to upload your application and then call Commerce REST API endpoints. To do this, see the Set Up a Workspace topic in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

Communications widgets access the TM Forum open APIs through CX Industries Framework to retrieve data from the server. Therefore, you must configure your local workspace to access CX Industries Framework.

To do so, you must update the following site settings in the packages/apps/dx4c-store/config/app-config/index.js file:

- `dx4cBuyingUrl`: The CX Industries Framework server URL to connect to the Buying Experience REST APIs.
- `dx4cBillingUrl`: The CX Industries Framework server URL to connect to the Billing REST APIs.

For more information on updating these settings, see the Change Site Settings topic in the Configure Application Settings Chapter in this guide.

Build the Application

To build the application in development mode, enter the following command:

```
yarn build
```

This transforms all of the application's source code into code that's suitable for use in your development environment.

> **Note:** If you're building the application just once, use Ctrl-C to exit from Rollup once you see the "waiting for changes" message.

Upload the Application to the Administration Server

To publish your application and its page layouts and components, you must deploy the application to the administration server. To do this, see the Set Up a Workspace topic in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

Run the Application

To run the application in the development mode, see the Set Up a Workspace topic in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

Now that you have set up your environment, use the copy of this template as a starting point to build your application. For the necessary steps to be performed to develop and deploy applications, see the Develop and Deploy Applications topic in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

Related Topics

Workspace Commands and Scripts

The open storefront framework workspace includes commands and scripts that you use to build, deploy, and manage the lifecycle of storefront applications. For more information, see the Workspace commands and scripts in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide.

You can also run the Storybook application for visualizing components in various configurations. For information on using Storybook, see the Use Storybook chapter within this guide.

Related Topics

- Visualize Components Using Storybook

4 Configure and Extend

Implement Single Sign-On for Your Storefront

By default, the Communications Open Storefront Framework provides Single Sign-On (SSO) access to Oracle Identify Cloud Service and the reference storefront and self-care application on Oracle Commerce Cloud.

The SSO access facilitates the use of Oracle Commerce Cloud's secure routing feature for your application. The secure routing feature prevents users from accessing URLs that they don't have access to. For example, it prevents an anonymous user trying to access the `/myAccount` URL, without first signing in. Also, allow both anonymous and signed in access to TMF APIs with the appropriate access tokens.

You can implement the default SSO for your storefront, or implement SSO using an external identity management tool. For information on implementing storefront SSO in OCC, see Related Topics.

Note that in the default SSO implementation, the communications storefront and self-care application is the service provider, while the Oracle Identity Cloud Service that provides authentication is the identity provider.

Understand the SSO Message Flow

The default SSO uses SAML (Security Assertion Markup Language) 2.0. SAML 2.0 supports a variety of different message flows for authentication and authorization.

In the default SSO process, when an anonymous user creates an account during checkout, an account is automatically created in Oracle Identify Cloud Service. Once the user completes the order, Oracle Identify Cloud Service sends an email to the user requesting to activate the user's account. Once the account is activated, the user can sign in using storefront SSO and access the account information.

Here's the SSO message flow that describes the approach for the communications storefront and self-care application. It shows the flow of messages when a shopper sign-in using SSO.

1. When the user clicks Sign In, the ssoLogin action is triggered.
2. The ssoLogin action calls the /ccstore/v1/samlAuthnRequest?encode=true endpoint to get the following:
    - `authnRequestTarget`: The identify provider SSO URL. For example, https://<your_IDCS_Instance>.com/fed/v1/idp/sso
    - `authnRequest`: The SAML 2.0 request conforming to the authentication request protocol.
3. The ssoLogin submits an XHTML form to the identify provider SSO URL (which is `authnRequestTarget`) containing a SAMLRequest, which is authnRequest, and the RelayState , which is `/myAccount`.
4. The browser redirects the user to the identify provider login page.
5. After successful login, identify provider redirects a POST request to the Assertion Consumer URL (https://<cc-store>/SAML/post) with a samlResponse and the RelayState as provided in step 3.
6. The myAccount page detects the samlResponse and triggers the OCC login action to generate an access token. In the reference application, you must manually generate the access token by using endpoints. For more information on generating access tokens, see the Use Server-Side Extensions chapter in this guide.
7. The access token is stored in the browser local storage and cookie. It expires in 15 minutes.

However, there's a limitation in this SSO implementation. The sign-out doesn't log the user out of the identity provider. For more information on this limitation, see the Understand storefront SSO limitations topic in the Extending Oracle Commerce Cloud guide.

Related Topics

Change Site Settings

When you develop the application locally, you might need to change some of the following required site settings:

- `dx4cBuyingUrl`: The CX Industries Framework server URL to connect to the Buying Experience REST APIs.
- `dx4cBillingUrl`: The CX Industries Framework server URL to connect to the Billing REST APIs.
- `dx4cSSEUrl`: The URL to connect to the Oracle Identity Cloud Service Server-Side Extension (dx4c-idcs-sse). Leave it blank to connect to the default Server-Side Extension.
- `dx4cGeoLocationSSEUrl`: The URL to connect to the geographic location server-side extension. Leave it blank to connect to the default geographic location server-side extension.
- `dx4cPaymentMSUrl`: The URL to connect to your payment gateway. If left blank, the storefront payment components aren't visible to the users and the payment endpoints aren't called.
- `dx4cSupportNumber`: The support number you want to display to users when an error occurs.
- `dx4cDefaultPriceListName`: The default price list name to use for communications products. The default value is DX4C NA Pricelist.
- `dx4cOdaEnabled`: The parameter to enable or disable Oracle Digital Assistant. By default, this value is set to false (disabled).
- `dx4cOdaUrl`: The URL for the Digital Assistant instance.
- `dx4cOdaChannelId`: The channel ID for the Digital Assistant instance.
- `dx4cEligibilityProfileCountry`: The default country to be used if the customer's location can't be detected, or the customer's location isn't part of the list of supported countries.
- `dx4cEligibilityProfileZipCode`: The default ZIP code to be used if the customer's ZIP code can't be detected, or the customer's location isn't part of the list of supported countries.
- `dx4cSupportedLocations`: A comma-separated list of countries that are used to validate the automatically detected eligibility profile country.
- `dx4cOCEUrl`: The URL for the Oracle Content and Experience Cloud instance.
- `dx4cOCEChannelToken`: The token to access the content published on the Oracle Content and Experience Cloud channel.

These settings are configurable parameters that are globally accessible to the communications storefront and self-care components. Review all the settings to verify that it has appropriate values.

Here's how you can update the settings in the Oracle Commerce Cloud administration console:

1. In the OCC administration console, navigate to **Settings** > **Extension Settings** > **DX4C Store Site Settings**.
2. Update the site settings as needed.
3. Click **Save** and then **Publish**.

You can also update the settings in the *packages/apps/dx4c-store/.occ/config.js* file manually. Here's a sample site setting configuration:

```
module.exports = {

 "renderToString": true,

 "appContext": {

 "default": {

 "configRepositoryState": {

 "dx4cConfig": {

 "dx4cBuyingUrl": "http://<Industries_Framework_server_url>/api/01",

 "dx4cBillingUrl": "http://<Industries_Framework_server_url>/api/01",

 "dx4cSSEUrl": "",

 "dx4cGeoLocationSSEUrl": "",

 "dx4cPaymentMSUrl": "",

 "dx4cOceUrl": "https://<oce-instance>",

 "dx4cOceChannelToken": "<oce-channel-token>",

 "dx4cDefaultPriceListName": "DX4C NA Pricelist",

 "dx4cOdaEnabled": false,

 "dx4cOdaUrl": "",

 "dx4cOdaChannelId": "",

 "dx4cEligibilityProfileCountry": "US",

 "dx4cEligibilityProfileZipCode": "",

 "dx4cSupportedLocations": "US,CA"

 }

 }

 },

 "test": {

 "configRepositoryState": {

 "dx4cConfig": {

 "dx4cBuyingUrl": "http://<Industries_Framework_server_url>/api/01",

 "dx4cBillingUrl": "http://<Industries_Framework_server_url>/api/01",

 "dx4cSSEUrl": "",

 "dx4cGeoLocationSSEUrl": "",

 "dx4cPaymentMSUrl": "",

 "dx4cOceUrl": "https://<oce-instance>",

 "dx4cOceChannelToken": "<oce-channel-token>",
 "dx4cDefaultPriceListName": "DX4C NA Pricelist",

 "dx4cOdaEnabled": false,

 "dx4cOdaUrl": "",

 "dx4cOdaChannelId": "",

 "dx4cEligibilityProfileCountry": "US",

 "dx4cEligibilityProfileZipCode": "",

 "dx4cSupportedLocations": "US,CA"

 }

 }

 }

 }
```

When you start the application using the command *yarn start*, the application uses the configuration values specified in the site settings. If you want to override these values, you can append the *--appContext* parameter to the *yarn start* command. For example, if you want to use the values for the test configuration as shown in the sample, you have to run the following command:

```
yarn start --appContext test
```

Note: You can override the configuration only for local development applications. When the application is running on a deployed server, it always uses the values configured in the communications store site settings extension in the Oracle CX Commerce admin console.

When you create widgets for the application, you want to configure the settings for it to display your site. To add new site settings for your widgets, refer to the Add Site Settings topic in the Oracle CX Commerce Developing Widgets guide.

How to Set Channel Token

You must set the DX4C OCE channel token as a post deployment step for retrieving Oracle Content and Experience Cloud images. Review and follow steps available in the Channel Token and Authentication chapter of the REST API for Content Delivery guide.

1. Run GET request: https://Content Management URL/content/management/api/v1.1/channels

   In the response body, user will get the channel ID, for example: RCHANNEL0E38831154DF4D848CED28979824F2D4
2. Run GET request: https://Content Management URL/content/management/api/v1.1/channels/<ID>, for example: https://Content Management URL/content/management/api/v1.1/channels/RCHANNEL0E38831154DF4D848CED28979824F2D4

   In the response body, user will get "channelTokens.token", for example: "efe506cee46d451bb31d1c36e12e5b71"
3. In the OCC administration console, navigate to **Settings > Extension Settings > DX4C Store Site Settings**.
4. Enter the DX4C OCE channel token and save it.

Related Topics

Configure Custom Models

You use Buying and Billing REST APIs to retrieve data for displaying it in the UI.

The API model doesn't always suit the requirements of the UI and sometimes it can be difficult to efficiently extract the required information from large API responses. To avoid this, you can configure a custom model to assist in extracting only the required information from an API response by using the app-config/index file. You can configure the settings in the packages/apps/dx4c-store/config/app-config/index.js file manually.

Here's a sample app-config/index.js file for extracting data from the Buying and Billing API responses:

```
{
 phoneCategoryName: 'Smart Phone',
 productCharacteristics: ['Color', 'Capacity'],
 characteristicMetaData: {
 Color: {
 control: 'ColorPicker',
 color: true
 },
 Capacity: {
 control: 'ProductOption',
 sort: true
 }
 },
 productLineName: 'Mobile Phones',
 planMetaData: {
 'Supremo Byod': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Unlimited Line', 'Wireless Unlimited Voice Service', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Basic': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Premium': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Kids Text and Data': {
 phoneRequired: true,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Text & Data', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Lite': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Zoom': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Unlimited': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Kids': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Gold': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Mobile Bundle', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Silver': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Mobile Bundle', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo Bronze': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Mobile Bundle', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo 5G Premium': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 },
 'Supremo 5G Lite': {
 phoneRequired: false,
 simCardName: 'SIM Card',
 planToServiceHierarchy: ['Wireless Line', 'SIM Card'],
 defaultUsageType: 'Data'
 }
 }
 }
```

Let's see how you can use the following fields in the sample app-config/index.js file to determine the data to be retrieved from the API response:

phoneCategoryName

The category name of a phone or device. Use this as a query parameter to filter the phones or devices when calling the Get all product offers Buying API.

For example, when you set this field to smartphone as shown in the sample file, the query for the Get all product offers API is set as productCatalogManagement/v4/productOffering?category.name=Smart%20Phone. This ensures that only smartphone offers are returned in the API's response.

Similarly, you can use this to find the compatible devices for a given plan.

productLineName

The name of the Product Line. Used to identify the product line which defines the compatible devices for a plan.

productCharacteristics

The properties of the product. Use this to transform the response from the Get a product Buying API. By setting this parameter, you can filter out any product characteristics that aren't required to be displayed on the UI.

Typically, products include a number of characteristics. Here's a sample API response file with six product characteristics. However, when you set this field to Color and Capacity as shown in the sample file, only those product characteristics are retrieved to be displayed in the UI. All the other characteristics are hidden.

```
"prodSpecCharValueUse": [
 {
 "@type": "ProductSpecificationCharacteristicValueUseOracle",
 "name": "Model",
 "valueType": "string",
 "isConfigurable": true,
 "productSpecCharacteristicValue": [...],
 "@baseType": "ProductSpecificationCharacteristicValueUse",
 "@schemaLocation": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/describe"
 },
 {
 "@type": "ProductSpecificationCharacteristicValueUseOracle",
 "maxCardinality": 1,
 "minCardinality": 1,
 "name": "Color",
 "valueType": "string",
 "isConfigurable": true,
 "productSpecCharacteristicValue": [...]
 "@baseType": "ProductSpecificationCharacteristicValueUse",
 "@schemaLocation": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/describe"
 },
 {
 "@type": "ProductSpecificationCharacteristicValueUseOracle",
 "maxCardinality": 1,
 "minCardinality": 1,
 "name": "Capacity",
 "valueType": "number",
 "isConfigurable": true,
 "productSpecCharacteristicValue": [...],
 "@baseType": "ProductSpecificationCharacteristicValueUse",
 "@schemaLocation": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/describe"
 },
 {
 "@type": "ProductSpecificationCharacteristicValueUseOracle",
 "maxCardinality": 1,
 "minCardinality": 1,
 "name": "IMEI",
 "valueType": "string",
 "isConfigurable": true,
 "productSpecCharacteristicValue": [...],
 "@baseType": "ProductSpecificationCharacteristicValueUse",
 "@schemaLocation": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/describe"
 },
 {
 "@type": "ProductSpecificationCharacteristicValueUseOracle",
 "name": "Supported 3rd Party Apps",
 "isConfigurable": true,
 "productSpecCharacteristicValue": [...],
 "@baseType": "ProductSpecificationCharacteristicValueUse",
 "@schemaLocation": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/describe"
 },
 {
 "@type": "ProductSpecificationCharacteristicValueUseOracle",
 "description": "Brand",
 "name": "Brand",
 "valueType": "string",
 "isConfigurable": false,
 "productSpecCharacteristicValue": [...],
 "@baseType": "ProductSpecificationCharacteristicValueUse",
 "@schemaLocation": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/describe"
 }
]
```

characteristicMetaData

Each characteristic in the application is modeled using this field. Use this to map the product characteristics to an individual UI widget.

Also, you can use it to specify whether the options for that characteristic need to be sorted to ensure consistent display in the UI. Based on the values shown in the sample file, the Color characteristic is displayed using the ColorPicker component and the Capacity characteristic is displayed using the ProductOption control in the UI.

planMetaData

Each plan in the application is modeled using this field. The sample file contains the following metadata:

- phoneRequired: This value determines whether the delete icon should be displayed for the cart items associated with the given plan. If the phoneRequired value is set to true, the delete icon isn't displayed for that cart item in the plan.
- simCardName: The name of the SIM Card. Used to identify the SIM Card in the overall service hierarchy.
- planToServiceHierarchy: An array which lists the various services that are bundled with the plan. The various services listed in the sample file form a hierarchical path that ends with SIM Card. With this, it's easier for you to locate the SIM Card service and parent services.
- defaultUsageType: This value determines the usage type to display it in the Service Summary component.

For example, let's consider the following as the bundledProductOffering structure for the Supremo Basic plan:

```
    "bundledProductOffering": [
     {
     "id": "88-1WZKQ6",
     "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZKQ6",
     "name": "Wireless Lite Voice Service",
     "bundledProductOfferingOption": {
     "numberRelOfferDefault": 1,
     "numberRelOfferLowerLimit": 1,
     "numberRelOfferUpperLimit": 1
     }
     },
     {
     "id": "88-1WZGJQ",
     "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZGJQ",
     "name": "Wireless Text Service",
     "bundledProductOfferingOption": {
     "numberRelOfferDefault": 1,
     "numberRelOfferLowerLimit": 1,
     "numberRelOfferUpperLimit": 1
```

```
      }
      },
      {
      "id": "88-1WZGDM",
      "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZGDM",
      "name": "Wireless Lite Data Service",
      "bundledProductOfferingOption": {
      "numberRelOfferDefault": 0,
      "numberRelOfferLowerLimit": 0,
      "numberRelOfferUpperLimit": 1
      }
      },
      {
      "id": "88-1WZCBE",
      "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZCBE",
      "name": "X-Hot Spot",
      "bundledProductOfferingOption": {
      "numberRelOfferDefault": 0,
      "numberRelOfferLowerLimit": 0,
      "numberRelOfferUpperLimit": 1
      }
      },
      {
      "id": "88-1WZFOQ",
      "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZFOQ",
      "name": "Amazon Prime Video",
      "bundledProductOfferingOption": {
      "numberRelOfferDefault": 0,
      "numberRelOfferLowerLimit": 0,
      "numberRelOfferUpperLimit": 1
      }
      }
     ]
```

In the sample file, the value for the planToServiceHierarchy for the Supremo Basic plan is ["Wireless Lite Voice Service", "SIM Card"]. If you browse through the productOffering hierarchy, you can find that the bundledProductOffering for the Wireless Lite Voice Service is within this structure, where SIM Card is the last entry in the hierarchy. The bundledProductOffering section looks similar to the following:

```
     "bundledProductOffering": [
      {
      "id": "88-1WZF81",
      "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZF81",
      "name": "5G Lite Voice Service",
      "bundledProductOfferingOption": {
      "numberRelOfferDefault": 1,
      "numberRelOfferLowerLimit": 1,
      "numberRelOfferUpperLimit": 1
      }
      },
      {
      "id": "88-1WZDMX",
      "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZDMX",
      "name": "Wireless Voice Features",
      "bundledProductOfferingOption": {
      "numberRelOfferDefault": 1,
      "numberRelOfferLowerLimit": 1,
      "numberRelOfferUpperLimit": 1
      }
      },
      {
      "id": "88-1WZDLM",
      "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZDLM",
      "name": "SIM Card",
      "bundledProductOfferingOption": {
```

```
     "numberRelOfferDefault": 1,
     "numberRelOfferLowerLimit": 1,
     "numberRelOfferUpperLimit": 1
     }
     },
     {
     "id": "88-1WZDKI",
     "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZDKI",
     "name": "National Roaming",
     "bundledProductOfferingOption": {
     "numberRelOfferDefault": 1,
     "numberRelOfferLowerLimit": 1,
     "numberRelOfferUpperLimit": 1
     }
     },
     {
     "id": "88-1WZDJF",
     "href": "https://<SERVER-INFO>/productCatalogManagement/v4/productOffering/88-1WZDJF",
     "name": "International Roaming",
     "bundledProductOfferingOption": {
     "numberRelOfferDefault": 0,
     "numberRelOfferLowerLimit": 0,
     "numberRelOfferUpperLimit": 1
     }
     }
    ]
```

Set Up OCE

You use Oracle Content Cloud and Experience (OCE) to host the images for your storefront and self-care application.

By default, a new repository named DX4C Media is created in OCE at the time of provisioning to group and categorize the images. You can also create your own repository if required. A new channel named DX4C Channel is also created for you to upload the images. These channels are basically logical entities in OCE to which your content or images are published. All images used in the storefront are published to this channel with the asset type as DX4C-Asset.

For more information on how to upload images or content to OCE, refer to the Oracle Content Management documentation.

Retrieve Content from OCE

Once an image has been published to OCE, it can be retrieved for display in your application by using the image slug.

Here's how you can set up your application to retrieve content from OCE:

1. Navigate to the Assets menu in OCE.
2. Select the appropriate repository (for example, DX4C Media).

   The images are displayed.
3. Right-click on an image and select **Properties**.

   The slug is displayed under Friendly URL Slug.

4. Enter the slug as a parameter in the @oracle-dx4-buying/common/utils/getOCEImageUrl utility function to generate the URL for an image published to OCE.

Here's the source code of the @oracle-dx4-buying/common/utils/getOCEImageUrl function:

```
export const getOCEImageUrl = (slug, state) => {
 const dx4cOceUrl = getStoreSetting(DX4C_OCE_URL, state);
 const dx4cOceChannelToken = getStoreSetting(DX4C_OCE_CHANNEL_TOKEN, state);

 return `${dx4cOceUrl}/content/published/api/v1.1/assets/.by.slug/${slug}/native?channelToken=
${dx4cOceChannelToken}`;
};
```

The @oracle-dx4-buying/common/utils/getOCEImageUrl function uses the `dx4cOCEUrl` and `dx4cOCEChannelToken` values as well as the slug of the respective image to generate the URL of the image.

You can also use the @oracle-dx4c-buying/components/call-to-action and @oracle-dx4c-buying/components/call-to-action-image components to display images from OCE by passing the oceImageSlug as a property to these components. The application's Home page contains a number of examples of the Call To Action component displaying images from OCE.

Enable Digital Assistant

The communications storefront and self-care application is preintegrated with Oracle Digital Assistant. But, it is disabled by default. You can enable it based on your business needs.

Here's how you can enable Digital Assistant:

1. In the OCC Admin console, navigate to **Settings** > **Extension Settings** > **DX4C Store Site Settings**.
2. Update the following site settings:
   - Oracle Digital Assistant Enabled: Select to enable Digital Assistant.
   - Oracle Digital Assistant Instance URL: Enter the URL for the Digital Assistant instance.
   - Oracle Digital Assistant Channel ID: Enter the channel ID for the Digital Assistant instance.
3. Click **Save** and then **Publish**.

Once you update the site settings, the Digital Assistant chat widget appears in the application when a user is signed in.

You can also use the useDigialAssistant hook in the **@oracle-dx4c-buying/common/hooks/hook.js** file to enable Digital Assistant. By default, this hook also displays the chat widget only if the user is signed in.

To modify when the ODA chat widget is displayed, use the following functions from **@oracle-dx4c-buying/common/digital-assistant/index.js**:

- initializeODA: Dynamically imports the Digital Assistant SDK script located in <workspace>//occ-public/web-sdk.js and then displays the chat widget in the UI.
- closeODA: Closes the SDK instance and removes the Digital Assistant chat widget from the UI.
- isODAEnabled: Specifies whether the Digital Assistant SDK has been initialized or not.

For more information, see the Oracle Digital Assistant documentation.

Use Communications Server-Side Extensions to Access Buying and Billing APIs

In addition to using REST APIs for integration with external systems, you can use server-side extensions written in JavaScript in your communications storefront and self-care application based on your business scenarios. These extensions let you run critical code in a secure environment that can't

The communications storefront and self-care application uses an Oracle Commerce Cloud server-side extension to communicate with Oracle Identity Cloud Service. All the endpoints in the application use this server-side extension to generate the appropriate access tokens for accessing the Buying and Billing APIs. You can use these endpoints to generate access tokens based on your business needs.

CAUTION: Don't modify this server-side extension.

You can use this server-side extension in your application for generating access tokens through these endpoints:

- `/v1/getAccessToken`: The getAccessToken endpoint generates the access tokens for anonymous access. With this token, only the following endpoints can access the TMF APIs:
    - `dx4cRetrieveProductOffering`
    - `dx4cListProductOfferings`
    - `dx4cRetrieveProductOfferingPrice`
    - `dx4cCreateShoppingCart`
    - `dx4cRetrieveShoppingCart`
    - `dx4cPatchShoppingCart`
    - `dx4cCreateIndividual`
    - `dx4cRetrieveProductLine`
    - `dx4cRetrieveCatalogs`
    - `dx4cRetrieveCategory`
    - `dx4cListCategory`
- `/v1/getSubscriberAccessToken`: The endpoint to generate the access tokens for signed-in users. You can access all the Buying and Billing APIs using this token.
- `/v1/getLogoutRequest`: The endpoint to generate a SAML sign out request. The payload must contain a user name. You can use an optional boolean query parameter, Encode, with this endpoint. If this is set to true, the SAML sign out request that's returned will be base64 encoded.

You can also create server-side extensions for your application based on your business needs. For more information on developing new server-side extensions, refer to the Develop Server-Side Extensions topic in the Extending Oracle CX Commerce guide.

Related Topics

Use Communications Server-Side Extensions to Detect Customer Location

The communications storefront and self-care application uses a server-side extension to automatically detect the customer location. This server side extension leverages Oracle Commerce Cloud Integrated Edge Caching (CDN) to detect a customer location based on the customer's IP address.

Here's the endpoint to detect the customer location and its sample response:

`/v1/getGeoLocation`

```
    {
     "georegion": "155",
     "country_code": "NL",
     "city": "AMSTERDAM",
     "lat": "52.35",
     "long": "4.92",
     "timezone": "GMT+1",
     "continent": "EU",
     "throughput": "vhigh",
     "bw": "5000",
     "asnum": [
     "206209"
     ],
     "location_id": "0"
    }
```

5 Design Storefront and Self-Care Pages

Work with Default Communications Widgets

You can use the default layouts and widgets in the Communications Open Storefront Framework to build your store. Layouts represent different page types, such as phones, plans, cart, checkout, and order confirmation.

You can download the layouts from the server environment and modify them to suit your needs before deploying them to the live environment. This includes the widgets that are used in each of these layouts. You can do this using the Design page in the Oracle CX Commerce administration interface. This section assumes you have some experience working with layouts and widgets in the CX Commerce administration interface.

Let's see the default layouts and the widgets in Communications Open Storefront Framework. You can find these components in the @oracle-dx4c-buying/components directory. You can also run Storybook in your workspace for a quick visual reference of these components. For more information on Storybook, see the Use Storybook chapter in this guide.

Home Page Widgets

The reference template includes the following widgets for your store's home page:

- Header
- Footer
- Notification
- Container
- Call To Action, Call to Action Content and Call to Action Image
- Logo
- Cart Icon
- Menu
- Mobile Menu
- Sign In
- Main

Phone Listing Widgets

The reference template includes the following widgets for displaying the phones in the store:

- Title
- Breadcrumbs
- Products Container
- Product Grid
- Sort By

- Product Card

Product Details Page Widgets

The reference template includes the following widgets for a product's details page:

- Title
- Breadcrumbs
- Products Details Container
- Product Image
- Product Description
- Product Options
- Product Price
- Product Details Continue Button

Plan Listing and Manage Plan Widgets

The reference template includes the following widgets for displaying the plans in the store:

- Title
- Breadcrumbs
- Plans Container
- Plan Filters
- Plan Filter Option
- Plans
- Plan
- Plan Details
- Plan Features

The reference template includes the following widgets for customers to manage their plans:

- Title
- Breadcrumbs
- Manage Plan Container
- Manage Plan Actions
- My Plan Details
- Add-Ons Container
- Add-Ons
- Add-On Card
- Shop Add-Ons
- Suspend Plan Dialog

- Resume Plan Dialog
- Terminate Plan Dialog
- Switch Plan Train (including Switch Plan Review and Switch Plan Order Summary)

Cart and Checkout Widgets

The reference template includes the following widgets for the cart:

- Title
- Breadcrumbs
- Cart Container
- Cart Buttons
- Cart
- Cart Group
- Cart Item
- Cart Item Actions
- Cart Summary
- Add Change Phone Button
- Empty Cart

The reference template includes the following widgets for the checkout process:

- Title
- Breadcrumbs
- My Account Container
- Checkout Train
- Account including Account Form and Account Summary
- Address including Address Form and Delivery Options
- Payment Method
- Order Summary
- Order

Account Widgets

The reference template includes the following widgets for a customer to view details about their services, billing, and usage information:

- My Account Container
- Bill Unit
- Bill Details
- Service Summary
- My Usage

- Usage

Profile Widgets

The reference template includes the following widgets for the customers to view and update their profile:

- Edit Profile which includes Account Form, Address and Payment Method
- Profile Contact

Note that the Address, Address Form, and Payment Method widgets used in both Profile and Checkout flows are the same.

For more information on working with the widgets using the Design page in the administration interface, see Related Topics.

> **Note:** The Communications Open Storefront Framework uses the Buying APIs to retrieve the display color for products from the catalogs configured in Launch Experience. Hence, the Configure and display color swatches topic in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide isn't applicable for Communications Open Storefront Framework.

Related Topics

Build a Communications Widget

Use widgets to create the user interface for your application. You can extend or modify the default CX Commerce and Communications widgets, or create new widgets based on your business scenarios. For example, add a widget to display a list of plans for your shoppers.

For an overview of widgets and the necessary steps to be followed for building a widget, refer to the Create Custom Widgets chapter in the Developing Open Storefront Framework Applications for Oracle CX Commerce guide. All the steps in that chapter are applicable for building a widget for the communications storefront application. However, note the following:

- The communications storefront application don't implement fetchers or subscribers. If required, you can create a subscriber or fetcher based on your business scenarios.
- Most of the communications widgets use Buying and Billing API endpoints to retrieve data for the UI. You don't create an endpoint using CX Commerce REST API functions. Here's an example that describes how to build a custom widget for a communications storefront and self-care application.

Let's assume that you're creating custom widgets to display a list of plans on your storefront. You build two widgets, Plans widget and Plan widget, one to display the plans group and the other one to display specific plans.

To build the widgets, you must perform the following steps:

- Create the directory structure needed for your new widget.
- Create the initial widget files. These files are later modified as you continue to add other components.
- Add the widget to your application.
- Identify and add your widget to your page layout.

- Create and generate the CSS style sheet for your widget.
- Display localized strings, if necessary.

Create a Widget Directory Structure

You must build the widgets in the /plugins/components/plans directory. You can create these files and directories manually, or use the create-widget script in the package.json file to create them using templates. Remember that if you copy and modify an existing widget, the structure still remains the same.

Here's the sample widget directory structure for your Plans widget:

```
/plugins
 /components
 /plans
 index.js
 meta.js
 styles.css
 /locales
 de.js
 en.js
```

Create Widget Files

Widgets contain an index.js file and a meta.js file. Note that these are different than the index.js and meta.js files that reside in the /plugins directory.

You must create an initial index.js and a meta.js file for your widget and add them to the components directory that you just created.

Here's the sample /plugins/components/plans/index.js file for creating the Plans widget. This file defines that widget and it's available in the @oracle-dx4c-buying/components/plans directory.

```
import React from 'react';
import Styled from '@oracle-cx-commerce/react-components/styled';
import Plan from '../plan';
import css from './styles.css';


function Plans(props) {
 //Promotion Offering deliberately hardcoded just for illustrative purposes
 const promotionOffering = [
 {
 id: 'plan1',
 name: 'Supremo Kids',
 description: 'Manage screen time, filter contents and track location on your kids phone and get peace of
 mind.',
 price: '$45.99',
 marketingFeature: [
 {name: 'Unlimited voice and text, nationwide up to 20 managed contacts'},
 {name: 'Unlimited voice and text, nationwide up to 20 managed contacts'},
 {name: 'Hellicopter parent, enable this feature to watch their every online action'},
 {name: 'Punishment time blocks anything fun'}
 ]
 },
 {
 id: 'plan2',
 name: 'Supremo Zoom',
 description: 'Best for super-fast downloads and UHD streaming, plus everything Unlmited Max can do.',
 price: '$65.99',
 marketingFeature: [
 {name: 'Unlimited voice and text, nationwide up to 20 managed contacts'},
 {name: 'Free calls to Canada and Mexico'}
 ]
 }
 ];

 return (
 <Styled id="Plans" css={css}>
 <div>
 <div className="plans-container">
 {promotionOffering &&
 promotionOffering.map(promotion => {
 return (
 <div key={`promotion_${promotion.id}`}>
 <Plan promotion={promotion} {...props} />
 </div>
 );
 })}
 </div>
 </div>
 </Styled>
 );
}

export default Plans;
```

Here's the sample index.js file for creating the Plan widget, which is a child widget of the Plans widget. This widget also follows the same directory structure as the Plans widget.

```
import React, {useContext} from 'react';
import {StoreContext} from '@oracle-cx-commerce/react-ui/contexts';
import Styled from '@oracle-cx-commerce/react-components/styled';
import Button from '@oracle-dx4c-buying/components/button';
import css from './styles.css';

function Plan(props) {
 const {action} = useContext(StoreContext);
 const {promotion, subTextLabel, choosePlanLabel} = props;
 const {name, description, marketingFeature, price} = promotion;
 const checkMark = '\u2714';

 function handleChoosePlan() {
 action('_setPlan', promotion);
 }

 return (
 <Styled id="Plan" css={css}>
 <div className="plan-container">
 <div className="plan-title-container">
 <div className="plan-name">{name}</div>
 <div className="plan-price">
 <span className="plan-price-span">
 <span className="plan-price-amount">{price}</span>
 <div className="plan-sub-text">{subTextLabel ? subTextLabel : '\u00A0'}</div>
 </span>
 </div>
 </div>
 <p className="plan-description">{description}</p>
 <div className="plan-info-container">
 {marketingFeature &&
 marketingFeature.map(feature => {
 return (
 <p key={feature} className="plan-feature-info">
 <span className="plan-check-mark">{checkMark}</span>
 {feature.name}
 </p>
 );
 })}
 </div>
 <div className="plan-button">
 <Button
 label={choosePlanLabel}
 size="md"
 chroming="solid"
 className="dx4c-brand-button"
 onClick={() => {
 handleChoosePlan();
 }}
 />
 </div>
 </div>
 </Styled>
 );
}

export default Plan;
```

Here's the sample /plugins/components/plans/meta.js file to create a metadata object for the Plans widget:

```
/**
* Metadata for the plans component.
*/
export default {};
```

Add Your Widget to Your Application

Now that you have created the index.js and meta.js files for your widget, update both the files to add your widgets to the application. To do so, add the following reference to the /plugins/components/index.js file:

```
// Export a reference to our _Plans widget.
export const _Plans = () => import('./plans');
```

Now create a parallel metadata object by adding the following reference in the /plugins/components/meta.js file:

```
export {default as _Plans} from './plans/meta';
```

Note: Preceding the names of your new components with an underscore is the recommended naming convention for custom widgets, as it will prevent naming collisions with default widgets. For example, _Plans.

Identify the Layout

You must create a JSON file (plans.json) under **assets < pages** to set the layout page for your widget. Here's the sample plans.json file to set the layout page for the Plans widget:

```
{
 "title": Plan Page",
 "address": "/plan",
 "shortName": "plans",
 "defaultPage": true,
 "layout": [
 {
 "type": "main",
 "width": 12,
 "components": [
 "_Plans"
 ]
 }
 ]
}
```

Note that the _Plans has been added as a component. To test the custom page locally, use the cx-layout query parameter `localhost?cx-layout=plan`.

You initially create your layouts locally and then upload them to the Design page of the administration interface. For uploading the layouts using CX Commerce, see the Design Your Store Layout chapter in the Using Oracle CX Commerce guide.

Set the Widget Style Sheet

If you create or use any CSS that's specific to a user interface component, you must store that CSS in the widget styles.css file.

For creating the Plans widget, you use the CSS style sheet defined in the /plugins/components/plans/styles.css file. It contains the standard CSS styling.

Here's the sample CSS for the Plans widget:

```
.plans-container {
 margin-top: 3rem;
 display: grid;
 grid-template-columns: repeat(auto-fit, minmax(0, 27rem));
 grid-column-gap: 1.5rem;
 grid-row-gap: 1.5rem;
 width: auto;
}
```

Once you create the CSS, update the Plans widget to use the CSS by doing the following:

1. Import the styles.css file into the component. To do this, enter the following command:

   ```
   import css from './styles.css';
   ```

2. Wrap the React element for the widget in a Styled component, with an ID property named after the widget (in this example, Plans) and a CSS property that contains the imported CSS. This adds the CSS to an HTML style element before rendering the widget. For example:

   ```
   <Styled id="Plan" css={css}>
    //Widget implementation goes here...
   </Styled>
   ```

3. Use class names from the CSS by setting the React className property on individual React elements. For example:

   ```
   <div className="plans-container">
   </div>
   ```

Repeat a similar process to add the styles.css Plan component. Here's the sample CSS for the Plan widget:

```
.plan-container {
 margin-bottom: 2rem;
 text-align: left;
 width: auto;
 }
 .plan-title-container {
 background-color: var(--dx4c-light-yellow-background-color);
 height: 10.8rem;
 padding-top: 2rem;
 padding-right: 2rem;
 padding-left: 2rem;
 }
```

```
.plan-name {
font-size: var(--dx4c-typography-heading-sm-font-size);
font-weight: var(--dx4c-typography-heading-sm-font-weight);
line-height: var(--dx4c-typography-heading-sm-line-height);
height: 4.6rem;
}

.plan-sub-text {
color: var(--dx4c-secondary-text-color);
}

.plan-price {
display: inline-flex;
width: 100%;
}

.plan-price-span {
width: 50%;
}

.plan-price-amount {
font-size: var(--dx4c-typography-heading-xl-font-size);
font-weight: var(--dx4c-typography-heading-xl-font-weight);
line-height: var(--dx4c-typography-heading-xl-line-height);
}

.plan-button {
margin-left: 2rem;
margin-top: 2rem;
}

.plan-info-container {
height: 17.9rem;
overflow: hidden;
padding: 1rem 2rem 0rem 2rem;
}

.plan-feature-info {
margin-block-end: 1rem;
font-size: var(--dx4c-typography-body-md-font-size);
line-height: var(--dx4c-typography-body-md-line-height);
}

.plan-description {
width: auto;
height: 5.25rem;
padding: 2rem 2rem 0rem 2rem;
display: -webkit-box;
-webkit-line-clamp: 4 !important;
-webkit-box-orient: vertical;
overflow: hidden;
font-size: var(--dx4c-typography-body-md-font-size);
line-height: var(--dx4c-typography-body-md-line-height);
}

.plan-check-mark {
margin-right: 0.625rem;
width: 1.1rem;
height: 0.72rem;
}
```

Create Locale Information

You need to add any component-specific locale strings to the /locales directory under the component directory for your widget, which is /plugins/components/plan/locales in this example. Note that any shared locale strings should be put into this common resource directory.

The locales directory should contain resource bundles for all the supported locales. These resource bundles contain strings. For example, the English locale, or the en.json file, contains the following strings for the Plans widget:

```
{
 "subTextLabel": "Per Month",
 "choosePlanLabel ": "Choose Plan"
}
```

The widget obtains localized text from the resource bundle that identifies the language of the shopper's browser. All these resources are passed into widgets as normal props (properties). You can add the Label suffix as a naming convention to distinguish these props from other props.

For example, the German locale, or the /plugins/components/plans/locales/de.json file, contains the following labels for the Plans widget:

```
{
 "subTextLabel": "[de]Per Month[de]",
 "choosePlanLabel ": "[de]Choose Plan[de]"
}
```

To pass the resources in as component props, replace the meta.js file for the widget with the following:

```
import de from './locales/de.json';
import en from './locales/en.json';

/**
 * Metadata for the Plans component.
 */
export default {
 // Include references to all of our resource strings in all supported locales.
 // This will enable the component to access any resource string via its props,
 // using the locale that is currently in effect.
 resources: {
 de,
 en
 }
};
```

Once you add the resources to the appropriate resource bundle file and export resources by updating the meta.js file, the subTextLabel and choosePlanLabel appear as part of the promotionOffering props object in the /plugins/components/plans/index.js file:

These props are also passed to the Plan widget as it's a child widget. Now you can see in the index.js file for your widget that you extract the two resources bundles from the promotionOffering props object and use those labels, subTextLabel and choosePlanLabel, in the Plan widget.

Note that you must redeploy your application for localization to take effect as the localization mechanism requires the Design Studio text snippets to be created to contain the translated strings.

Now that you have created your widget, you can configure it to communicate with the TM Forum API. You can also perform the following steps if the predefined resources don't match your requirement:

- Add a new selector to the widget to access the data.
- Write an action to access an endpoint.

- Create an endpoint.

For more information, see Related Topics.

Related Topics

- Create a Selector
- Write an Action
- Create an Endpoint

Configure a Widget to Use Buying and Billing APIs

After you build a widget, you update your application to communicate with the Buying and Billing APIs. To do so, you must determine the location from which the data should be retrieved.

In the plans widget example, assume that the `promotionOffering` data is in the Redux store. To get the data from the Redux data store into your Plans UI component, you export a connect component. For example:

```
export default connect(getPromotionOffering)(Plans);
```

This component acts as wrapper around your UI component and binds your component to the specific state located in the Redux store. It supplies the plan information as props to your UI component.

You then pass the `getPromotionOffering` selector to connect the Plans widget to the specific state in the Redux store. Once you connect your widget, any changes to the props or the Redux state results in rendering of the UI component again to implement those changes.

Once you complete this configuration, the /plugin/components/plans/index.js file for the getPromotionOffering selector looks similar to the following:

```
import React from 'react';
import Styled from '@oracle-cx-commerce/react-components/styled';
import {connect} from '@oracle-cx-commerce/react-components/provider';
import {getPromotionOffering} from '@oracle-dx4c-buying/common/selector';
import Plan from '../plan-feature';
import css from './styles.css';


function Plans(props) {
 const {promotionOfferings} = props;

 return (
 <Styled id="PlanFeatures" css={css}>
 <div>
 <div className="plan-features-container">
 {promotionOfferings &&
 promotionOfferings.map(promotion => {
 return (
 <div key={`promotion_${promotion.id}`}>
 <Plan promotion={promotion} {...props} />
 </div>
 );
 })}
 </div>
 </div>
 </Styled>
 );
}

export default connect(getPromotionOffering)(Plans);
```

The Communications Open Storefront Framework provides a number of selectors that you can use to connect your components to various parts of the Redux state. You can find them in the @oracle-dx4c-buying/common/selector/index.js file. If you can't find a selector that meets your needs, you can create your own selector. For more information, see the Create Selector topic in this chapter.

You can also use the useSelector hook to obtain part of the Redux state information.

Create a Selector

You use selectors to access state models and return specific data from the Redux data store. It selects a data, only if it's updated.

The selector when used with the connect component, passes the properties in the Redux store into the UI component as props and redisplays the UI component if any of the properties change. You can find the default selectors in the @oracle-dx4c-buying/common/selector/index.js file.

In the Plans widget example, the getPromotionOffering selector is used in multiple widgets, hence it's defined in the index.js file in the common directory. If a selector is more specific to an individual widget, you can write the selector in the widget code, or create a dedicated selectors.js file within the widget file structure.

In the Plans widget example, assume that the promotionOfferings object contains the plans data and it's automatically populated in the following part of the Redux state:

```
dx4cRepository: {
 planContext: {
 planSearch: {
 loading: false,
 promotionOfferings: [...]
 }
 }
}
```

You can write the getPromotionOffering selector to extract the promotionOfferings object in the Redux state into your UI component. Here's the sample getPromotionOffering selector file:

```
/**
* Selector to extract promotionOffering from a state
*/
export const getPromotionOffering = state => {
 const planContext = getDX4CRepository(state).planContext || {};
 const {planSearch} = planContext;
 if (planSearch && planSearch.planConfiguration) {
 const promotionOfferings = planSearch.promotionOfferings || [];

 return {
 promotionOffering: promotionOfferings
 };
 }

 return {promotionOffering: []};
};
```

Write an Action

You use actions to trigger an update of the Redux state. Actions don't update the state directly, but use reducers, which describe exactly how the state should change in response to a given action.

A reducer is a function that takes the previous state and an action and returns the next state. Communications Open Storefront Framework provides a number of default actions. You can find them in the **@oracle-dx4c-buying/actions/ <action_type>** directory. If you can't find an action that meets your needs, create an action for your business scenario and store it in this directory.

When you write an action, you specify the type of action that you expect the application to perform, such as get state. The application performs that action by calling a Buying and Billing API endpoint. The application uses Redux-Saga, which enables asynchronous calls, to make calls to the Buying and Billing API endpoints. Once the endpoint sends it response, the application takes the endpoint's response and places it in a specific part of the store. This triggers the selectors that listen to that part of the store and renders the updated data into the UI component.

In the Plans widget example, we assumed that the promotionOfferings object is automatically populated in the Redux state. Now, let's see how you can use an action to populate the same object in the Redux state. To do so, you write an action named _getPlans in the **actions/plan/index.js** file. Here's the sample getPlans action:

```
import {takeLeading, createReducer, combineReducers} from '@oracle-cx-commerce/store/utils';
import {LIST_PRODUCT_OFFERINGS} from '@oracle-dx4c-buying/common/endpoint/endpoints';
import {callEndpointSaga} from '@oracle-dx4c-buying/common/endpoint/endpoint-saga';
import {parsePlans} from './parser/plan-parser';

function* getPlansSaga(action) {
 return yield callEndpointSaga(action, LIST_PRODUCT_OFFERINGS, parsePlans);
}

function loadingReducer(state /*, action*/) {
 return {...state, loading: true};
}

export default {
 reducer: combineReducers({
 dx4cRepository: combineReducers({
 planContext: combineReducers({
 planSearch: createReducer({
 _getPlans: loadingReducer
 })
 })
 })
 }),
 *saga() {
 yield takeLatest('_getPlans', getPlansSaga);
 }
};
```

The getPlans action contains a reducer function called loadingReducer, which populates a loading mark in the state when the data is retrieved from the endpoint. Most of the actions in the communications storefront application follow the same pattern. Here's the sample loadingReducer function:

```
function loadingReducer(state /*, action*/) {
 return {...state, loading: true};
}

reducer: combineReducers({
 dx4cRepository: combineReducers({
 planContext: combineReducers({
 planSearch: createReducer({
 _getPlans: loadingReducer
 })
 })
})
```

Now this function is called when the _getPlans action is triggered. The state once the action is complete looks similar to the following:

```
dx4cRepository: {
 planContext: {
 planSearch: {
 loading: true
 }
 }
}
```

In our example, the getPlans action uses a saga (Redux-Saga), which updates the state when the data is retrieved from the endpoint. However, you can write corresponding reducers as needed when a particular action is complete.

```
*saga() {
 yield takeLatest('_getPlans', getPlansSaga);
}
```

Sagas use the takeEvery and takeLatest functions to update the state. The takeEvery function allows multiple instances to be started concurrently. The takeLatest function allows only the latest request to be triggered (displaying the latest version of data). It only allows one task to run at any moment.

When you create an action that uses the saga function, the application adds it to the saga middleware to call an endpoint.

In our example, the getPlansSaga calls the LIST_PRODUCT_OFFERINGS endpoint to complete the action:

```
import {LIST_PRODUCT_OFFERINGS} from '@oracle-dx4c-buying/common/endpoint/endpoints';
 import {callEndpointSaga} from '@oracle-dx4c-buying/common/endpoint/endpoint-saga';
 import {
 transformProductOfferingsForPlanSearch,
 } from '@oracle-dx4c-buying/actions/transform/product-offering/product-offering';

 function* getPlansSaga(action) {
 return yield callEndpointSaga(action, LIST_PRODUCT_OFFERINGS, transformProductOfferingsForPlanSearch);
 }
}
```

In this file, you can see that the getPlansSaga also passes a transform callback function to the callEndpointSaga. Transform functions typically take the response from the API endpoint and convert them into a compact model that's more suitable for the UI development. The endpoint places the output of this transform operation directly into the state. In this case, the transform function plays the role of the reducer by updating the appropriate part of the state.

Here's the sample transformProductOfferingsForPlanSearch function:

```
export function transformProductOfferingsForPlanSearch(plans, state) {
 return {
 dx4cRepository: {
 planContext: {
 planSearch: {
 loading: false,
 ...transformProductOfferingsIntoPlans(plans, state)
 }
 }
 }
 }
```

```
     }
    }
```

You can find the specifics of the transformProductOfferingsIntoPlans function in the **@oracle-dx4c-buying/actions/utils/transform/product-offering/product-offering.js** file. The transform function generates similar output for the Plans widget:

```
dx4cRepository: {
 planContext: {
 planSearch: {
 loading: false,
 promotionOfferings: [...] //This is the output of the transform function above
 }
 }
}
```

Note that the loading flag has been set to false in this output to indicate that the data loading is complete.

Once you create the action, you can add it in the Plans index.js file to retrieve the plan data. Here's the sample **Plans index.js** file with the getPlans action:

```
import React, {useEffect, useContext} from 'react';
import Styled from '@oracle-cx-commerce/react-components/styled';
import {connect} from '@oracle-cx-commerce/react-components/provider';
import {StoreContext} from '@oracle-cx-commerce/react-ui/contexts';
import {getPromotionOffering} from '@oracle-dx4c-buying/common/selector';
import Plan from '../plan';
import css from './styles.css';

function Plans(props) {
 const {promotionOffering} = props;

 const {action} = useContext(StoreContext);

 useEffect(() => {
 action('_getPlans', {
 type: 'package',
 limit: 50,
 expand: 'productOfferingPrice',
 priceListName: 'DX4C NA Pricelist'
 });
 }, [action]);

 return (
 <Styled id="Plans" css={css}>
 <div>
 <div className="plans-container">
 {promotionOffering &&
 promotionOffering.map(promotion => {
 return (
 <div key={`promotion_${promotion.id}`}>
 <Plan promotion={promotion} {...props} />
 </div>
 );
 })}
 </div>
 </div>
 </Styled>
 );
}

export default connect(getPromotionOffering)(Plans);
```

You can also add a useEffect to trigger the getPlans action. This hook automatically triggers the getPlans action when the corresponding component is initialized. Once the object in the state is updated, the Plans widget redisplays the plan data on the UI.

Here's a sample useEffect for triggering the getPlans action:

```
const {action} = useContext(StoreContext);

useEffect(() => {
 action('_getPlans', {
 type: 'package',
 limit: 50,
 expand: 'productOfferingPrice',
 priceListName: 'DX4C NA Pricelist'
 });
}, [action]);
```

Create an Endpoint

You use endpoints to retrieve data by calling Buying and Billing APIs.

The Communications Open Storefront Framework provides a number of endpoints for calling the Buying and Billing APIs. See the Design Storefront and Self-Care Pages chapter in this guide for the complete list of endpoints and how they're mapped to the TMF APIs. You can also find the endpoints in the **packages/core/plugins/common/endpoint/endpoints.js** file and the Buying and Billing API URLs in the **@oracle-dx4c-buying/common/endpoint/endpoint-urls/index.js** file. You can use these default endpoints for calling the Buying and Billing APIs, or create an endpoint if you want to integrate a new API.

In the Plans widget example, the promotion offering or plan data is returned by the list-product-offerings endpoint, which maps to the `/productCatalogManagement/v4/productOffering` Buying API.

Here's the directory structure for the list-product-offerings endpoint:

```
/plugins
 /endpoints
 /product-offering
 index.js
 meta.js
 /list-product-offering
 index.js
 meta.js
```

You create the index.js and meta.js files for your endpoints in this directory. Here's the sample **/plugins/endpoint/product-offering/list-product-offerings/index.js** file:

```
/*
 ** Copyright (c) 2020 Oracle and/or its affiliates.
 */

import {getProductOfferingsUrl} from '@oracle-dx4c-buying/common/endpoint/endpoint-urls';
import {handleEndpointResponse, getBuyingEndpointRequest} from '@oracle-dx4c-buying/common/endpoint/
endpoint-saga';
import {LIST_PRODUCT_OFFERINGS} from '@oracle-dx4c-buying/common/endpoint/endpoints';

/**
 * Return an object that implements the endpoint adapter interface.
 */
const listProductOfferings = {
 /** The id of the endpoint */
 endpointId: LIST_PRODUCT_OFFERINGS,

 /**
 * Return a Fetch API Request object to be used for invoking the endpoint.
 *
 * @param payload Optional payload to be included in the request
 * @param state The current application state
 * @return Request object for invoking the endpoint via Fetch API
 */
 async getRequest(payload, state) {
 const {categoryName, type, expand, limit, priceListName} = payload;

 const queryParams = {
 'category.name': categoryName,
 type,
 expand,
 limit,
 'productOfferingPrice.priceList.name': priceListName
 };

 return getBuyingEndpointRequest(state, LIST_PRODUCT_OFFERINGS, getProductOfferingsUrl(queryParams),
 payload);
 },

 /**
 * Return a Fetch API Response object containing data from the endpoint.
 *
 * @param response The Response object returned by the fetch call
 * @param state The current application state
 * @param payload Optional payload that was included in the request
 * @return Response object, augmented with an async getJson function to return
 * an object to be merged into the application state
 */
 getResponse(response, state, payload) {
 return handleEndpointResponse(response, state, payload);
 }
};

export default listProductOfferings;
```

The getRequest function in this file generates the request to call the GET method of the productOfferings Buying API. It adds the appropriate query parameters and passes the request onto the getBuyingEndpointRequest function. The getBuyingEndpointResponse function generates the appropriate authorization headers for the calling the Buying API.

```
async getRequest(payload, state) {
 const {categoryName, type, expand, limit, priceListName} = payload;

 const queryParams = {
 'category.name': categoryName,
 type,
 expand,
 limit,
 'productOfferingPrice.priceList.name': priceListName
 };

 return getBuyingEndpointRequest(state, LIST_PRODUCT_OFFERINGS, getProductOfferingsUrl(queryParams),
 payload);
},
```

The getRequest function passes the response of the endpoint into the handleEndpointResponse function. This function parses the raw response from the Buying API and converts it into the desired UI model. This function also throws an endpoint error if the response has any issues.

```
export function handleEndpointResponse(response, state, payload = {}) {
     let json;
     const {parser} = payload;
     response.getJson = async () => {
     if (json === undefined) {
     json = await getBodyAsJson(response);
     if (response.ok) {
     json.headers = response.headers;

     return parser ? parser(json, state) : json;
     }
     throw new EndpointError(response, payload);
     }

     return json;
     };

     return response;
    }
```

When you create an endpoint, you must also create the appropriate meta.js file. Here's the sample **/plugins/endpoint/product-offering/list-product-offerings/meta.js** file. This file specifies the package ID of the endpoint. The package ID must be unique for each endpoint.

```
/**
 * Metadata for the _listProductOfferings endpoint.
 */
export default {
 packageId: '@oracle-dx4c-buying/endpoints'
};
```

To export the references to the list-product-offerings endpoint and any other product-offering related endpoints, add the following In the **/plugins/endpoint/product-offering/index.js** file:

```
export const _listProductOfferings = () => import('./list-product-offerings');
```

Create a parallel reference in the **/plugins/endpoint/product-offering/meta.js** to export the references to the list-product-offerings endpoint metadata:

```
export * from '@oracle-cx-commerce/endpoints/meta';

// Export references to product-offering endpoints.
export {default as _listProductOfferings} from './list-product-offerings/meta';
```

Now export the list-product-offerings endpoint so that your application can recognize the endpoint. The **/plugins/endpoint/index.js** file contains a set of functions with the names that correspond to the names of your application's endpoints. Edit the **/plugins/endpoint/index.js** file to include the new endpoint:

```
export * from './product-offering';
```

Also, edit the /plugins/endpoint/meta.js file to include the new endpoint:

```
export * from './product-offering/meta';
```

Visualize Components Using Storybook

Use Storybook to visualize and build UI components for your communications storefront and self-care application.

Storybook helps you visualize components in the various configurations. You can have a quick visual reference of the components in this reference application and understand the various properties that can be passed to each component.

To display the components that are connected to the Redux state through the connect function, use the following utility components:

- `@oracle-dx4c-buying/test/test-store`: The test-store component accepts a state object. This component is used to predefine the Redux state without explicitly calling the actions or reducers.
- `@oracle-dx4c-buying/test/storybook-container`: The storybook-container component replicates the layout mechanism of the open storefront framework so that container-style components can be displayed in Storybook. These components can be used to only create stories for the Storybook UI. You can't use them for building the UI for your application.

To start Storybook, enter the following command from your workspace:

```
yarn storybook
```

Storybook is configured to use the port 3001 by default. You can modify the port (if needed) by updating the storybook script in the root package.json file of your workspace. For example:

```
"storybook": "start-storybook -p 3001"
```

The Storybook UI automatically runs on your default browser once it finishes compiling. You can then browse the various communications storefront and self-care components in this UI. You can find all the stories created for a given component in the __stories__ folder of each component. For example, the stories for the Plans component are located in the @oracle-dx4c-buying/components/plans/__stories__/plans.stories file.

For more information on Storybook, visit https://storybook.js.org/.

6 Integrate with External Applications

Integrate Data as a Service

Here are the steps to integrate DaaS (address verification service) with CX Industries Framework:

Configure IDCS

Here's how you can configure:

1. Create an IDCS user in IDCS integrated with DaaS.
2. Add following roles to the created user:
   - DATASERVICE_CLIENT_API_APPID
   - DATASERVICE_USER
3. The user names should be the same in both IDCS (CX Industries Framework and DaaS).
   - Click **Oracle Cloud Services** tab, in the Application Roles section, users are assigned to the following roles:
     - DATASERVICE_CLIENT_API_APPID
     - DATASERVICE_USER
     - DATASERVICE_ADMINISTRATOR
4. Click **Applications** tab, click **Configuration**. Create an application with the following client configuration:
   a. In the Client Configuration section, Select **Register Client** option.
   b. Select the required options such as Resource Owner, Client Credentials, Refresh Token, and Authorization Code corresponding to Allowed Grant Types.
   c. Select **Allow non-HTTPS URLs** option and provide appropriate value for Redirect URL.
   d. For Client Type, select **Trusted**.
   e. In the Token Issuance Policy section, for Authorized Resources, select **Specific**.
   f. In the Resources section, select **Register Resources**.
   g. Configure application APIs in the `oAuth` protected section:
      - Access Token Expiration is set to a value such as 3,600 seconds.
      - Refresh Token Expiration is set to a value such as 604,800 seconds.
      - For Primary Audience field, enter the primary recipient where the token is processed.
5. a. Click **Applications** tab, click **Users**.

   The users are listed.

Create System Descriptors (TTD)

```
https://<host>:<port>/admin/systemDescriptors
Method: POST
Payload
{
 "target-name": "Address Verification",
 "external": {
 "apis": [
 {
```

```
 "api-id": "orcl-303",
 "api-version": "v3",
 "api-resources": [
 {
 "resources": [
 {
 "resource-id": "address"
 }
 ],
 "url-prefix": "av"
 }
 ]
 }
 ]
 },
 "system": "AddressVerify",
 "edk-enabled": false,
 "domain": "AddressVerify",

 "type": "external"
}
```

Create Connection Descriptors (TIC)

```
https://<host>:<port>/admin/connectionDescriptors
Method: POST
Payload
{
 "system-descriptor": "<TTD id>",
 "endpoint-name": "address-verify-api",
 "endpoint-url": "<DAAS API>",
 "fabric-facing-auth": {
 "oidc-client-credentials": {
 "client-id": "<Client Id>",
 "client-secret": "<Client Secret>",
 "identity-uri": "<IDENTITY_URI>",
 "scope": "<Scope>" //Default Scope is urn:opc:idm:__myscopes__ }
 },
 "type": "external"
}
```

Apply Routing Rules

```
https://<host>:<port>/admin/gatekeepingRules/<gkr>
Method: PUT
Payload
{
 "endpoint-name": "address-verify-api",
 "rule-name": "Generated gatekeeping rule for endpoint address-verify-api",
 "external-event-emitter-identification":"<Client Id>",
 "destination-selection": [
 {
 "api-id": "orcl-303",
 "api-version": "v3",
 "criteria": [
 {
 "rank": 2,
 "resource-ids": [
 "address"
 ]
 }
 ]
```

```
    }
    ]

   }
```

Test the Configuration

```
https://<host>:<port>/api/api/v3/address/verify
Method: POST
Headers:
x-id-dataservice-user: datauser

Payload:
{
 "inputs":[
 {
 "Address1":"london",
 "Address2":"Shanfrfrfkar V",
 "City":"Srinagarffff",
 "Postalcode":"ssssssss"
 }
 ]
}
```

Enable Address Verification for Address Search

You must enable address verification for address search.

Enable Address Verification in the Tenant Configuration File for Storefront Application

Change Tenant Configuration **enableDaas** from enable: 'false' to enable: 'true'.

> **Note:** Enable address verification for the Storefront application only after integrating address verification API with CX Industries Framework.

Test Service Installation with Mock Services

The mock API implementation displays the available time slots on the Service Installation page in the checkout flow.

> **Note:** This application starts only in a machine with Nodejs installed.

Steps to start mock project:

1. Provide available time slots in the file slots.json or availableSlots.json. Use the format for date and time slot as shown in the following sample:

   ```
   {
    "dates": [
   ```

```
         {
         "date": "2023-11-06T00:00:00.000Z",
         "timeSlots": [
         {
         "timeSlot": "9am to 10am"
         },
         {
         "timeSlot": "10am to 11am"
         },
         {
         "timeSlot": "2pm to 3pm"
         },
         {
         "timeSlot": "4pm to 5pm"
         }
         ]
         },
         {
         "date": "2023-11-09T00:00:00.000Z",
         "timeSlots": [
         {
         "timeSlot": "8am to 9am"
         },
         {
         "timeSlot": "11am to 12pm"
         },
         {
         "timeSlot": "12pm to 1pm"
         },
         {
         "timeSlot": "2pm to 3pm"
         },
         {
         "timeSlot": "3pm to 4pm"
         },
         {
         "timeSlot": "4pm to 5pm"
         }
         ]
         }
         ]
        }
```

2. Start the application by running the **node index.js** command from dx4c-appointment-handle folder.
3. Verify if the API is started correctly by accessing the following URL:

   For server deployed app with secure mode (IS_LOCAL: N)
   `https://<hostname>:<port>/api/01/getAvailableSlots`
4. Configure the API URL in Storefront configuration variable: dx4cInstallationScheduleServiceUrl
   `https://<hostname>:<port>`

Configure Oracle Commerce Cloud Settings

Here's how you can:

1. Configure the URL in the Oracle Commerce Cloud settings (DX4C store site settings) in the Service Installation Scheduling URL field.

2. Publish the changes.
    a. Click **Default** under **Changes**.
    b. Select changes done on the **Page Repository Site**.
    c. Click **Publish Workset** button to publish.

    Done dialog box appears.

Access API URL from Storefront UI

You can access this API URL:

```
https://<hostname>:<port>/api/01/getAvailableSlots
```

Steps to Add Terms and Conditions PDF File for Downloading from Service Installation Page in the Checkout Flow

Note: Terms and Conditions link is a placeholder to provide PDF for the client.

Steps to upload the PDF document:

1. Identify a Terms and Conditions PDF file.
2. Upload the PDF with slug name 1481787191863-termsandconditions in Storefront connected OCE in DX4C_Assets.

    Sample value to enter in the request:

    ```
    {"name": "TermsAndConditions.pdf","repositoryId": "{{REPOSITORY_ID}}","type": "DX4C-Asset", "slug":
     "1481787191863-termsandconditions"}
    ```

3. Publish the PDF file.
4. Download the PDF from service installation step in the checkout flow.

    Provide the slug name for the PDF in Oracle Content and Experience as **1481787191863-termsandconditions**.

Configure Customer Reviews URL

Here's how you can configure:

1. Configure the following mock API URL instead of Customer Reviews Service in the **Customer Reviews URL** field in the DX4C Store Site Settings of the Admin Console.

    ```
    https://<hostname>:<port>
    ```

2. Publish the changes.
3. Click **Default** under **Changes**.
4. Select changes done on the **Page Repository Site**.
5. Click **Publish Workset** button to publish.

    Done dialog box appears.

7 Manage Tenant Configuration

Configure Tenant

Tenant configuration is a REST service available to manage business configuration for the storefront application.

1. To view existing configuration:
   a. Request URL: `<fabric-host>/admin/tenantConfigurations`
   b. Operation: `GET`
   c. Authorization: `OAUTH2.0`
      Use bearer token generated from `<FA-Host>/fscmRestApi/tokenrelay?scope=<Fabric-Host>`
2. To edit the configuration, get the response using GET call and update the required configuration and do a PUT operation to the same endpoint with the whole object as payload.
3. Here are the business configurations available by default:

Area	Comments	Object	Configuration Key	Configuration Value	Default Value	Comments
salesCatalog		catalogMenu	browsingStartMen	Catalog or Category	Category	Controls from where the browse menu starts. Either from catalog or category in storefront header.
salesCatalog		catalogMenu	relevantCatalogs	A valid catalog name to browse the categories within that catalog	Supremo %20Consumer %20 Catalog	Controls where to look for categories.
salesCatalog		catalogMenu	relevantCategoryL	A valid category names separated by a comma	Residential	Controls which categories to display on UI header.
salesCatalog		catalogMenu	relevantCategoryL	A valid category names separated by a comma	Accessory,Smart %20Phone, Mobile,Streaming %20Service, SIM,Internet %20Packages	Controls which categories to display within previous L1 category.
salesCatalog		daas	enableDaas	true/false	false	Controls if address

Area	Comments	Object	Configuration Key	Configuration Value	Default Value	Comments
						verification service can be enabled or not.
salesCatalog		accessoryConfig	Color	Color%20Palette	Color%20Palette	
salesCatalog		accessoryConfig	Capacity	Selection %20Button	Selection %20Button	
salesCatalog		accessoryConfig	Size	Selection %20Button	Selection %20Button	
salesCatalog		accessoryConfig	Style	Selection %20Button	Selection %20Button	
salesCatalog		search	filters	A valid category names separated by a comma	Smart%20Phone, Internet %20Packages	Controls the suggestions to be shown on the search page.
salesCatalog		homeWidgetCateg	carousel	Category Name	TV Promotion, Internet Promotion, Streaming Promotion, Mobile Promotion	Controls what product categories are shown on the carousel widgets on the homepage.
salesCatalog		homeWidgetCateg	BestSeller	Category Name	Mustang 2011, Mustang 2011 Pro, Vola S10,Ty G8 ThinkQ	Controls what product category is mapped to the best seller widget on the homepage.
salesCatalog		homeWidgetCateg	topPlans	Category Name	Recommendation	Controls what product category is mapped to the top plans widget on the homepage.
salesCatalog		homeWidgetCateg	topBanner	Category Name	LimitedTime	Controls what product category is mapped to the top banner widget on the homepage.
salesCatalog		homeWidgetCateg	MiddleAdBanner	Category Name	Bundle Promotion	Controls what product category

Area	Comments	Object	Configuration Key	Configuration Value	Default Value	Comments
						is mapped to the middle advertisement widget on the homepage.
salesCatalog		homeWidgetCateg	Promotion1	Category Name	Device Promotion	Controls what product category is mapped to the second promotion widget on the homepage.
salesCatalog		homeWidgetCateg	Promotion2	Category Name	New Customer Promo	Controls what product category is mapped to the second promotion widget on the homepage.
salesCatalog		homeWidgetImag	carousel	Image URL	carousel-1,carouse sel-4	Controls what images are mapped to the carousel widgets on the homepage.
salesCatalog		homeWidgetImag	topPromotionBan	Image URL	promo-belt	Controls what image is mapped to the top promotion banner widget on the homepage.
salesCatalog		homeWidgetImag	MiddleAdBanner_ Image	Image URL	home-promo	Controls what image is mapped to the middle advertisement banner widget on the homepage.
salesCatalog		homeWidgetImag	Promotion1_ Image	Image URL	Device Promotion	Controls what image is mapped to the first promotion banner widget on the homepage.

Area	Comments	Object	Configuration Key	Configuration Value	Default Value	Comments
salesCatalog		homeWidgetImag	Promotion2_ Image	Image URL	New Customer Promo	Controls what image is mapped to the second promotion banner widget on the homepage.
salesCatalog		pdpWidgetImageN	TopAdBanner_ Image	Image URL	promo-belt	Controls what image is mapped to the top advertisement banner widget on the plan details page.
salesCatalog		pdpWidgetImageN	poFeatureBanner_ Image	Image URL	home-promo	Controls what image is mapped to the feature banner widget on the plan details page.
salesCatalog		pdpWidgetImageN	pdpPromotion1_ Image	Image URL	Device Promotion	Controls what image is mapped to the first promotion banner widget on the plan details page.
salesCatalog		pdpWidgetImageN	pdpPromotion2_ Image	Image URL	New Customer Promo	Controls what image is mapped to the second promotion banner widget on the plan details page.
salesCatalog		pdpWidgetCatego	TopAdBanner	Category Name	Streaming Promotion	Controls what product category is mapped to the top advertisement banner widget on the plan details page.
salesCatalog		pdpWidgetCatego	poFeatureBanner_ Image	Category Name	Promotion	Controls what product category is mapped to the

Area	Comments	Object	Configuration Key	Configuration Value	Default Value	Comments
						feature banner widget on the plan details page.
salesCatalog		pdpWidgetCatego	pdpPromotion1	Category Name	Internet Promotion	Controls what product category is mapped to the first promotion banner widget on the plan details page.
salesCatalog		pdpWidgetCatego	pdpPromotion2	Category Name	Hotspot Accessory Promotion	Controls what product category is mapped to the second promotion banner widget on the plan details page.
salesCatalog		plansGrid	planGridBanner	Image URL	plans-grid-banner	
salesCatalog		productsGrid	productsGridBann	Image URL	products-grid-banner	
salesCatalog		myaccountwidget(	MyAccountDiscou	Category Name	Internet Promotion	Controls what product category is mapped to the first discount banner on My Account page.
salesCatalog		myaccountwidget(	MyAccountDiscou 1	Category Name	Internet Promotion	Controls what product category is mapped to the second discount banner on My Account page.
salesCatalog		myaccountwidget(	MyAccountDiscou 2	Category Name	Bundle Promotion	Controls what product category is mapped to the third discount banner on My Account page.
salesCatalog		myaccountwidgetI	MyAccountDiscou Image	Image URL	img-1	Controls what image is mapped

Area	Comments	Object	Configuration Key	Configuration Value	Default Value	Comments
						to the first discount banner on My Account page.
salesCatalog		myaccountwidgetI	MyAccountDiscou 1_Image	Image URL	img-1	Controls what image is mapped to the second discount banner on My Account page.
salesCatalog		myaccountwidgetI	MyAccountDiscou 2_Image	Image URL	img-1	Controls what image is mapped to the third discount banner on My Account page.
salesCatalog		orderCancellation	cancellationReaso	Cancel Reason Strings	Order Created by Mistake, Shipping Cost Too High, Product Price Too High, Found Cheaper Somewhere Else, Order Delay, Incorrect Billing Address, Need To Change Payment Method	Defines the cancellation reasons that will be displayed to the customer when canceling an order.
salesCatalog		orderDuration	orderDurationinDa	Count in Days	30	Dictates the time period for fetching the orders to be shown on the order history page.
salesCatalog		defaultDiscountTy	discountType	Discount Priority	Limited Time	Defines the priority of the discount type. This dicates what discounts are displayed on the plan and the device summary card when there are three or more

Area	Comments	Object	Configuration Key	Configuration Value	Default Value	Comments
						discounts for an offer.
salesCatalog		upSellRecommend	popUpDuration	Time	5 Secs	Dictates how long the upsell recommendation pop up window will remain on the screen.
salesCatalog		crossSellRecomme	maxItemsToDispla	Item Count	3	Dictates how many cross recommendations will be shown to the shopper.
footerLinks		footerLinksMap	siteMapLabel	siteMap	siteMap	
footerLinks		footerLinksMap	termsOfUseAndPr	terms	terms	
footerLinks		footerLinksMap	countrySelectorLa	countrySele	countrySele	
footerLinks		footerLinksMap	cookiePreferences	cookiePref	cookiePref	
B2CLinks		whySupremoLabe	ourCoverageLabel	ourCovrage	ourCovrage	
B2CLinks		whySupremoLabe	supremoAnd5GLa	supremo5G	supremo5G	
B2CLinks		purchasingOnlinel	bestSellingPhones	bestSelling	bestSelling	
B2CLinks		purchasingOnlinel	specialDealsLabel	specialDeals	specialDeals	
B2CLinks		purchasingOnlinel	deviceAccessories	deviceAccess	deviceAccess	
B2CLinks		helpLabel	generalHelpLabel	generalHelp	generalHelp	
B2CLinks		helpLabel	deviceHelpLabel	deviceHelp	deviceHelp	
B2CLinks		helpLabel	faqsLabel	faqs	faqs	
B2CLinks		contactUsLabel	returnsLabel	returns	returns	
B2CLinks		contactUsLabel	lostOrStolenDevic	lostOrStolen	lostOrStolen	
B2CLinks		contactUsLabel	repairsLabel	repairs	repairs	
B2CLinks		supremoLabel	aboutUsLabel	about	about	
B2CLinks		supremoLabel	careersLabel	careers	careers	
B2CLinks		supremoLabel	findAStoreLabel	fas	fas	

8 Appendix

Work with REST API Endpoints

The Communications Open Storefront Framework uses the Buying and Billing APIs to retrieve data for the storefront. While designing the layouts and widgets for your storefront, refer to this table for information on the APIs that are integrated with Communications Open Storefront Framework.

Buying Experience APIs

Here are the APIs for the Product Offering entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get Product Offering	GET	`/ productCatalogManagem v4/productOffering/ {id}`	`dx4cRetrieveProductOf`	@oracle-dx4c-buying/ endpoints/product-offering/retrieve-product-offering
Get all Product Offerings	GET	`/ productCatalogManagem v4/productOffering`	`dx4cListProductOfferi`	@oracle-dx4c-buying/ endpoints/product-offering/list-product-offerings
Get All Catalogs	GET	`/ productCatalogManagem v4/catalog/? catalogType=product`	`dx4cRetrieveCatalogs`	@oracle-dx4c-buying/ endpoints/catalog/ retrieve-catalogs
Get Category	GET	`/ productCatalogManagem v4/category/{id}`	`dx4cRetrieveCategory`	@oracle-dx4c-buying/ endpoints/category/ retrieve-category
Get all Categories	GET	`/ productCatalogManagem v4/productOffering/ ? catalogId={id}`	`dx4cListCategory`	@oracle-dx4c-buying/ endpoints/category/list-category

Here are the APIs for the Product entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get a Product	GET	`/productInventory/v4/ product/{id}`	`dx4cRetrieveProduct`	@oracle-dx4c-buying/ endpoints/product/ retrieve-product

API	Method	Path	Endpoint ID	Endpoint Source
Get all Products	GET	`/productInventory/v4/product`	`dx4cListProducts`	@oracle-dx4c-buying/endpoints/product/list-products
Calculate disconnect penalty for a product	GET	`/productInventory/v4/product/{id}/calculateDisconnectPe`	`dx4cCalculateProductP`	@oracle-dx4c-buying/endpoints/product/calculate-product-penalty-charges
Get product change options	GET	`/productInventory/v4/product/{id}/changePackage`	`dx4cRetrieveProductPa`	@oracle-dx4c-buying/endpoints/product/retrieve-product-package-charges
Update a product	PATCH	/productInventory/v4/product/{id}	dx4cPatchProduct	

Here are the APIs for the Shopping Cart entity:

API	Method	Path	Endpoint ID	Endpoint Source
Check out a shopping cart	POST	`/shoppingCart/v4/shoppingCart/{id}/checkout`	`dx4cCheckoutShoppingC`	@oracle-dx4c-buying/endpoints/shopping-cart/checkout-shopping-cart
Create a shopping cart	POST	`/shoppingCart/v4/shoppingCart`	`dx4cCreateShoppingCar`	@oracle-dx4c-buying/endpoints/shopping-cart/create-shopping-cart
Get a shopping cart	GET	`/shoppingCart/v4/shoppingCart/{id}`	`dx4cRetrieveShoppingC`	@oracle-dx4c-buying/endpoints/shopping-cart/retrieve-shopping-cart
Update a shopping cart	PATCH	`/shoppingCart/v4/shoppingCart/{id}`	`dx4cPatchShoppingCart`	@oracle-dx4c-buying/endpoints/shopping-cart/patch-shopping-cart
Create a subscriber shopping cart	POST	/shoppingCart/v4/shoppingCart	dx4cCreateSubscriberShopp	@oracle-dx4c-buying/endpoints/subscriber-shopping-cart/create-subscriber-shopping-cart
Get a subscriber shopping cart by id	GET	/shoppingCart/v4/shoppingCart/{id}	dx4cRetrieveSubscriberShop	@oracle-dx4c-buying/endpoints/subscriber-shopping-cart/retrieve-subscriber-shopping-cart
Update a subscriber shopping cart	PATCH	/shoppingCart/v4/shoppingCart/{id}	dx4cPatchSubscriberShoppi	@oracle-dx4c-buying/endpoints/subscriber-shopping-cart/patch-subscriber-shopping-cart

API	Method	Path	Endpoint ID	Endpoint Source
Get a subscriber shopping cart by query parameter	GET	/shoppingCart/v4/ shoppingCart?<param_ name>=<param_value>	dx4cListSubscriberShopping	@oracle-dx4c-buying/ endpoints/subscriber- shopping-cart/list- subscriber-shopping-cart
Check out a subscriber shopping cart	POST	/shoppingCart/v4/ shoppingCart/{id}/ checkout	dx4cCheckoutSubscriberSho	@oracle-dx4c-buying/ endpoints/subscriber- shopping-cart/checkout- subscriber-shopping-cart

Here are the APIs for the Product Orders entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get a product order	GET	`/ productOrderingManage v4/productOrder/{id}`	`dx4cRetrieveProductOr`	@oracle-dx4c-buying/ endpoints/product-order/ retrieve-product-order
Get all product orders	GET	`/ productOrderingManage v4/productOrder`	`dx4cListProductOrder`	@oracle-dx4c-buying/ endpoints/product-order/ list-product-order
Create a product order	POST	`/ productOrderingManage v4/productOrder`	`dx4cCreateProductOrde`	@oracle-dx4c-buying/ endpoints/product-order/ create-product-order
Update a product order	PATCH	`/ productOrderingManage v4/productOrder/{id}`	`dx4cPatchProductOrder`	@oracle-dx4c-buying/ endpoints/shopping-cart/ patch-product-order

Here's the API for the Product Offering Price entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get a product offering price	GET	`/ productCatalogManagem v4/ productOfferingPrice/ {id}`	`dx4cRetrieveProductOf`	@oracle-dx4c-buying/ endpoints/product- offering-price/retrieve- product-offering-price

Here are the APIs for the Payment Method entity:

API	Method	Path	Endpoint ID	Endpoint Source
Create a payment method	POST	`/paymentMethods/v1/ paymentMethod`	`dx4cCreatePaymentMeth`	@oracle-dx4c-buying/ endpoints/payment- method/create-payment- method

API	Method	Path	Endpoint ID	Endpoint Source
Update a payment method	PATCH	`/paymentMethods/v1/paymentMethod/{paymentMethodId}`	`dx4cPatchPaymentMetho`	@oracle-dx4c-buying/endpoints/payment-method/patch-payment-method

Here are the APIs for the Individual entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get all individuals	GET	`/party/v4/individual`	`dx4cListIndividual`	@oracle-dx4c-buying/endpoints/individual/list-individual
Create an individual	POST	`/party/v4/individual`	`dx4cCreateIndividual`	@oracle-dx4c-buying/endpoints/individual/create-individual
Update an individual	PATCH	`/party/v4/individual/{id}`	`dx4cPatchIndividual`	@oracle-dx4c-buying/endpoints/individual/patch-individual

Here are the APIs for the Customer entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get all customers	GET	`/customerManagement/v4/customer`	`dx4cListCustomer`	@oracle-dx4c-buying/endpoints/customer/list-customer
Get a customer	GET	`/customerManagement/v4/customer/{id}`	`dx4cRetrieveCustomer`	@oracle-dx4c-buying/endpoints/customer/retrieve-customer
Create a customer	POST	`/customerManagement/v4/customer`	`dx4cCreateCustomer`	@oracle-dx4c-buying/endpoints/customer/create-customer
Update a customer	PATCH	`/customerManagement/v4/customer{id}`	`dx4cPatchCustomer`	@oracle-dx4c-buying/endpoints/customer/patch-customer

Here are the APIs for the Party entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get all party accounts	GET	`/accountManagement/v4/partyAccount`	`dx4cLlistPartyAccount`	@oracle-dx4c-buying/endpoints/party-account/list-party-account
Create a party account	POST	`/accountManagement/v4/partyAccount/{id}`	`dx4cCreatePartyAccoun`	@oracle-dx4c-buying/endpoints/party-account/create-party-account
Update a party account	PATCH	`/accountManagement/v4/partyAccount/{id}`	`dx4cPatchPartyAccount`	@oracle-dx4c-buying/endpoints/party-account/patch-party-account

Here is the API for the Product Line entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get a Product Line	GET	`/productCatalogManagem v4/productLine`	`dx4cRetrieveProductLi`	@oracle-dx4c-buying/endpoints/product-line/retrieve-product-line

Here is the API to onboard a user:

API	Method	Path	Endpoint ID	Endpoint Source
Onboard User	POST	`/buyingUserManagement/v1/onboardUser`	`dx4OnboardUser`	@oracle-dx4c-buying/endpoints/user/onboarduser

Billing APIs

Here are the APIs for the Billing Cycle Specification entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get all billing cycle specifications	GET	`/accountManagement/v4/billingCycleSpecifica`	`dx4cListBillingCycleS`	@oracle-dx4c-buying/endpoints/billing-cycle-specifications/list-billing-cycle-specification
Get a billing cycle specification	GET	`/accountManagement/v4/billingCycleSpecifica`	`dx4cRetrieveBillingCy`	@oracle-dx4c-buying/endpoints/billing-cycle-specifications/retrieve-billing-cycle-specification

Here's the API for the Bucket entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get all buckets	GET	`/ prepayBalanceManageme v4/bucket`	`dx4cListBucket`	@oracle-dx4c-buying/ endpoints/buckets/list-bucket

Here's the API for the Customer Bill entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get all customer bills	GET	`/ customerBillManagemen v4/customerBill`	`dx4cListCustomerBill`	@oracle-dx4c-buying/ endpoints/buckets/list-customer-bill

Here's the API for the Payment entity:

API	Method	Path	Endpoint ID	Endpoint Source
Get all payments	GET	`/payment/v4/payment`	`dx4cListPayment`	@oracle-dx4c-buying/ endpoints/buckets/list-payment

Transform Endpoint Responses

Typically, the responses received from the Buying and Billing APIs can be quite large and contain complex structures. You may not require all of the information in the response for your storefront. In such cases, use the transform functions in the Communications Open Storefront Framework to convert the Buying and Billing API responses into a more compact UI model, which can be used by your UI components.

Transform functions are located in the @oracle-dx4c-buying/actions/utils/transform directory and they are grouped per API entity.

For example, you can use the functions in the @oracle-dx4c-buying/actions/utils/transform/product-offering/ product-offering.js file for transforming the Get a Product API's response.

Here's the sample Get Product API response payload:

```
{
 "isBundle": false,
 "productSpecification": {
 "id": "0CX-1X8ODC",
 "href": "https://SERVER-INFO/productCatalogManagement/v4/productSpecification/0CX-1X8ODC",
 "name": "Wireless Mustang Handset PS"
 },
 "lifecycleStatus": "active",
 "@type": "ProductOfferingEligibleOracle",
 "marketingFeature": [],
 "description": "Say hello, to the Mustang 11!
A triple-camera system provides gorgeous high fidelity images.
This model has the speedy chip that has the critics in awe.
```

```
    And it features the highest-quality video in a smartphone, so your memories look better than ever!",
     "productOfferingPrice": [
     {
     "@type": "ProductOfferingPriceOracle",
     "id": "0CX-1X8P6I",
     "href": "https://SERVER-INFO/productCatalogManagement/v4/productOfferingPrice/0CX-1X8P6I",
     "isBundle": false,
     "lifecycleStatus": "active",
     "name": "Mustang 11 USD Variant Price",
     "priceType": "oneTime",
     "constraint": [
     {
     "@type": "ConstraintRefPricingVariantOracle",
     "name": "Mustang 11 - USD Pricing Constraint",
     "productSpecCharUse": [
     {"@type": "ProductSpecificationCharacteristicValueUseOracle", "name": "Capacity", "charSpecSeq": 1}
     ]
     }
     ],
     "price": {"unit": "USD", "value": 599.0},
     "validFor": {"startDateTime": "2021-01-01T00:00:00.000Z"},
     "@baseType": "ProductOfferingPrice",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOfferingPrice/describe",
     "compositePopRelationship": [
     {
     "@type": "ProductOfferingPriceOracle",
     "name": "Mustang 11 - USD Variant 1",
     "constraint": [
     {
     "@type": "ConstraintRefPricingVariantOracle",
     "constraintRule": [{"name": "Tag-CapacityModel", "rank": 1, "valueRelationship": [{"value": "32"}]}]
     }
     ],
     "price": {"value": 599.0}
     }
     ],
     "priceList": [{"id": "0CX-1X8P20", "name": "DX4C NA Pricelist"}]
     }
     ],
     "type": "device",
     "offerOption": [],
     "productLine": [
     {
     "id": "0CX-1X8K0V",
     "href": "https://SERVER-INFO/productCatalogManagement/v4/productLine/0CX-1X8K0V",
     "name": "Mustang and Vola Handsets"
     }
     ],
     "offerRecommendation": [],
     "@baseType": "ProductOffering",
     "attachment": [],
     "isCustomizable": true,
     "id": "0CX-1X8P2O",
     "href": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/0CX-1X8P2O",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe",
     "sku": [
     {
     "sku": "M110R-1WFGT9-25",
     "attributes": [
     {"name": "Model", "value": "Mustang 11"},
     {"name": "Capacity", "value": "32", "unitOfMeasure": "GB"},
     {"name": "Color", "value": "Red", "valueCode": "#FF0000"}
     ]
     },
     {
     "sku": "M110B-1WFGT9-13",
     "attributes": [
     {"name": "Model", "value": "Mustang 11"},
     {"name": "Capacity", "value": "32", "unitOfMeasure": "GB"},
     {"name": "Color", "value": "Black", "valueCode": "#000000"}
     ]
     }
     ],
     "prodSpecCharValueUse": [
     {
     "@type": "ProductSpecificationCharacteristicValueUseOracle",
     "maxCardinality": 1,
     "minCardinality": 1,
     "name": "Color",
     "valueType": "string",
     "isConfigurable": true,
     "productSpecCharacteristicValue": [
     {
     "@type": "ProductSpecificationCharacteristicValueOracle",
     "isDefault": true,
     "valueType": "string",
     "validFor": {"endDateTime": "2999-12-31T00:00:00.000Z", "startDateTime": "2021-01-01T00:00:00.000Z"},
     "value": "Red",
     "@baseType": "ProductSpecificationCharacteristicValue",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe"
     },
     {
     "@type": "ProductSpecificationCharacteristicValueOracle",
     "isDefault": false,
     "valueType": "string",
     "validFor": {"endDateTime": "2999-12-31T00:00:00.000Z", "startDateTime": "2021-01-01T00:00:00.000Z"},
     "value": "Black",
     "@baseType": "ProductSpecificationCharacteristicValue",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe"
     }
     ],
     "@baseType": "ProductSpecificationCharacteristicValueUse",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe"
     },
     {
     "@type": "ProductSpecificationCharacteristicValueUseOracle",
     "maxCardinality": 1,
     "minCardinality": 1,
     "name": "Capacity",
     "valueType": "number",
     "isConfigurable": true,
     "productSpecCharacteristicValue": [
     {
     "@type": "ProductSpecificationCharacteristicValueOracle",
     "isDefault": true,
     "unitOfMeasure": "GB",
     "valueType": "number",
     "validFor": {"endDateTime": "2999-12-31T00:00:00.000Z", "startDateTime": "2021-01-01T00:00:00.000Z"},
     "value": "32",
     "@baseType": "ProductSpecificationCharacteristicValue",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe"
     },
     ],
     "@baseType": "ProductSpecificationCharacteristicValueUse",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe"
     },
     {
     "@type": "ProductSpecificationCharacteristicValueUseOracle",
     "description": "Brand",
     "name": "Brand",
     "valueType": "string",
     "isConfigurable": false,
     "productSpecCharacteristicValue": [
     {
     "@type": "ProductSpecificationCharacteristicValueOracle",
     "isDefault": true,
     "value": "Mustang",
     "@baseType": "ProductSpecificationCharacteristicValue",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe"
     }
     ],
     "@baseType": "ProductSpecificationCharacteristicValueUse",
     "@schemaLocation": "https://SERVER-INFO/productCatalogManagement/v4/productOffering/describe"
     }
     ],
     "validFor": {"startDateTime": "2021-01-01T00:00:00.000Z"},
     "isSellable": true,
     "banner": [],
     "version": "1",
     "lastUpdate": "2021-04-08T12:15:57.000Z",
     "name": "Mustang 11",
     "category": [
     {
     "id": "0CX-1X8K00",
     "href": "https://SERVER-INFO/productCatalogManagement/v4/category/0CX-1X8K00",
     "name": "Mustang"
     },
     {
     "id": "0CX-1X8JZX",
     "href": "https://SERVER-INFO/productCatalogManagement/v4/category/0CX-1X8JZX",
     "name": "Smart Phone"
     }
     ],
     "compatibleOffering": []
    }
```

Here's the simplified output after transforming the Get Product API response:

```
   {
    "productDetails": {
    "id": "0CX-1X8P2O",
    "brand": "Mustang",
    "name": "Mustang 11",
    "sku": [
    {
    "sku": "M110R-1WFGT9-25",
    "attributes": [
    {
    "name": "Model",
    "value": "Mustang 11"
    },
    {
    "name": "Capacity",
    "value": "32",
    "unitOfMeasure": "GB"
    },
    {
    "name": "Color",
    "value": "Red",
    "valueCode": "#FF0000"
    }
    ]
    },
    {
    "sku": "M110B-1WFGT9-13",
    "attributes": [
    {
    "name": "Model",
    "value": "Mustang 11"
    },
    {
    "name": "Capacity",
    "value": "32",
    "unitOfMeasure": "GB"
    },
    {
    "name": "Color",
    "value": "Black",
    "valueCode": "#000000"
    }
    ]
    }
    ],
    "description": "Say hello, to the Mustang 11! A triple-camera system provides gorgeous high fidelity
    images. This model has the speedy chip that has the critics in awe. And it features the highest-quality
    video in a smartphone, so your memories look better than ever!",
    "configuration": [
    {
    "name": "Color",
    "isConfigurable": true,
    "valueType": "string",
    "configurationOptions": [
    {
    "isDefault": true,
    "value": "Red",
    "valueCode": "#FF0000"
    },
    {
    "isDefault": false,
    "value": "Black",
    "valueCode": "#000000"
    }
    ]
    },
    {
    "name": "Capacity",
    "isConfigurable": true,
    "valueType": "number",
    "configurationOptions": [
    {
    "isDefault": true,
    "value": "32",
    "unitOfMeasure": "GB"
    }
    ]
    }
    ],
    "hiddenConfiguration": [
    {
    "name": "Brand",
    "description": "Brand",
    "isConfigurable": false,
    "valueType": "string",
    "configurationOptions": [
    {
    "isDefault": true,
    "value": "Mustang",
    "valueCode": ""
    }
    ]
    }
    ]
    },
    "productPricing": {
    "pricingCharacteristics": ["Capacity"],
    "pricingMap": {
    "32": "$599.00"
    }
    },
    "selectedProductConfiguration": {
    "id": "0CX-1X8P2O",
    "configurationOptions": [
    {
    "name": "Color",
    "value": "Red",
    "valueCode": "#FF0000"
    },
    {
    "name": "Capacity",
    "value": "32",
    "unitOfMeasure": "GB"
    }
    ]
    },
    "compatiblePlans": []
   }
```