Custom Property Types & Editors — Quick Start Guide

for Experience Builder LWR Sites

Summer ’23 Beta

Last Updated: 1 June 2023

Ideas? Feedback?

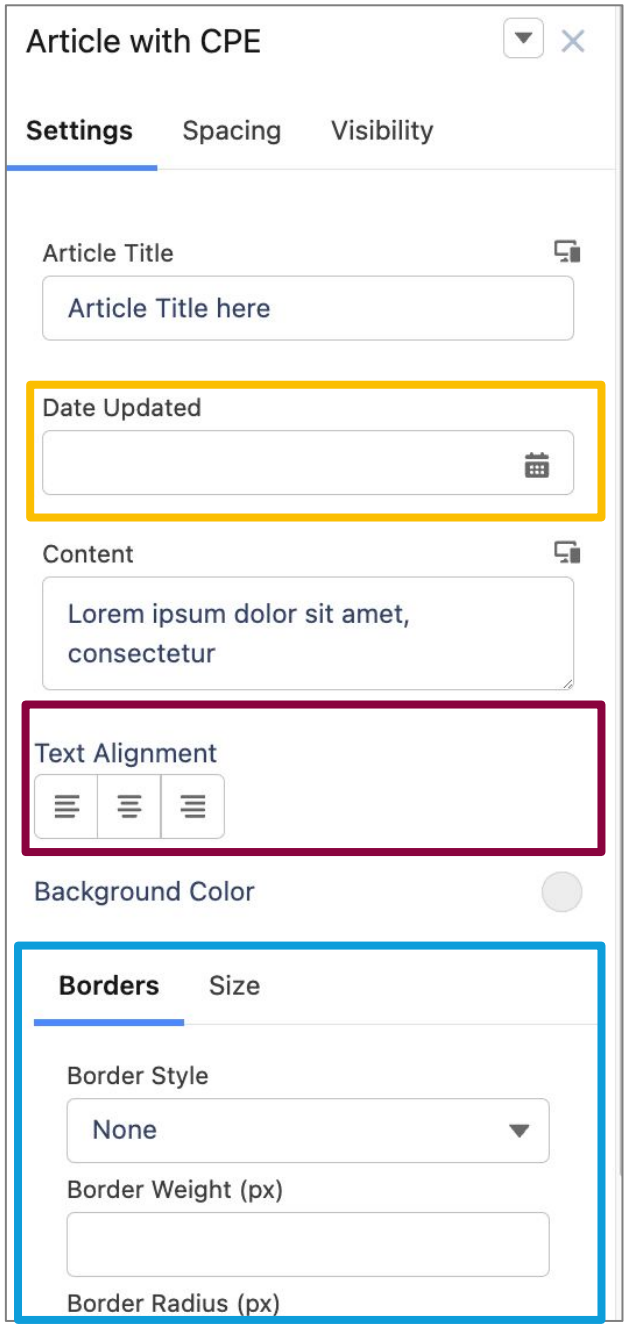
Let us know at the Experience Cloud Trailblazer Community

https://sfdc.co/b2nwQ3

Contents

Step-by-Step Walkthrough

[Download the sample component]



Our desired component

Property panel for the component

1 **articleDate**
Has a Lightning property type with validation to ensure that users enter a date

2 **textAlignment**
Has a custom property editor for text alignment

3 **layoutProperties**
Has a custom property type with multiple properties and a tab layout

1. articleDate

Reference lightning__dateType, one of the new Lightning property types, in our component

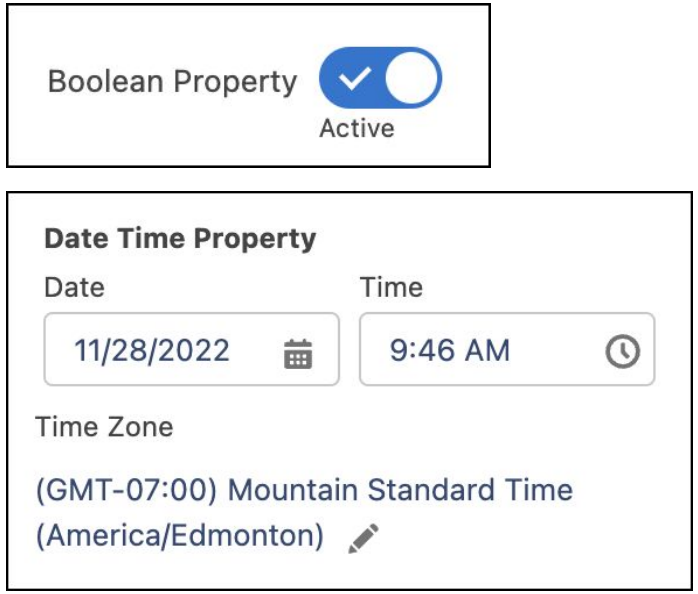
Previously, the `type` property in a component's js-meta.xml file was limited to String, Integer, Boolean, ContentReference, or Color.

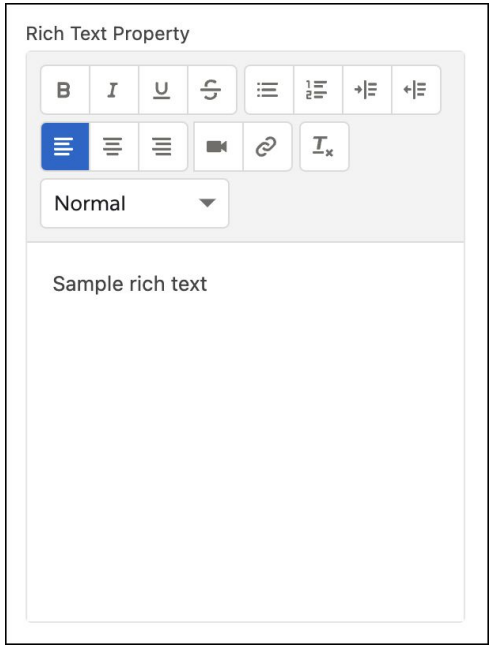
Additional Lightning property types are now available. Each Lightning property type includes a default property editor, so if you use one of these property types, you don't have to create a custom property editor yourself.

Available Lightning property types

- `lightning__booleanType`
- `lightning__dateType`
- `lightning__dateTimeType`
- `lightning__integerType`
- `lightning__multilineTextType`
- `lightning__numberType`
- `lightning__richTextType`
- `lightning__textType`
- `lightning__urlType`

Examples of the default property editors available with Lightning property types





Documentation: Lightning Property Types, Reference a Custom Property Type by Type Attribute

Reference lightning__dateType, one of the new Lightning property types, in our component

For the `articleDate` property, we're using `lightning__dateType`.

articlewCPE.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
   <apiVersion>58.0</apiVersion>
   <isExposed>true</isExposed>
   <targets>
       <target>lightningCommunity__Page</target>
      <target>lightningCommunity__Default</target>
  </targets>
  <targetConfigs>
      <targetConfig targets="lightningCommunity__Default">

     <!--Article Content-->
       <property type="String" name="articleTitle" label="Article Title" default="Article Title here"
screenResponsive="true" exposedTo="css" />
       <property type="lightning__dateType" name="articleDate" label="Date Updated" />
       <property type="lightning__multilineTextType" name="articleContent" label="Content"
       default="Lorem ipsum dolor sit amet, consectetur" screenResponsive="true" exposedTo="css" />

       </targetConfig>
  </targetConfigs>
  <masterLabel>Article with CPE</masterLabel>
</LightningComponentBundle>
```

Documentation: Lightning Property Types, Reference a Custom Property Type by Type Attribute

2. textAlignment

Step 1. Create a Lightning web component to act as the custom property editor for textAlignment

Custom property editors are Lightning web components (LWCs). To enable an LWC to act as a custom property editor:

1. Include these public properties in your custom property editor's javascript file: **`label`**, **`description`**, **`value`**, **`errors`**.
2. Use the **`lightning__PropertyEditor`** target in the custom property editor's js-meta.xml file.
3. Throw a single CustomEvent called **`valuechange`** in the custom property editor's .js file to send the value to Experience Builder

alignmentCPE is the LWC that will act as the custom property editor in our component (articlewCPE).

Documentation: [Create a Custom Property Editor]

Step 2. Reference alignmentCPE as the property editor for textAlignment in our component

In the js-meta.xml file of our component, use the **`editor`** attribute to reference the `fullyQualifiedName` string of the custom property editor that we created in step 1.

articlewCPE.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>58.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightningCommunity__Page</target>
       <target>lightningCommunity__Default</target>
  </targets>
  <targetConfigs>
      <targetConfig targets="lightningCommunity__Default">
      <!--Article Content -->
          . . .  . . .
       <!--Style Properties-->
       <property type="String" name="textAlignment" label="Text Alignment" editor="c/alignmentCPE"/>
       <property type="Color" name="backgroundColor" label="Background Color" />

       </targetConfig>
  </targetConfigs>
  <masterLabel>Article with CPE</masterLabel>
</LightningComponentBundle>
```

Documentation: [Reference a Custom Property Editor by Editor Attribute]

3. layoutProperty

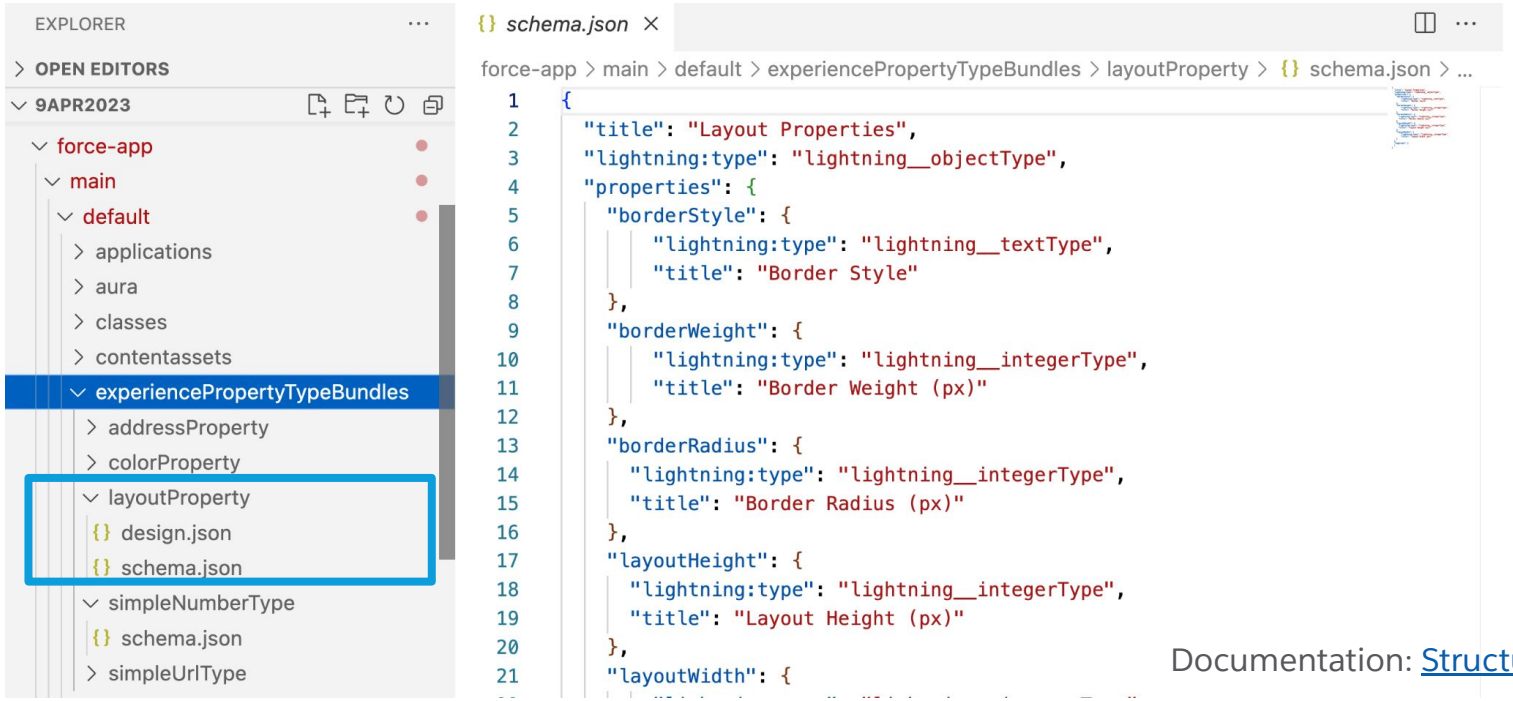
Step 1. Create an ExperiencePropertyTypeBundle

ExperiencePropertyTypeBundle is a new metadata type that describes property types.

Each custom property type in the **`experiencePropertyTypeBundles`** folder has a **`propertyTypeName`** folder that is named for the property type. Each **`propertyTypeName`** folder contains a JSON file or files that define the property type.

- `schema.json`: JSON schema that drives the property type validation
- `design.json`: optional file that provides the layout and property editor information for that property type

This experiencePropertyTypeBundles folder contains multiple custom property types, including **layoutProperty**



Documentation: [Structure of the ExperiencePropertyTypeBundle]

Step 2. Define the schema.json file for layoutProperty

For custom property types, Salesforce uses JSON Schema to define the data type and data structure of Lightning web component properties. We use JSON Schema so we can ensure that the structure of the component's property values is valid.

The schema delineates a specific data structure and type for that property, so you can build around that information.

Note: The `lightning:type` field in the schema.json file can reference only standard Lightning property types and the custom property types that you create.

The schema.json file for layoutProperty

schema.json

force-app > main > default > experiencePropertyTypeBundles > layoutProperty > schema.json > ...

```
{
  "title": "Layout Properties",
  "lightning:type": "lightning__objectType",
  "properties": {
    "borderStyle": {
      "lightning:type": "lightning__textType",
      "title": "Border Style"
    },
    "borderWeight": {
      "lightning:type": "lightning__integerType",
      "title": "Border Weight (px)"
    },
    "borderRadius": {
      "lightning:type": "lightning__integerType",
      "title": "Border Radius (px)"
    },
    "layoutHeight": {
      "lightning:type": "lightning__integerType",
      "title": "Layout Height (px)"
    },
    "layoutWidth": {
      "lightning:type": "lightning__integerType",
      "title": "Layout Width (px)"
    }
  },
  "required": [
  ]
}
```

Documentation: [The Schema.json File]

Step 3. Define the design.json file for layoutProperty

Design.json is an optional file where you can specify the layout of your property type–for example, in an accordion section, or on tabs. You can also specify an editor override–a property editor that overrides the default editor for any Lightning property type or custom property type.

To arrange layoutProperty on two tabs, we use lightning/tabSetLayout in the `view` keyword of the design.json file.

Syntax:

- lightning/tabSetLayout can take only lightning/tabLayout as its children.
- lightning/tabLayout, in turn, can use only lightning/propertyLayout as children.

The design.json file for layoutProperty

design.json

force-app > main > default > experiencePropertyTypeBundles > layoutProperty > design.json >

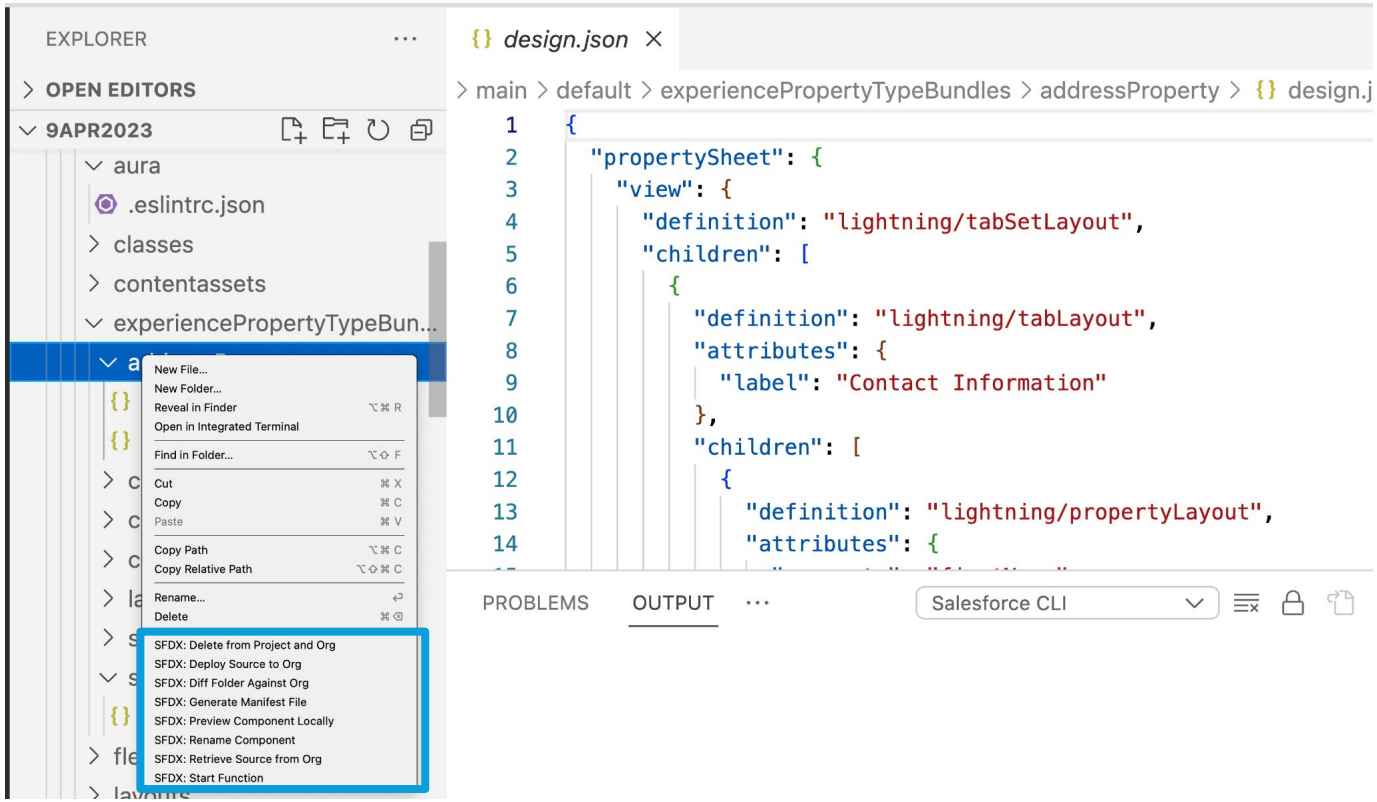
```
{
  "propertySheet": {
    "propertyRenderers": {
      "borderStyle": {
        "definition": "c/borderStyleDropdownCPE"
      }
    },
    "view": {
      "definition": "lightning/tabSetLayout",
      "children": [
        {
          "definition": "lightning/tabLayout",
          "attributes": {
            "label": "Borders"
          },
          "children": [
            {
              "definition": "lightning/propertyLayout",
              "attributes": {
                "property": "borderStyle"
              }
            },
            {
              "definition": "lightning/propertyLayout",
              "attributes": {
                "property": "borderWeight"
              }
            },
            {
              "definition": "lightning/propertyLayout",
              "attributes": {
                "property": "borderRadius"
```

Documentation: [The Design.json File > lightning/tabSetLayout]

Step 4. Deploy the layoutProperty custom property type

You can now use SFDX project commands in the Salesforce CLI to deploy experiencePropertyTypebundles–for example, `sfdx project deploy start [filepath]`. You can also use Salesforce Extensions for Visual Studio Code to deploy your component in VS Code.

Deploying the addressProperty using Salesforce Extensions for Visual Studio Code



Documentation: Use SFDX or Metadata API to Deploy ExperiencePropertyTypeBundles

Step 5. Reference layoutProperty in our component

In the js-meta.xml file of our component, use the type attribute to reference the fullyQualifiedName string of the custom property type.

articlewCPE.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
   <apiVersion>58.0</apiVersion>
   <isExposed>true</isExposed>
   <targets>
       <target>lightningCommunity__Page</target>
      <target>lightningCommunity__Default</target>
  </targets>
  <targetConfigs>
      <targetConfig targets="lightningCommunity__Default">
       <!--Article Content -->
           . . .  . . .
       <!--Style Properties -->
        . . .  . . .
       <!--Layout Properties-->
       <property name="layoutProperties" type="c__layoutProperty" label="Layout" />

       </targetConfig>
  </targetConfigs>
  <masterLabel>Article with CPE</masterLabel>
</LightningComponentBundle>
```

Documentation: [Reference a Custom Property Type by Type Attribute]

Limitations in the Summer ’23 Beta

Limitations in your component js-meta.xml if it references a custom property editor or type (1/2)

1. You can use custom property types and property editors only in Lightning web components that target `lightningCommunity__Default`. You can't use them in components that target `lightningCommunity__Page_Layout`, `lightningCommunity__Theme_Layout`, or other targets that are unrelated to Experience Builder. If you do, deployment of the component fails.

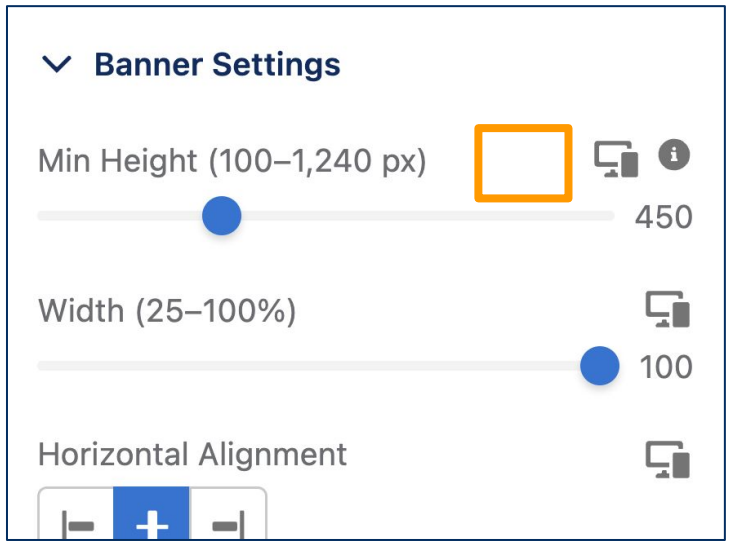
2. If a property references a custom property type (for example, `<property name="myProperty" type="c__customType" />`), you can't specify any of these attributes in the property; if you do, deployment fails.
   a. `datasource`
   b. `max`
   c. `min`
   d. `placeholder`

Documentation: Considerations and Limitations for Custom Property Types and Editors

Limitations in your component js-meta.xml if it references a custom property editor or type (2/2)

3. If a property references a custom property editor, you can specify `screenResponsive= "true"` for that property (for example, `<property name= “myProperty” editor= "c/myCustomEditor" screenResponsive= "true">`). However, in the component property panel, the screen-responsive icon doesn’t appear next to that property’s label.

Expected behavior for screenResponsive = true



Documentation: [Considerations and Limitations for Custom Property Types and Editors]

ExperiencePropertyTypeBundle Limitations (1/2)

1. You can't update an existing custom property type. You can only create and delete one. This limitation also applies to updating a package that contains custom property types.

2. You can't nest object properties in a custom property type schema. In a given schema, a custom property type of the data type object can't contain child properties of the data type object. Instead, create a custom property type for the child object property and reference it in the current schema using the lightning:type keyword. Here's an example of an invalid schema.

```
{
 "title": "Invalid Nested ObjectType",
 "lightning:type": "lightning__objectType",
 "properties": {
   "postalCode": {
     "title": "Invalid Nested ObjectType Property",
     "lightning:type": "lightning__objectType",
     “properties”: {
        “propertyName”: { … }
     }
   }
 }
}
```

Documentation: [Considerations and Limitations for Custom Property Types and Editors]

ExperiencePropertyTypeBundle Limitations (2/2)

3. Circular references in a property type are not blocked. A circular reference occurs when property type A refers to property type B, and property type B refers back to property type A.

   You aren't prevented from saving a property type with a circular reference, but these references cause gacks when they render a property editor. You also can't delete circularly referenced property types. For example, you can't delete property type A when property type B refers to A, and you can't delete property type B when property type A refers to B.

Documentation: Considerations and Limitations for Custom Property Types and Editors

Notable Changes from the Spring ’23 Developer Preview

1. Managed and unmanaged packages are now supported

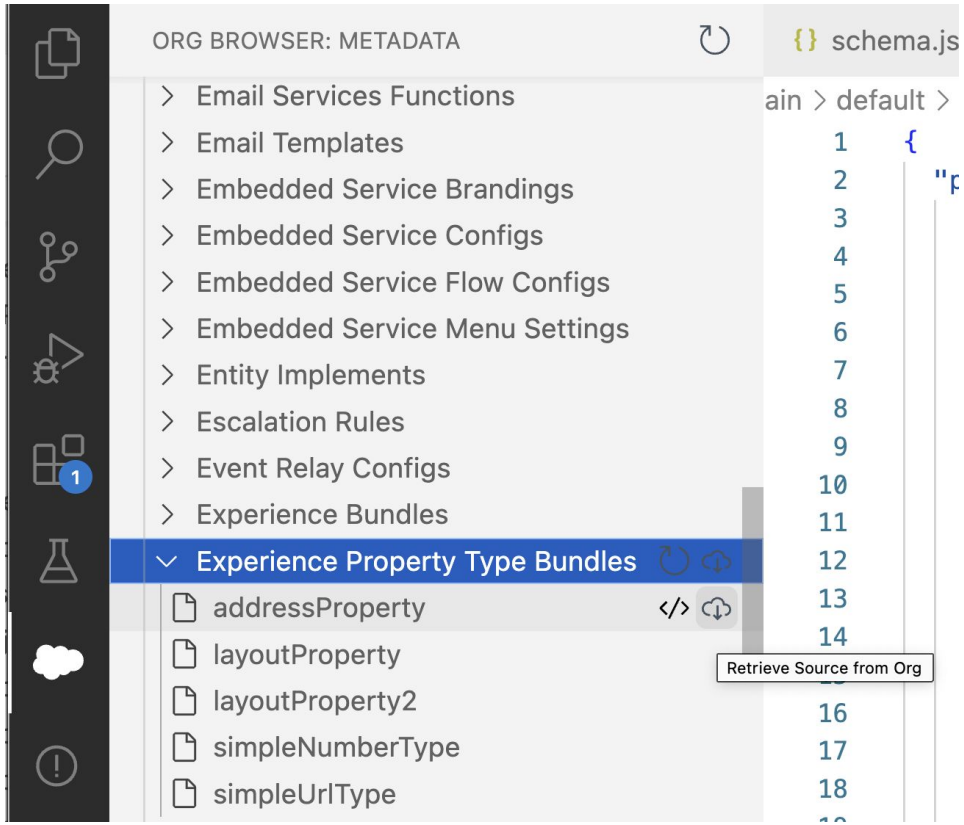
Packaging is now supported for the `ExperiencePropertyTypeBundle` metadata type, and for Lightning web components that reference a custom property type or property editor.

- Updates of managed packages containing `ExperiencePropertyTypeBundles` are not supported. If you upgrade a package that contains a custom property type or editor, users who installed this package must uninstall it and reinstall the updated package. We recommend that you don't include `ExperiencePropertyTypeBundles` in non-beta managed packages.
- We recommend that you delete the custom components, property types, and editors created in Spring '23 and redeploy them in order to get the new features in Summer '23.

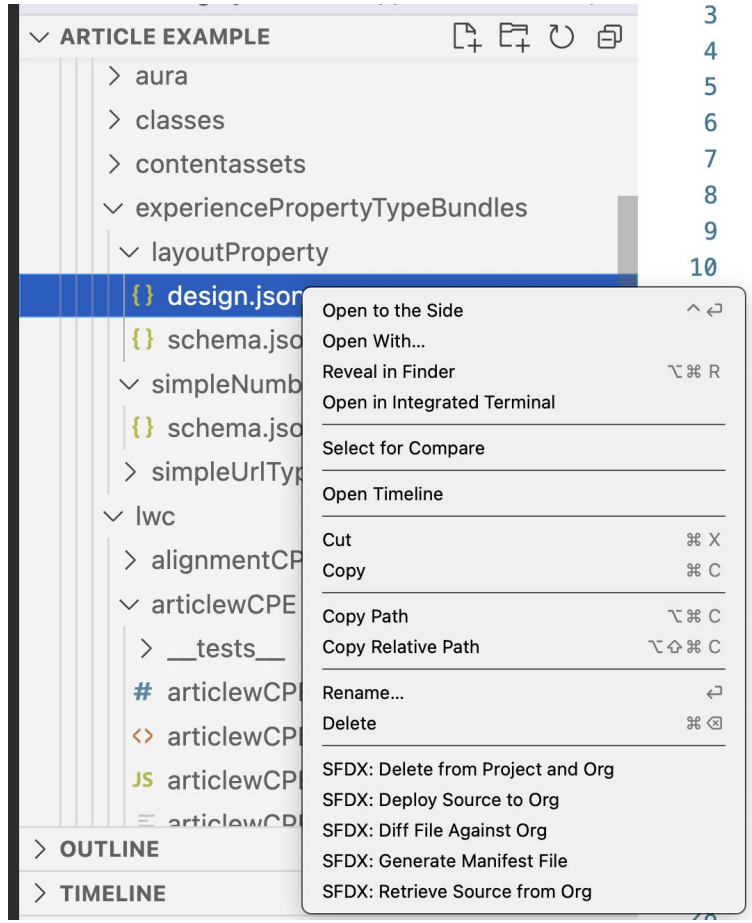
Documentation: Important Changes from the Developer Preview

2. Added support for CLI source commands and VSCode Salesforce Extensions

experiencePropertyTypeBundles show up as a metadata type in org browser



Salesforce Extensions metadata operations work for experiencePropertyTypeBundles



Documentation: [Important Changes from the Developer Preview]

3. New syntax in js-meta.xml to reference a custom property editor

Use “__” as the separator, instead of “/”.

Spring ’23 Developer Preview

```
<property name="myProperty"
type="String"
editor="c__myCustomEditor"/>
```

Summer ’23 Beta

```
<property name="myProperty"
type="String"
editor="c/myCustomEditor"/>
```

Documentation: Important Changes from the Developer Preview

4. New syntax in design.json to define a property renderer override for the entire property

Use “$” to define that a custom property editor acts as the editor for all the properties in the type.

Spring ’23 Developer Preview:

```
{
"propertySheet": {
   "view": {
      "definition": "c/myCustomMapEditor"
  }
}
```

Summer ’23 Beta:

```
{
 "propertySheet": {
   "propertyRenderers": {
     "$": {
       "definition": "c/myCustomMapEditor"
     }
   }
 }
}
```

Documentation: [Important Changes from the Developer Preview]

5. Localization support for ExperiencePropertyTypeBundle

You can add the translatable attribute to properties with a custom property type whose underlying JSON schema type is string. You can translate the values in these properties into all your site languages.

```
myCustomType > schema.json:
{
      "lightning:type": "lightning__textType",
}
```

```
js-meta.xml:
<property name="myProperty" type="c__myCustomType" translatable=true"/>
```

Custom labels are now supported for a custom property type's `title` and `description` fields. You can use custom labels in Salesforce Setup to add translations of the title and description for all your site languages.

Documentation: Important Changes from the Developer Preview

Thank you