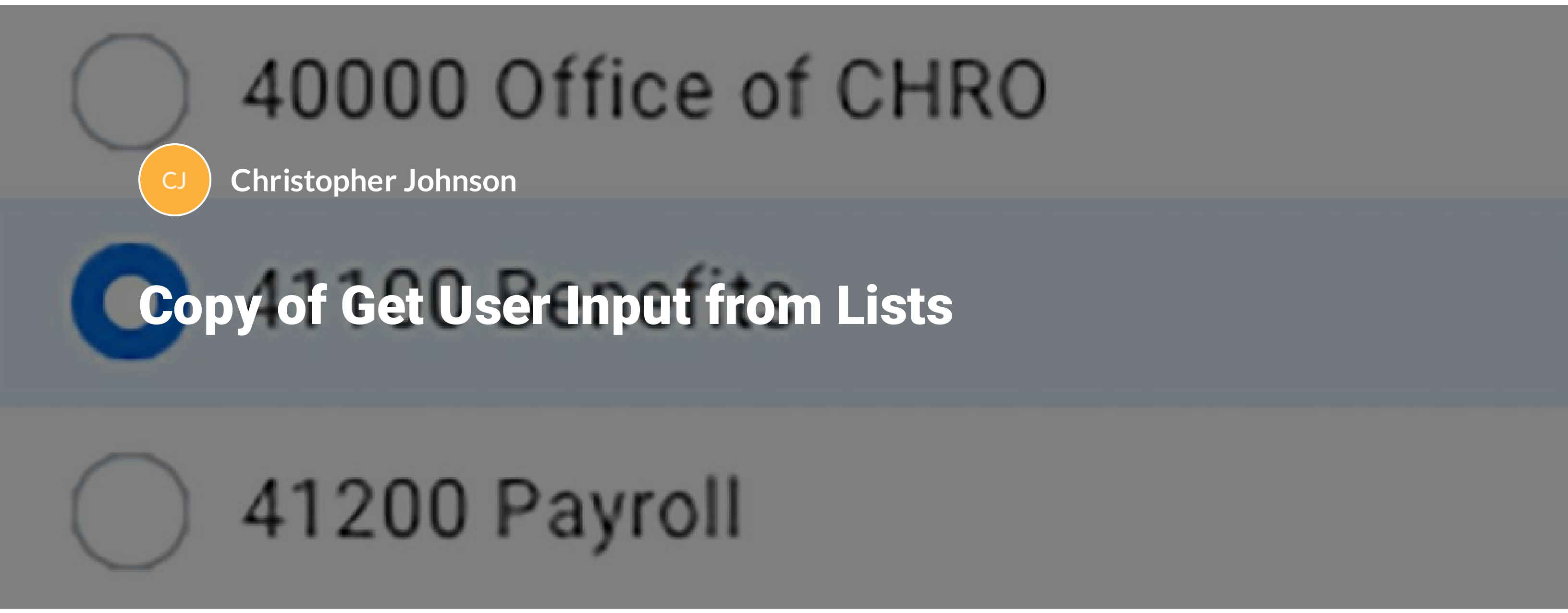


CJ Christopher Johnson

# Copy of Get User Input from Lists

You are probably already familiar with lists that are used for navigation or for collecting information from users. You've seen pull-down menus with options for navigating to a page or view within an app. Online questionnaires, surveys, and other types of forms commonly use dropdown lists, radio buttons, and check boxes to make it easy for users to select and input data. When used the right way, lists are a compact and intuitive way for users to interact with your app while giving you an effective way to manage the data that they input into the system.

In this tutorial, you'll learn how to configure a dropDownButton that will enable users to navigate to different pages in your app. You'll also learn how to define a flow in which users are prompted to select inputs from dropdown list, instanceList, radioGroup, and checkBoxList.

## Prerequisites

Before starting this tutorial, you should complete the following tutorials:

1. Getting Started with Workday Extend
2. Create an App
3. Access Workday Data

Estimated Time to Complete: 40 minutes

COURSE OVERVIEW

*Lesson 1 of 6*

# Course Overview

CJ Christopher Johnson

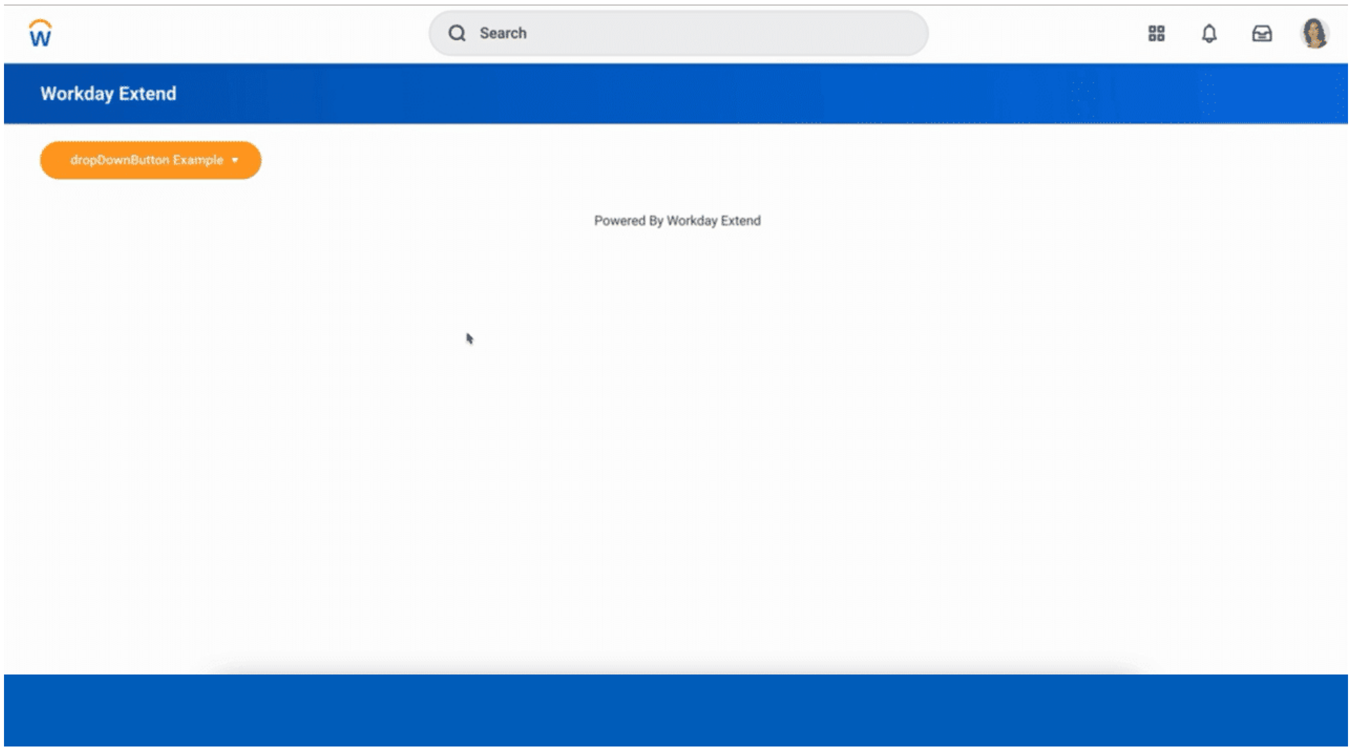
Lists are a great way to create a good user experience. With Workday Extend, you can use lists for navigation between pages and you can use lists to prompt users to input data.

In this tutorial, you'll learn how to use a **dropDownButton** for navigation between pages in your app. Then you'll use Workday Extend's **dropdown**, **instanceList**, **radioGroup**, and **checkBoxList** widgets to create a series of prompts. The user moves from one prompt to another to create a collection of related inputs.

You'll also define a **flow** and use **flowVariables** to persist the data throughout the flow so that it can be accessed by other pages.

## Look at What You'll Build

Watch the video to see how you'll use lists in your app to create navigation and to get user inputs.



## Learning Objectives

When you've completed this tutorial, you will be able to:

1. Use **dropDownButton** widgets to enable users to navigate between pages in an app.
2. Use **instanceList** and **instanceListLoopTag** to populate **dropdown**, **radioGroup**, and **checkBoxList** widgets.
3. Use a **flowVariable** to bind, submit, and persist values from different list widgets.

## How you'll learn

### Activity 1: Create a new app with dropdown button navigation

In this activity, you'll:

1. Create a new app.
2. Add three view PMD pages and one edit PMD page with provided endPoints and outboundEndPoints..
3. Add navigation to the PMD pages using dropDownButtons.
4. Validate and deploy the app.

### Activity 2: Add lists that will get data from users

In this activity, you'll:

1. Add dropdown, radioGroup, and checkBoxList widgets to the getUserInput PMD page.
2. Use instanceList to populate the list values and bind values from the dropdown list and the radioGroup list.
3. Use instanceListLoopTag to populate checkBoxList values.
4. Validate and deploy the app.

### Activity 3: Display data that users input with lists

In this activity, you'll:

1. Add tags to display chosen values bound from each list widget.
2. Add flowVariables.
3. Validate and deploy the app.

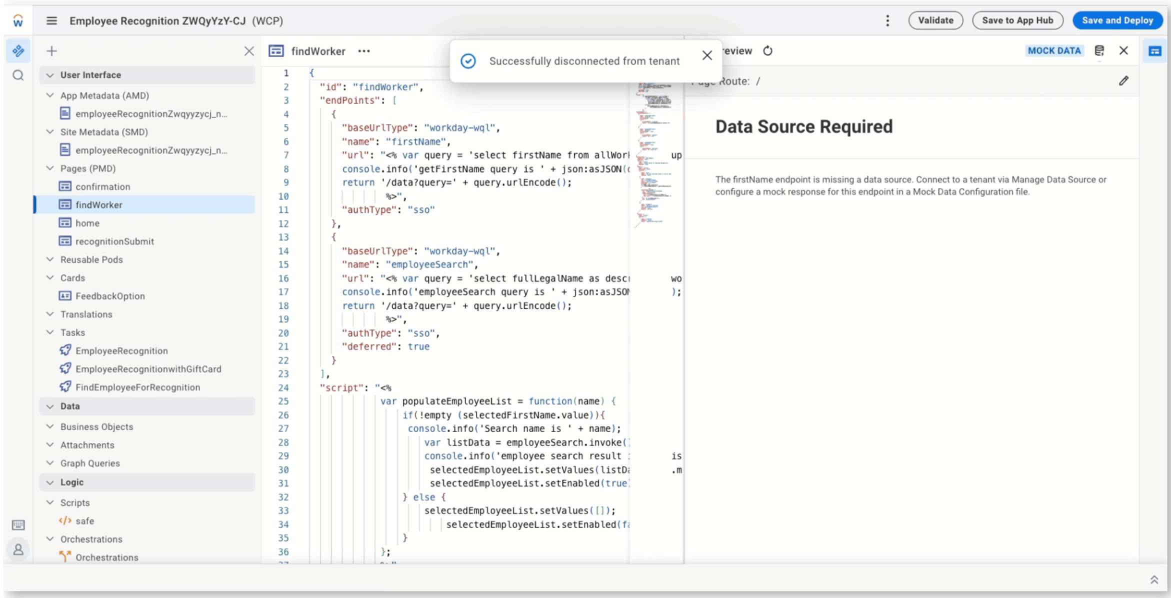
## Increase Your Velocity with App Preview

App Builder's App Preview enables you to validate, preview and test your PMD pages without deploying your app to a tenant. It's a big time-saver and a great way to do iterative development. You can *see* what your changes look like in your app against your actual tenanted data.

**Note: This tutorial assumes you have App Preview open and will test your changes in App Preview before Saving and Deploying your work.**

Some things to remember when using App Preview:

- App Preview enables you quickly validate and test changes, but it does not save those changes to App Hub for you. Any changes you test with App Preview are saved to your current session. Make it a habit to **Save to App Hub** when you are satisfied with your changes and want to save your work.
- App Preview accesses inbound endPoints or outboundEndPoints using a **Data Source**. You can either:
    - **Configure Mock Data** to precisely control what your page will display using static data.
    - **Connect to a Tenant** to actually fire the inbound and outbound endpoints in your app. To Connect to Tenant, click the **Manage Data Source** icon and then log in to a development or implementation tenant.
- The App Preview connection to your tenant will time-out after approximately 15 minutes, so you may get an HTTP error. No worries. Just click the **Manage Data Source icon** again and reconnect.



Successfully disconnected from tenant

```
{
  "id": "findWorker",
  "endPoints": [
    {
      "baseUrlType": "workday-wql",
      "name": "firstName",
      "url": "<% var query = 'select firstName from allWor
      console.info('getFirstName query is ' + json:asJSON(
      return '/data?query=' + query.urlEncode();
                %>",
      "authType": "sso"
    },
    {
      "baseUrlType": "workday-wql",
      "name": "employeeSearch",
      "url": "<% var query = 'select fullLegalName as desc
      console.info('employeeSearch query is ' + json:asJSO
      return '/data?query=' + query.urlEncode();
                %>",
      "authType": "sso",
      "deferred": true
    }
  ],
  "script": "<%
            var populateEmployeeList = function(name) {
              if(!empty (selectedFirstName.value)){
               console.info('Search name is ' + name);
                 var listData = employeeSearch.invoke(
                 console.info('employee search result
                  selectedEmployeeList.setValues(listD
                  selectedEmployeeList.setEnabled(true
              } else {
                 selectedEmployeeList.setValues([]);
                    selectedEmployeeList.setEnabled(f
              }
            };
```

## Data Source Required

The firstName endpoint is missing a data source. Connect to a tenant via Manage Data Source or configure a mock response for this endpoint in a Mock Data Configuration file.

Use the Manage Data Source icon to connect to a tenant and preview and test your app live from App Builder.

**Contact extend@workday.com for additional information**

**To ensure you have a fluid learning experience, we recommend you complete the practice activities in order, as they progressively build upon one another.**

## CHALLENGE 1: CREATE NAVIGATION WITH A DROP DOWN BUTTON

*Lesson 2 of 6*

# Introduction: Get Data from Lists

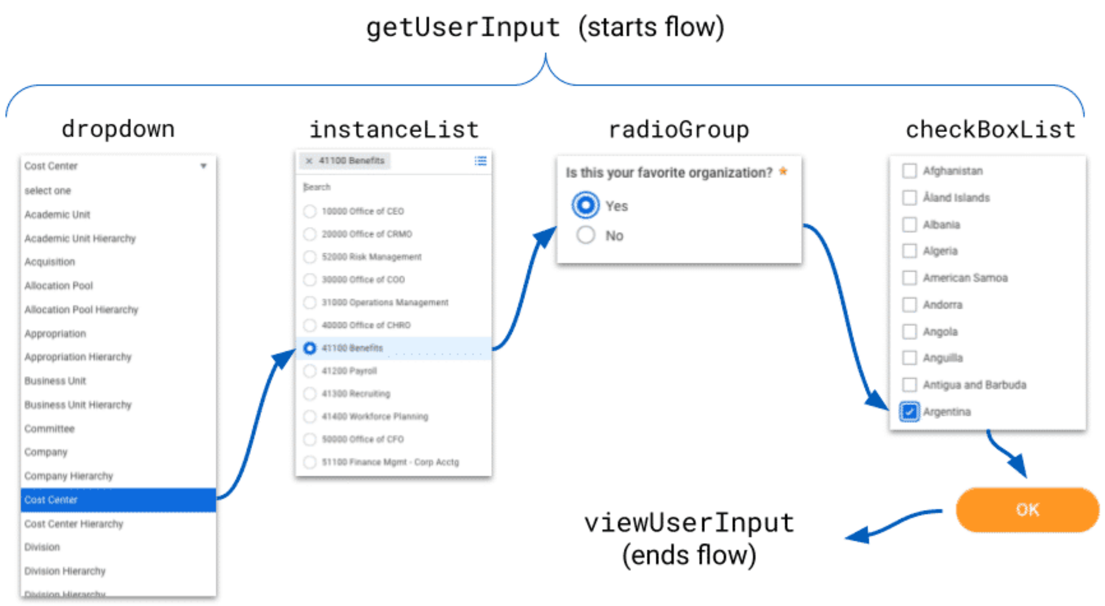
CJ Christopher Johnson

These days, a convenient way to collect information from a worker or a customer is to ask them to enter information online, usually in a questionnaire or some other form. There are challenges to doing it this way. The quality of the data that is collected is dependent on the user's experience. If it takes too long to complete a form or questions are hard to answer, the user could become frustrated and the quality of the data that is collected could be impaired.

This is why using lists to get are an efficient and effective way to get users' input. All they have to do is pick from a list of options.  It's a quick and easy way for the user to provide high-quality information while maintaining the integrity of the data.

In this lesson, you'll:

- To the **getUserInput** edit page, you'll add four list widgets: **dropdown**, **instanceList**, **radioGroup**, and **checkBoxList**.
- Employ different methods that will automatically populate each list with values that the user can select as inputs.
- Use **flowVariables** to persist the data inputs and make the data accessible to other pages in your app.
- Configure a **flow** and **flow steps** that will guide users from one page to another.



This illustration shows how you'll use flowVariables to persist data that a user selects from a series of lists and to send it from one page (getUserInput) to another page (viewUserInput).

## ACTIVITY 2.1: ADD DATA PROVIDERS AND DEFINE A FLOW

*Lesson 3 of 6*

# Activity 2.1: Add Data Providers and Define a Flow

CJ Christopher Johnson

The purpose of using lists for data input is to give users a list of options that they can easily search and select. You can manually hard-code the data into the list, but a best practice is to populate the lists using an existing database of values with which you can populate the lists.

Workday REST APIs are a convenient source of commonly used data, such as lists of organization-types, organization names, and country names. Your first step is to connect your app to the relevant Workday REST APIs.

In this activity, you'll:

- Add **dataProviders** to the **AMD file**.
- Define a **flow**.
- Add **EndPoints** and **outboundEndPoints** to the **getUserData** edit page.

START ACTIVITY

1

## Add Data Providers to Your App

Data providers will enable your app to get endpoint data and use it to populate the lists. To add dataProviders to your app:

1. Open your app’s **AMD file** and locate the "**applicationId**" tag.
2. After the "**applicationId**" tag**,** insert a **dataProviders** array**.**
3. In the dataProviders array, add the base URLs for the following Workday REST APIs:
   - Common (v1)
   - Person (v1)
   - Wql (v1)
4. Click **Save to App Hub** to validate and save your code changes.

```
  "dataProviders": [
    {
      "key": "workday-common",
      "value": "https://api.workday.com/common/v1/"
    },
    {
      "key": "WDAY-WQL",
      "value": "https://api.workday.com/wql/v1/"
    },
    {
      "key": "WDAY-Person",
      "value": "https://api.workday.com/person/v1/"
    }
 ]
```

Code Block — In your app's AMD file, this is how your dataProviders should appear for the base URLs.

## ADD INBOUND ENDPOINTS

2

## Add Inbound Endpoints to the getUserInput Page

In this task, we'll add three **endPoints** to the getUserInput page. The endPoints will fetch the data that we will use to populate the lists that will be displayed on the page.

1. Open the **getUserInput page** and find the **endPoints array**.
2. In the **endPoints array**, copy and paste the three endpoints from the code block below.
3. Click **Save to App Hub** to validate and save your code changes. **Note**: You will receive warnings that the endpoints are declared but not used in the PMD. This will be addressed in the later activities when we reference the endpoints in the list widgets that we'll add to the **getUserInput page**.

```
"endPoints" : [ {
  "name" : "orgTypes",
  "baseUrlType" : "workday-common",
  "url" : "organizationTypes",
  "authType" : "sso"
}, {
  "name" : "organizations",
  "deferred" : true,
  "baseUrlType" : "workday-common",
  "url" : "<% 'organizations?organizationType=' + id %>",
  "authType" : "sso"
}, {
  "name" : "countries",
  "baseUrlType" : "WDAY-Person",
  "url" : "countries?limit=10",
  "authType" : "sso"
} ],
```

Code Block - Copy and paste the endpoints into the getUserInput page. The endpoints will fetch the data that will be used to populate the lists.

## DEFINE THE PAGE FLOW IN THE AMD

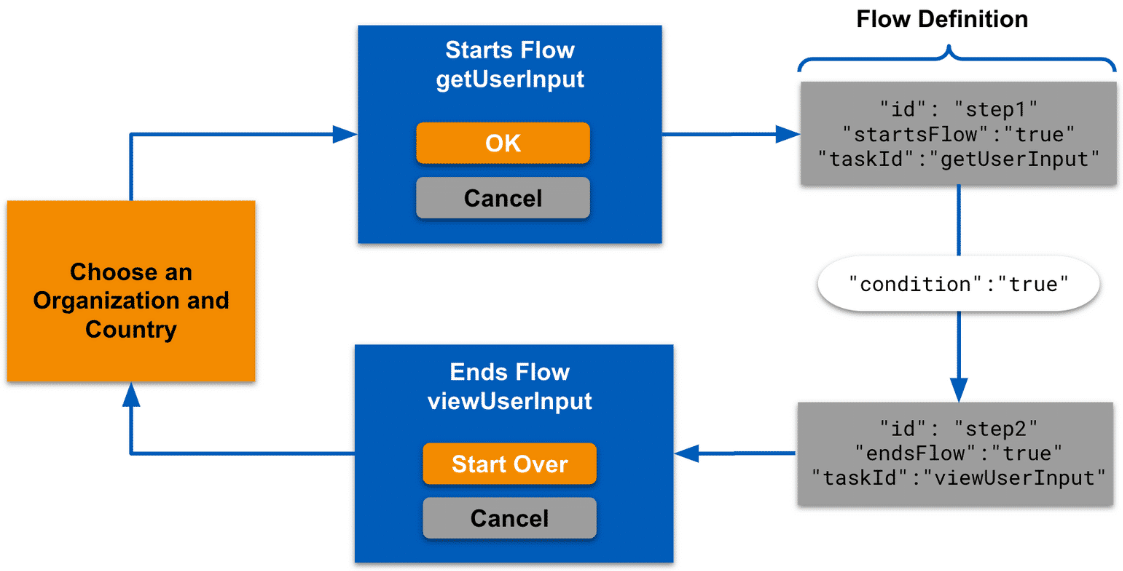
3

### Define a Flow in the AMD

Because your app will get user input from a sequence of lists displayed on the getUserData edit page, you'll need to define flow with **flow steps**. A flow is a sequence of pages or tasks that guide users through a multistep transaction.

The figure below illustrates how the flow you'll create works. The **flow** will start on the **getUserInput edit page**, which is step 1. The flow will override the edit page's default **OK button** so that it will redirect the user to step 2, which is the **viewUserInput** page and where the flow ends. The transition from one step to another is handled by conditional logic.

As you'll see in the next task, an outboundVariable will persist the data and make it accessible to other pages in the app during the lifecycle of the flow.



Define the flow:

1. In your app's **AMD file**, find the the **"appProperties" array**.
2. After the **"appProperties" array**, add a **"flowDefinitions" array**.
3. In the **"flowDefinitions" array**, define a new flow with an **"id"** set to **"flowNewPage"** and an empty **"flowSteps"** array.
4. In the **"flowSteps" array**, define step 1 of the flow using the attributes listed below. The getUserInput will start the flow.
   - "id" : "step1"
   - "startsFlow" : true"
   - "taskId" : "getUserInput"
5. In **step1**, add a transition that will define the condition in which the user will go to **step2** in the flow.  Add a **"transitions" array** to **step1,** and in the array, add an object with these attributes:
   - "order" : "a",
   - "value" : "step2"

    - "condition" : "true"

6. After **step1**, add **step 2** with the attributes listed below. The **viewUserInput** page will end the flow.
    - "id" : "step2"
    - "startsFlow" : true"
    - "taskId" : "viewUserInput"

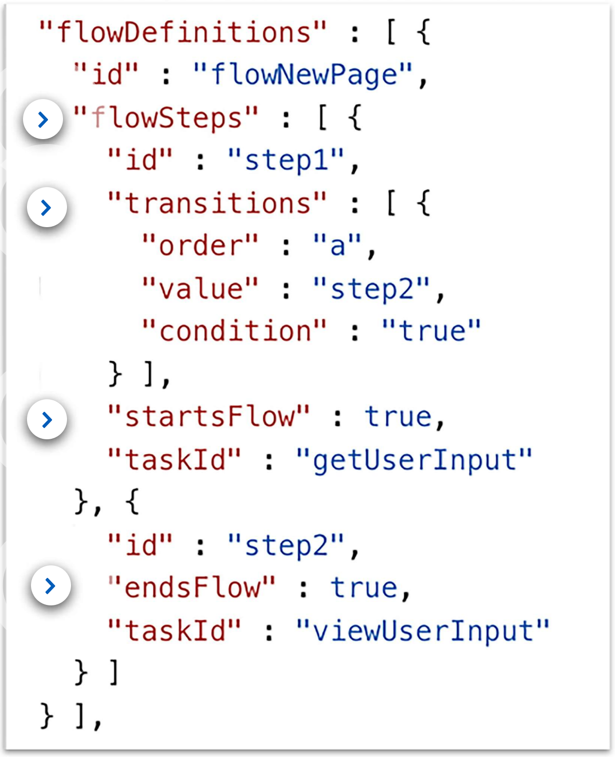
7. Click **Save to App Hub** to validate and save your code changes.

```
"flowDefinitions": [
    {
      "id": "flowNewPage",
      "flowSteps": [
        {
          "id": "step1",
          "transitions": [
            {
              "order": "a",
              "value": "step2",
              "condition": "true"
            }
          ],
          "startsFlow": true,
          "taskId": "getUserInput"
        },
        {
          "id": "step2",
          "endsFlow": true,
          "taskId": "viewUserInput"
        }
      ]
    }
  ],
```

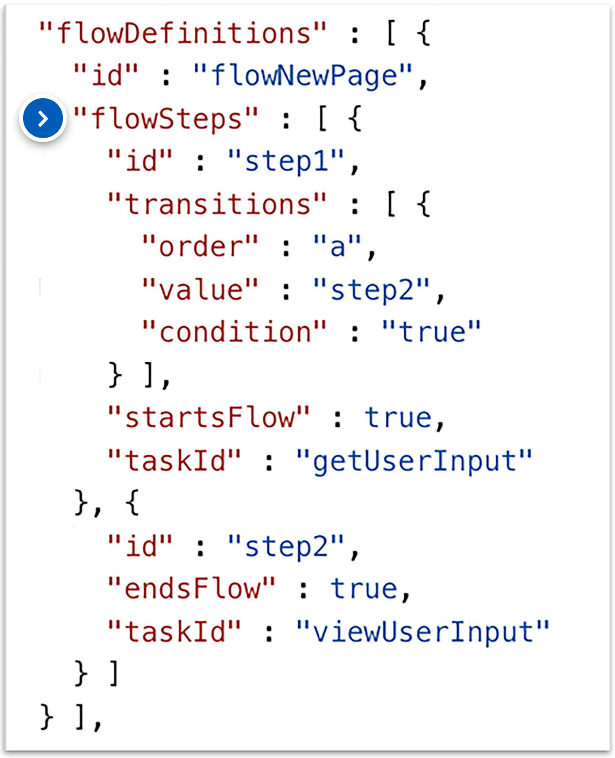
Code Block — Copy this code to add a flow definition with two steps.

## Annotated Flow Definition

A flow definition specifies the sequence of pages or tasks in a flow, and determines the conditions in which a transition is made from one page to another. The annotations on the image below describe the key element of a flow definition.



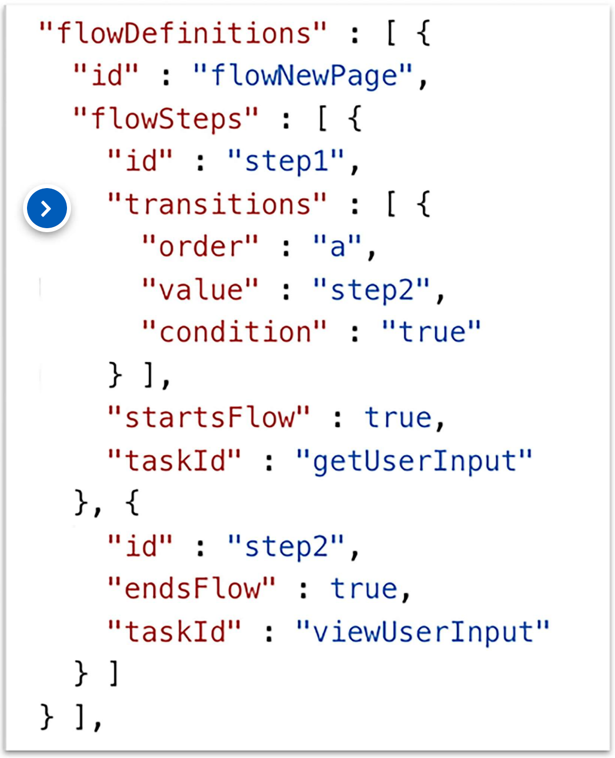
```
"flowDefinitions" : [ {
  "id" : "flowNewPage",
  "flowSteps" : [ {
    "id" : "step1",
    "transitions" : [ {
      "order" : "a",
      "value" : "step2",
      "condition" : "true"
    } ],
    "startsFlow" : true,
    "taskId" : "getUserInput"
  }, {
    "id" : "step2",
    "endsFlow" : true,
    "taskId" : "viewUserInput"
  } ]
} ],
```



```
"flowDefinitions" : [ {
  "id" : "flowNewPage",
  "flowSteps" : [ {
    "id" : "step1",
    "transitions" : [ {
      "order" : "a",
      "value" : "step2",
      "condition" : "true"
    } ],
    "startsFlow" : true,
    "taskId" : "getUserInput"
  }, {
    "id" : "step2",
    "endsFlow" : true,
    "taskId" : "viewUserInput"
  } ]
} ],
```

## flowSteps array

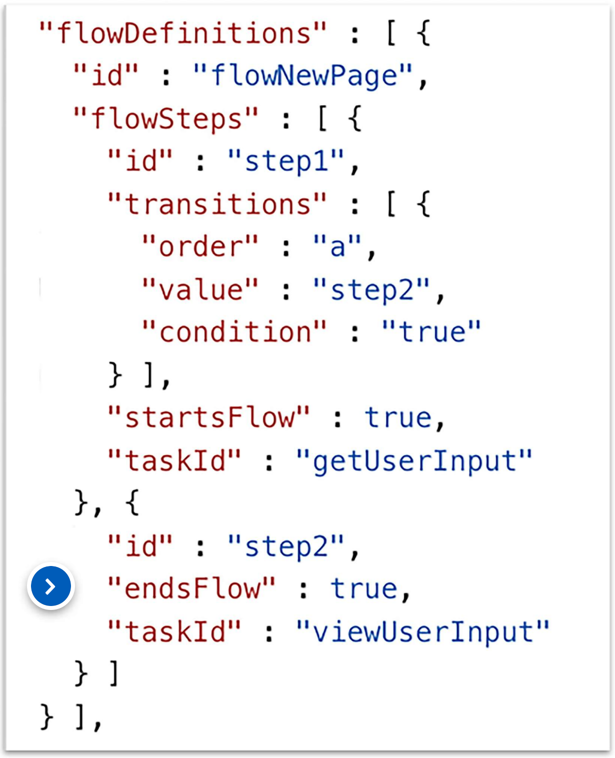
In the flowSteps array, you define the steps of your flow. A flow step defines each page or task in a flow.



```
"flowDefinitions" : [ {
  "id" : "flowNewPage",
  "flowSteps" : [ {
    "id" : "step1",
    "transitions" : [ {
      "order" : "a",
      "value" : "step2",
      "condition" : "true"
    } ],
    "startsFlow" : true,
    "taskId" : "getUserInput"
  }, {
    "id" : "step2",
    "endsFlow" : true,
    "taskId" : "viewUserInput"
  } ]
} ],
```

## Transition

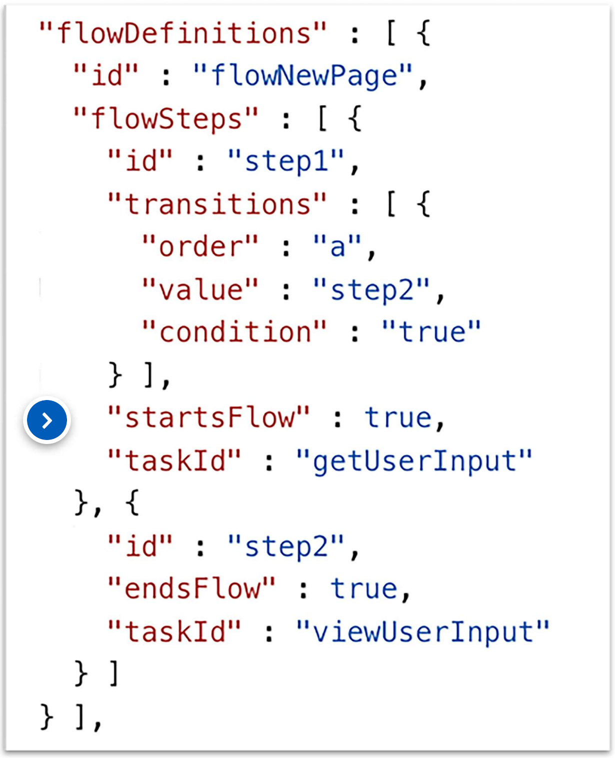
The **transitions** array defines the conditions for the next step in a flow.



```
"flowDefinitions" : [ {
  "id" : "flowNewPage",
  "flowSteps" : [ {
    "id" : "step1",
    "transitions" : [ {
      "order" : "a",
      "value" : "step2",
      "condition" : "true"
    } ],
    "startsFlow" : true,
    "taskId" : "getUserInput"
  }, {
    "id" : "step2",
    "endsFlow" : true,
    "taskId" : "viewUserInput"
  } ]
} ],
```

### endsFlow

The endsFlow attribute indicates that this is the last step in the flow and the taskId specifies the page.



```
"flowDefinitions" : [ {
  "id" : "flowNewPage",
  "flowSteps" : [ {
    "id" : "step1",
    "transitions" : [ {
      "order" : "a",
      "value" : "step2",
      "condition" : "true"
    } ],
    "startsFlow" : true,
    "taskId" : "getUserInput"
  }, {
    "id" : "step2",
    "endsFlow" : true,
    "taskId" : "viewUserInput"
  } ]
} ],
```

### startsFlow

The **startsFlow** attribute indicates that the page defined in the **taskId** beings the flow.

ADD OUTBOUND ENDPOINTS TO THE EDIT PAGE

4

## Add outboundData to the getUserInput Page

When you add **outboundData** to a page, you change the page from a view page to an edit page. The outbound endpoints in the *outboundEndPoints* array are used for submitting data to APIs as well as to other pages in your app.

The **outboundVariable** attribute, which is defined in the **outboundEndPoints** array, defines a collection of variables that represent the users data input. The outboundVariable persists the data so that it can be accessed by other pages in the app.

To add **outboundEndPoints** to the **getUserInput** page:

1. Open the **getUserInput page** and look for the **"id": "getUserInput"** tag, which you'll find at the top of the page.
2. After the **"id":"getUserInput"** tag, add an **outboundData** object with a **outboundEndPoints** array**.**
3. In the **outboundEndPoints** array, create an object with **"name"** set to **"transitionOutboundVars"** and **"type"** set to **"outboundVariable".**
4. Add a **variableScope** attribute and set it to **"flow"**. The **variableScope** attribute defines the scope of the **outboundData**. The value **"flow"** indicates that **flowVariables** are accessible to other pages during the lifecycle of the flow.
5. After the **variableScope** tag, add a **values** array.
6. In the **values** array, define the four **variables** that will represent the user's data input. Each of the four variable should include two attributes—**outboundPath** and **value**—as shown in this list:
   - "outboundPath": "orgType",  "value": "<% orgTypes.value %>"
   - "outboundPath": "organization",  "value": "<% instanceListOrgs.value %>"
   - "outboundPath": "favorite",  "value": "<% favoriteRadioGroup.value %>"
   - "outboundPath": "countries", "value": "<% testCheckboxList.value %>"
7. Click **Save to App Hub** to validate and save your code changes.

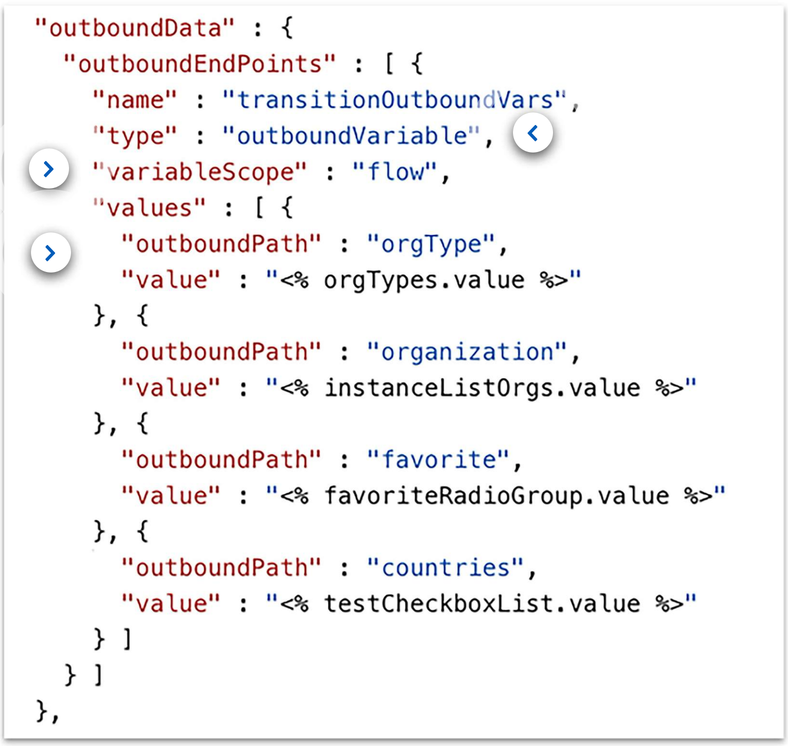
**Remember:** To make a page an edit page, you must add add **outboundData** or add the attribute **"pageType": "edit page"** to the page's presentation component.

```
"outboundData" : {
  "outboundEndPoints" : [ {
    "name" : "transitionOutboundVars",
    "type" : "outboundVariable",
    "variableScope" : "flow",
    "values" : [ {
      "outboundPath" : "orgType",
      "value" : "<% orgTypes.value %>"
    }, {
      "outboundPath" : "organization",
      "value" : "<% instanceListOrgs.value %>"
    }, {
      "outboundPath" : "favorite",
      "value" : "<% favoriteRadioGroup.value %>"
    }, {
      "outboundPath" : "countries",
      "value" : "<% testCheckboxList.value %>"
    } ]
  } ]
},
```

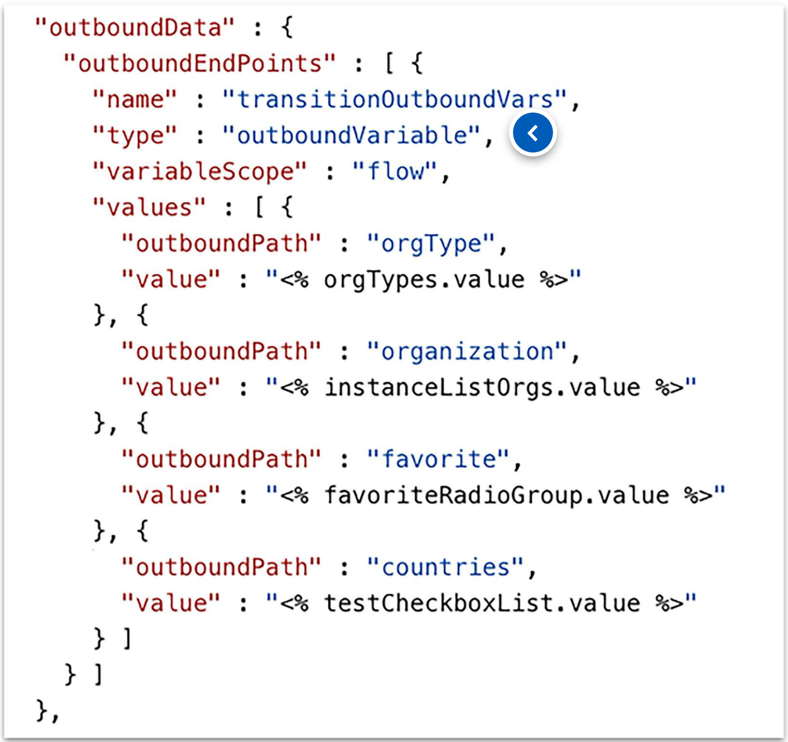
Code Block — Copy this code to define a collection of variables in an outboundVariable.

## Annotated outboundVariable Configuration

The values array of the outboundVariable defines and binds together the data from the lists. The data will be persisted within the variableScope, which is the flow.  Click the numbers in the below screenshot to learn more about the structure of an outboundVariable.



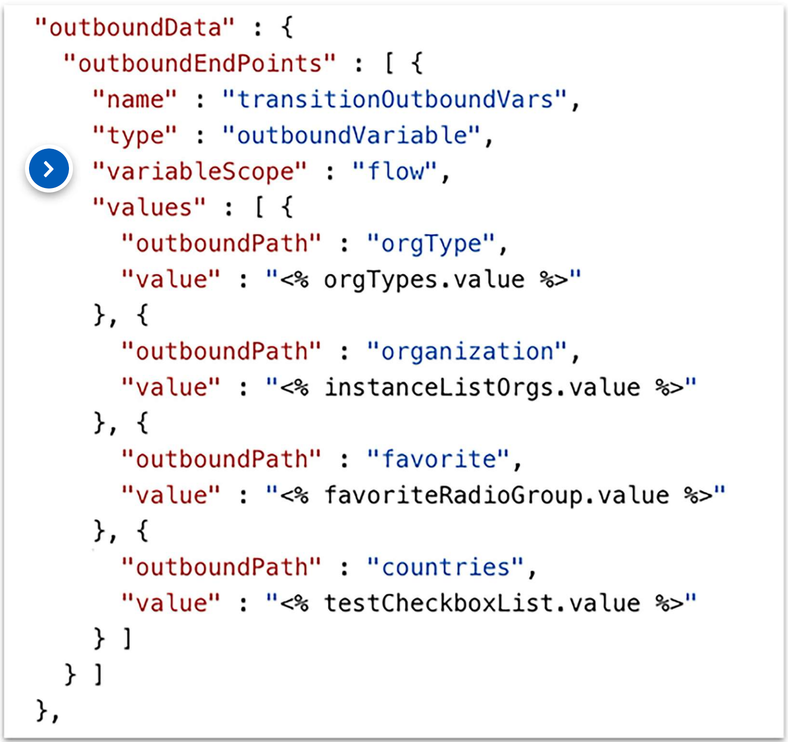
```
"outboundData" : {
  "outboundEndPoints" : [ {
    "name" : "transitionOutboundVars",
    "type" : "outboundVariable",
    "variableScope" : "flow",
    "values" : [ {
      "outboundPath" : "orgType",
      "value" : "<% orgTypes.value %>"
    }, {
      "outboundPath" : "organization",
      "value" : "<% instanceListOrgs.value %>"
    }, {
      "outboundPath" : "favorite",
      "value" : "<% favoriteRadioGroup.value %>"
    }, {
      "outboundPath" : "countries",
      "value" : "<% testCheckboxList.value %>"
    } ]
  } ]
},
```



```
"outboundData" : {
  "outboundEndPoints" : [ {
    "name" : "transitionOutboundVars",
    "type" : "outboundVariable",
    "variableScope" : "flow",
    "values" : [ {
      "outboundPath" : "orgType",
      "value" : "<% orgTypes.value %>"
    }, {
      "outboundPath" : "organization",
      "value" : "<% instanceListOrgs.value %>"
    }, {
      "outboundPath" : "favorite",
      "value" : "<% favoriteRadioGroup.value %>"
    }, {
      "outboundPath" : "countries",
      "value" : "<% testCheckboxList.value %>"
    } ]
  } ]
},
```

## outboundVariable

in the outboundEndPoints array, use outboundVariable to define one or more variables that other pages can reference. In this example, the **outboundVariable** defines a collection of variables in a **values array**. Each variable represents data that the user selected from the dropdown, instanceList, radioGroup, and checkBoxList widgets.



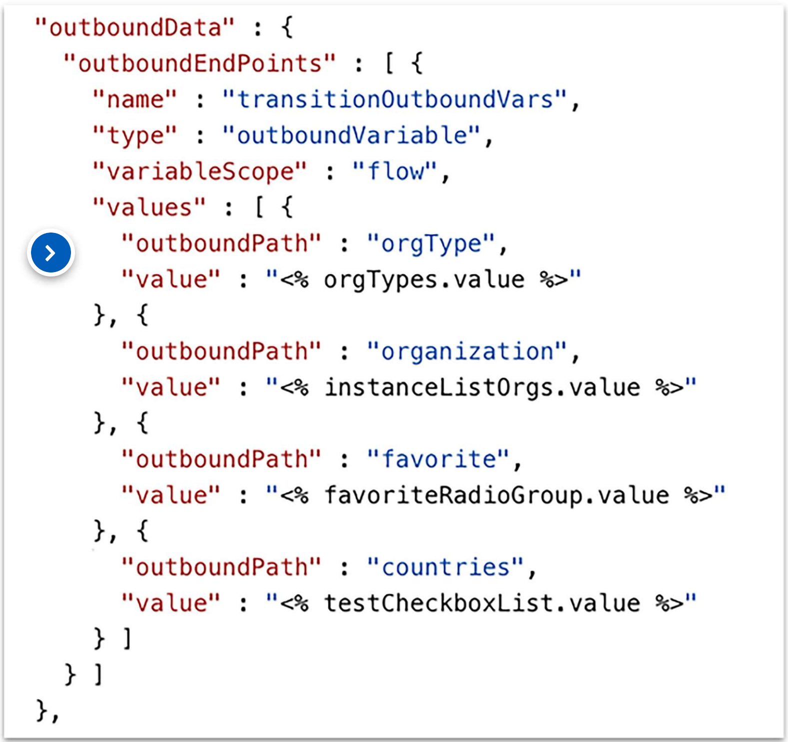
```
"outboundData" : {
  "outboundEndPoints" : [ {
    "name" : "transitionOutboundVars",
    "type" : "outboundVariable",
    "variableScope" : "flow",
    "values" : [ {
      "outboundPath" : "orgType",
      "value" : "<% orgTypes.value %>"
    }, {
      "outboundPath" : "organization",
      "value" : "<% instanceListOrgs.value %>"
    }, {
      "outboundPath" : "favorite",
      "value" : "<% favoriteRadioGroup.value %>"
    }, {
      "outboundPath" : "countries",
      "value" : "<% testCheckboxList.value %>"
    } ]
  } ]
},
```

### variableScope

Use the **variableScope** attribute to define the scope of the variables usage - a period when the variables can be accessed by other pages in the app.

Setting the variableScope to **"flow"** means that pages can access the variable during the lifecycle of a flow. In this tutorial, the scope starts on the **getUserData** page, where the user selects data from the four lists, and it ends on the **viewDataPage**, which uses the variables to display the user's data.

You can set the variableScope to **"session"**, which will make the variables available to other pages during the lifecycle of a user's session.



```
"outboundData" : {
  "outboundEndPoints" : [ {
    "name" : "transitionOutboundVars",
    "type" : "outboundVariable",
    "variableScope" : "flow",
    "values" : [ {
      "outboundPath" : "orgType",
      "value" : "<% orgTypes.value %>"
    }, {
      "outboundPath" : "organization",
      "value" : "<% instanceListOrgs.value %>"
    }, {
      "outboundPath" : "favorite",
      "value" : "<% favoriteRadioGroup.value %>"
    }, {
      "outboundPath" : "countries",
      "value" : "<% testCheckboxList.value %>"
    } ]
  } ]
},
```

### Define variables in the values array

Specify a collection of variables in the **values array** of the **outboundVariable.** Define each variable with two attributes: **outboundPath** and **value**. The outboundPath is the name of the variable. The value is the thing that you want other pages to access, such data input by users.

Alternatively, you can define flow variables on the PMD tag itself using the valueOutBinding attribute. In this case, you it's not necessary to define variables in the values array. Instead, the outboundVariable would be defined with two attributes: **name** and **variableScope**.

DEPLOY AND TEST YOUR APP

## Deploy and Test Your App

Test your app to make sure it works:

1. Click **Save and Deploy**.
2. Select a tenant and click **Deploy to Tenant**. Login to the tenant, if necessary.
3. When the **Deploy Successful box** appear, select **View in Tenant**.
4. You should see the app's root page in the browser window. The app doesn't do much yet, but that will change in the next activity!

## Download the Code

The .rtf file below contains the final code for this activity. You can use the file as a reference for each task.

ZIP

**Activity2.1.zip**

4.6 KB

**In the next activity, you'll add a radioGroup widget that will ask the user to select from two options. You'll use an instanceList to define each option with an id and a descriptor.**

## ACTIVITY 2.2: ADD A RADIO GROUP WITH TWO OPTIONS

# Activity 2.2: Add a radioGroup with Two Options

CJ Christopher Johnson

A **radioGroup widget** enables a user to quickly pick one option (and only one option) from a short list; it doesn't allow users to select more than one option. (Use the checkBoxList widget if you want to enable users to pick multiple options from the list.) In this case, you'll configure the radioGroup to display two options that will indicate if an organization is their favorite or not.

While the radioGroup list created in this activity will be short and simple, it will demonstrate some important concepts that you will apply in the other lists that you will configure in this tutorial:

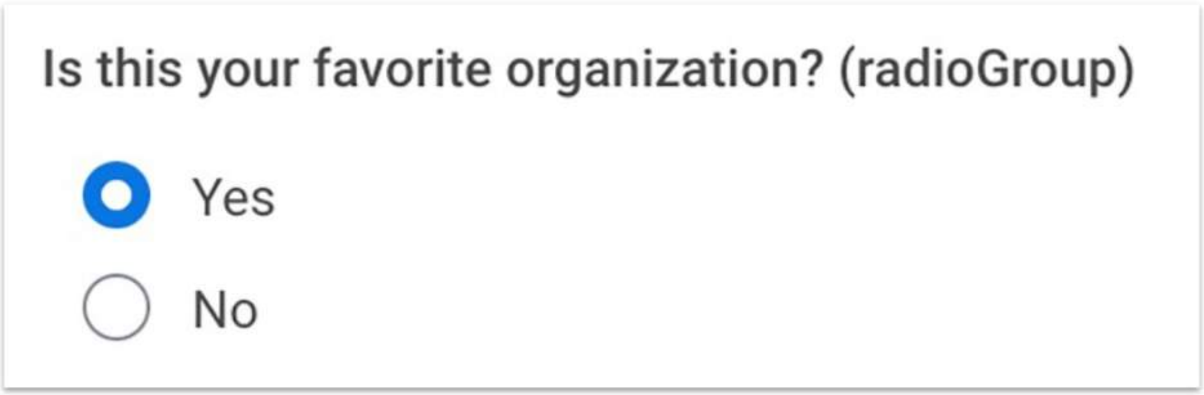
- First, an **instanceList** will define each list option with an **id** and a **descriptor**. The **radioGroup widget**, as well as the other list widgets, use the **id** and **descriptor** to populate the list with values represented by the **id** and **descriptor** parameters.
    - The **instanceList** attribute is what allows the **radioGroup** to process the **id** and **descriptor** parameters

- Second, the **valueOutBinding** attribute will be used to assign a flowVariable, which will persist the user's selected value and make it accessible to other pages in the app.

In this activity, you'll:

- Add a **radioGroup widget**.
- Use an **instanceList** attribute and the **id** and **descriptor** parameters to hard-code the list options.

- Set the **valueOutBinding** to "**flowVariables.favorite**", which will persist the user's chosen value.



ADD A RADIO GROUP LIST TO THE EDIT PAGE

1

## Add a radioGroup List to the getUserInput Page

In this task, you'll configure a radioGroup that will display to radio buttons enable users to select **Yes**, to indicate if their chosen organization is their favorite, or **No** to indicate that the organization is not their favorite.

The **radioGroup widget**, like the other list widgets you'll configure in this tutorial, needs an id and a descriptor for the values it displays on a list. When the data source is an API endpoint, radioGroup would use the **values attribute** to specify the API endpoint and get the values from the id and descriptor located at the root level of the data source.

However, in this case the radioGroup will display just two values: Yes and No. There's no need for a data source, but radioGroup needs an id and descriptor to display each value. This is why you'll use **instanceList** to hard-code the **id** and **descriptor** parameters for each option on the list.

1. In the **getUserInput page**, find the **body > children array.**

2. In the **children array**, replace the default **"hello" text widget** with a **radioGroup widget** defined by these attributes:

   - “id”: “favoriteRadioGroup”
   - “label”:  “Is this your favorite organization?”
   - “displayKey”: “descriptor"

3. After the radioGroup's **label**, add **"hideDisplayOption" : "true"**. This will *prevent* radioGroup from automatically generating a  "None of the Above" option when the widget is ***not*** required ("required": "true").

4. Add a **"valueOutBinding"** attribute that is set to **"flowVariables.favorite"**. This creates a flow variable that will persist the user's choice and make it available to other pages in the app during the lifecycle of the flow.

5. Add an **instanceList array** that will specify an **id** and a **parameter** for each option on the list. The **id** must be unique. The **descriptor** is the value that you want displayed on the list.

6. Click **Save to App Hub** to validate and save your code changes.

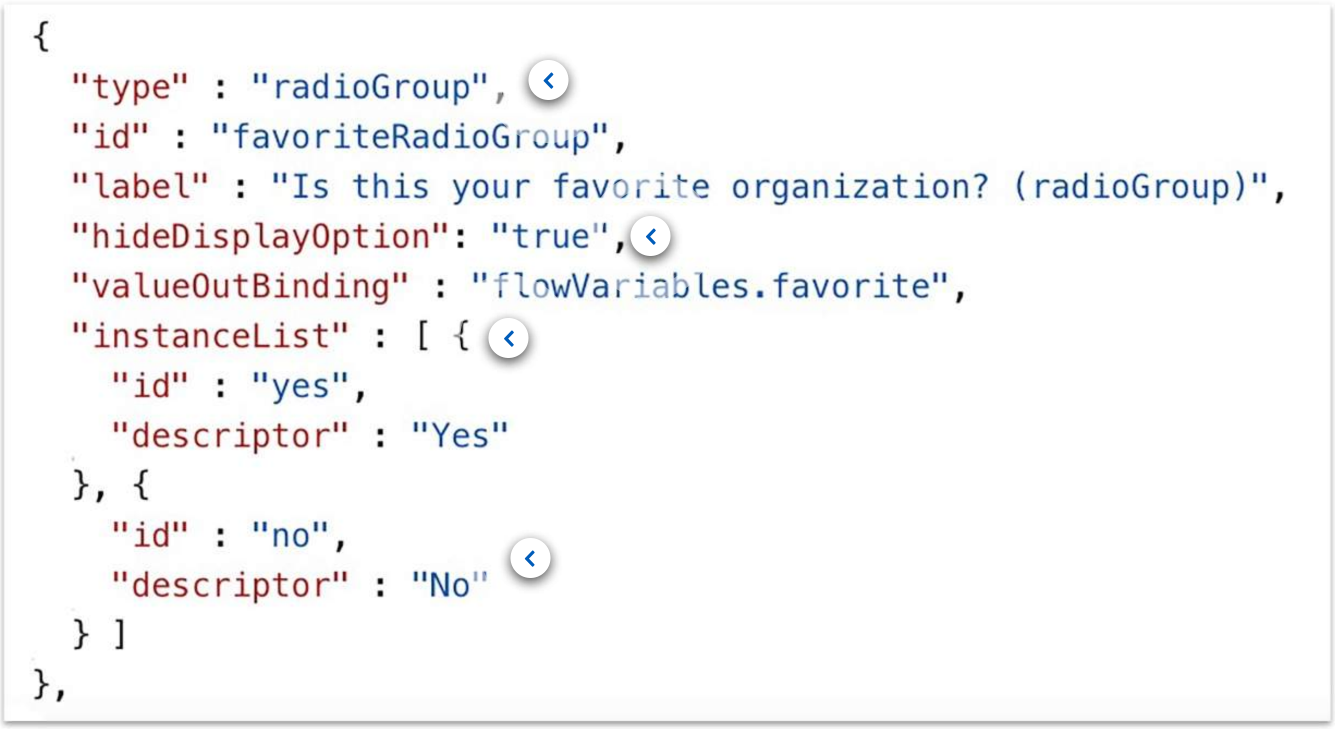
```
          {
            "type": "radioGroup",
            "id": "favoriteRadioGroup",
            "label": "Is this your favorite organization?",
            "hideDisplayOption": "true",
            "valueOutBinding": "flowVariables.favorite",
            "instanceList": [
              {
                "id": "yes",
                "descriptor": "Yes"
              },
              {
                "id": "no",
                "descriptor": "No"
```

```
                    }
                ]
            }
```

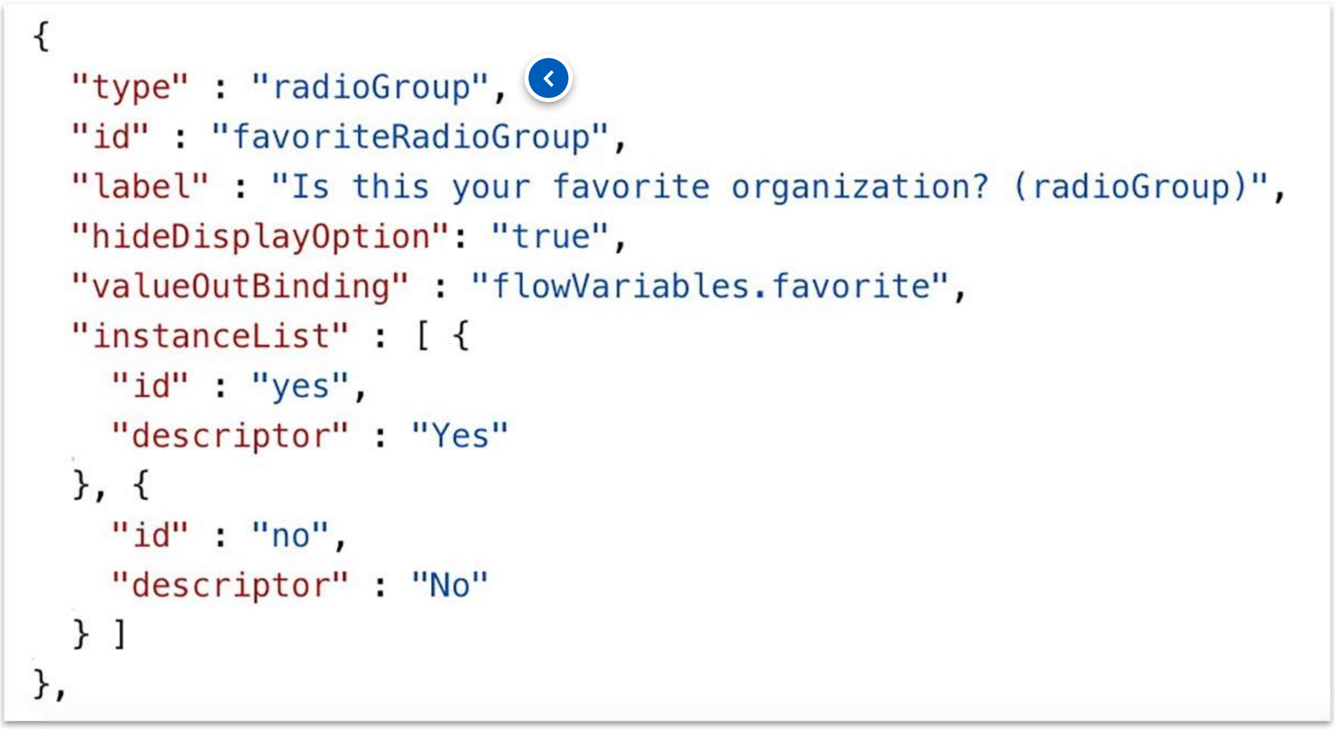
Code Block — Copy this code to add a radioGroup widget that will display two options: Yes and No.

## Annotated radioGroup Configured with instanceList

The **radioGroup** widget is good for displaying a short list of options. Use the image below to see how **instanceList** can be used to display a hard coded list of options using **id** and **descriptor** tags.



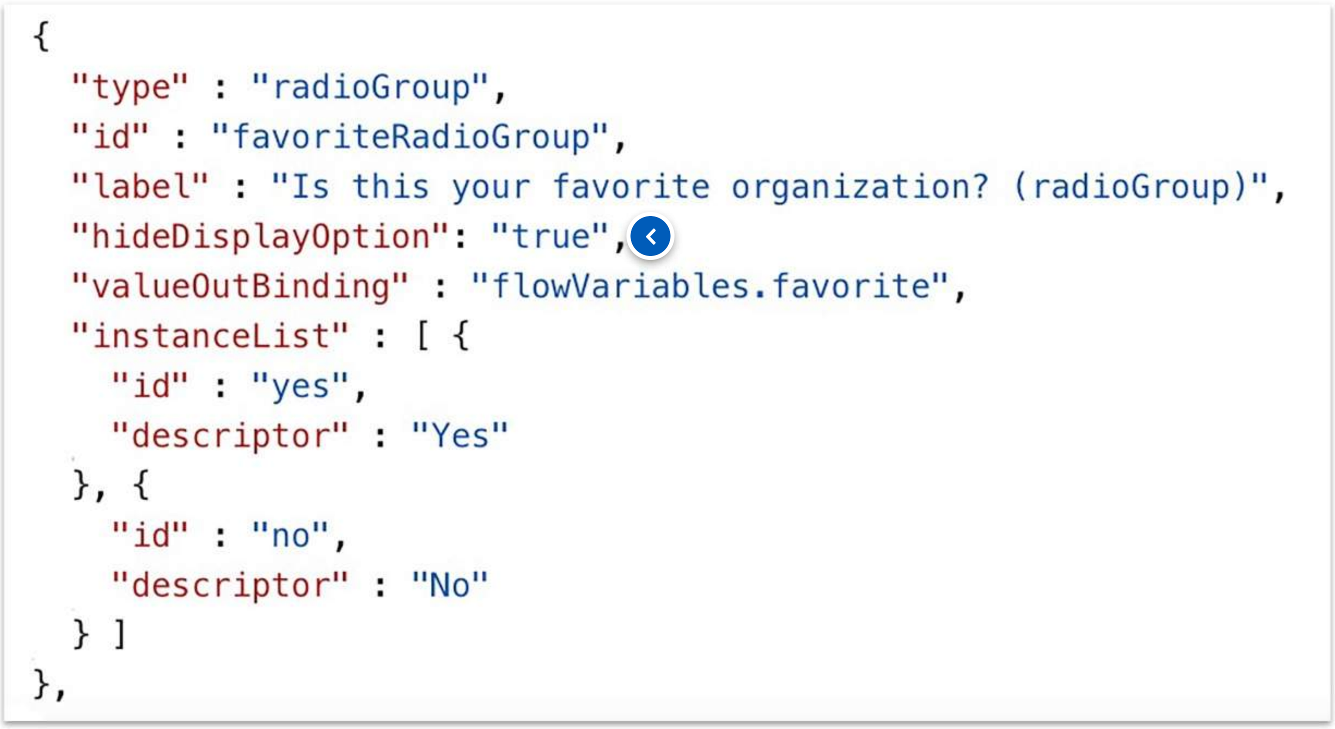
```
{
  "type" : "radioGroup",
  "id" : "favoriteRadioGroup",
  "label" : "Is this your favorite organization? (radioGroup)",
  "hideDisplayOption": "true",
  "valueOutBinding" : "flowVariables.favorite",
  "instanceList" : [ {
    "id" : "yes",
    "descriptor" : "Yes"
  }, {
    "id" : "no",
    "descriptor" : "No"
  } ]
},
```



```
{
  "type" : "radioGroup",
  "id" : "favoriteRadioGroup",
  "label" : "Is this your favorite organization? (radioGroup)",
  "hideDisplayOption": "true",
  "valueOutBinding" : "flowVariables.favorite",
  "instanceList" : [ {
    "id" : "yes",
    "descriptor" : "Yes"
  }, {
    "id" : "no",
    "descriptor" : "No"
  } ]
},
```

### radioGroup widget

The **radioGroup** widget displays a list of radio buttons, and a user can select only one option. A radioGroup is best for a short list of options.

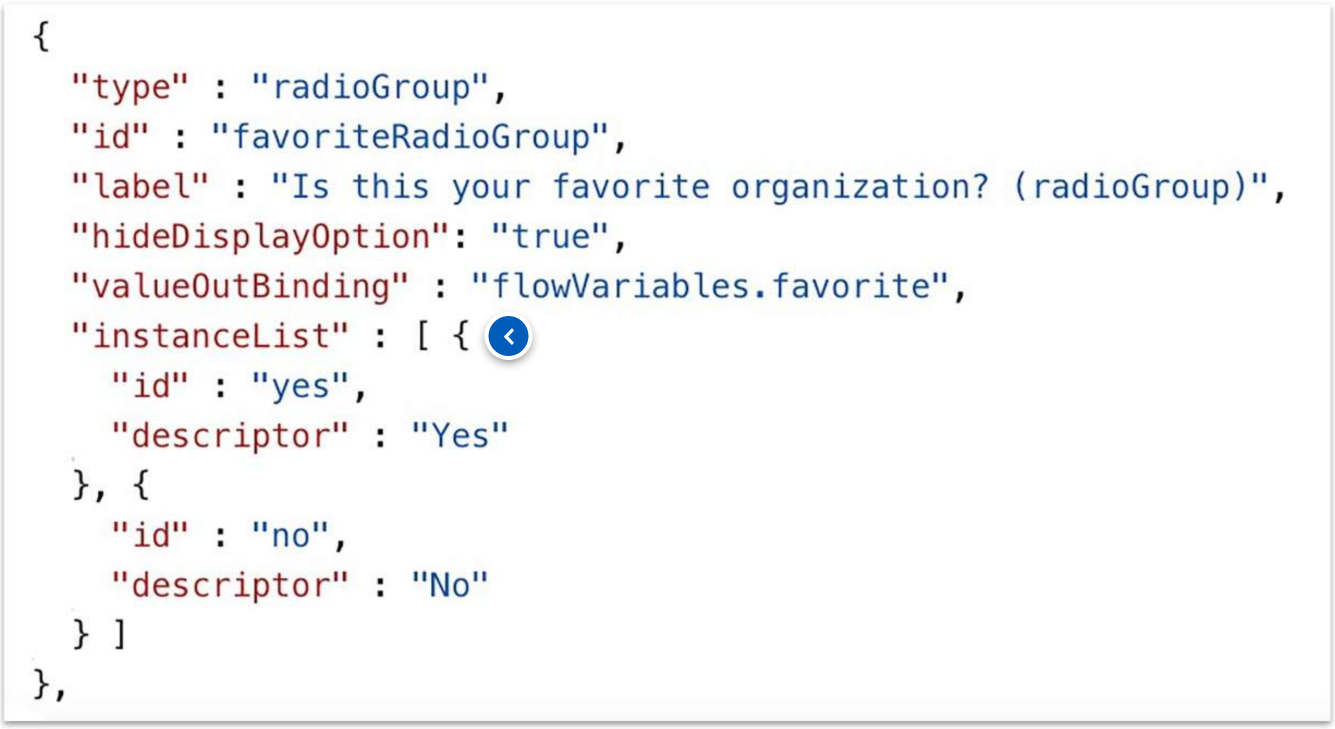


```
{
  "type" : "radioGroup",
  "id" : "favoriteRadioGroup",
  "label" : "Is this your favorite organization? (radioGroup)",
  "hideDisplayOption": "true",
  "valueOutBinding" : "flowVariables.favorite",
  "instanceList" : [ {
    "id" : "yes",
    "descriptor" : "Yes"
  }, {
    "id" : "no",
    "descriptor" : "No"
  } ]
},
```

### hideDisplayOption

By default, **radioGroup** will display a "None of the Above" option when the list is not required. To prevent radioGroup from displaying "None of the Above" on the list, add **"hideDisplayOption": "true"**.

If you want to require users to select an option, add **"required": "true"**.

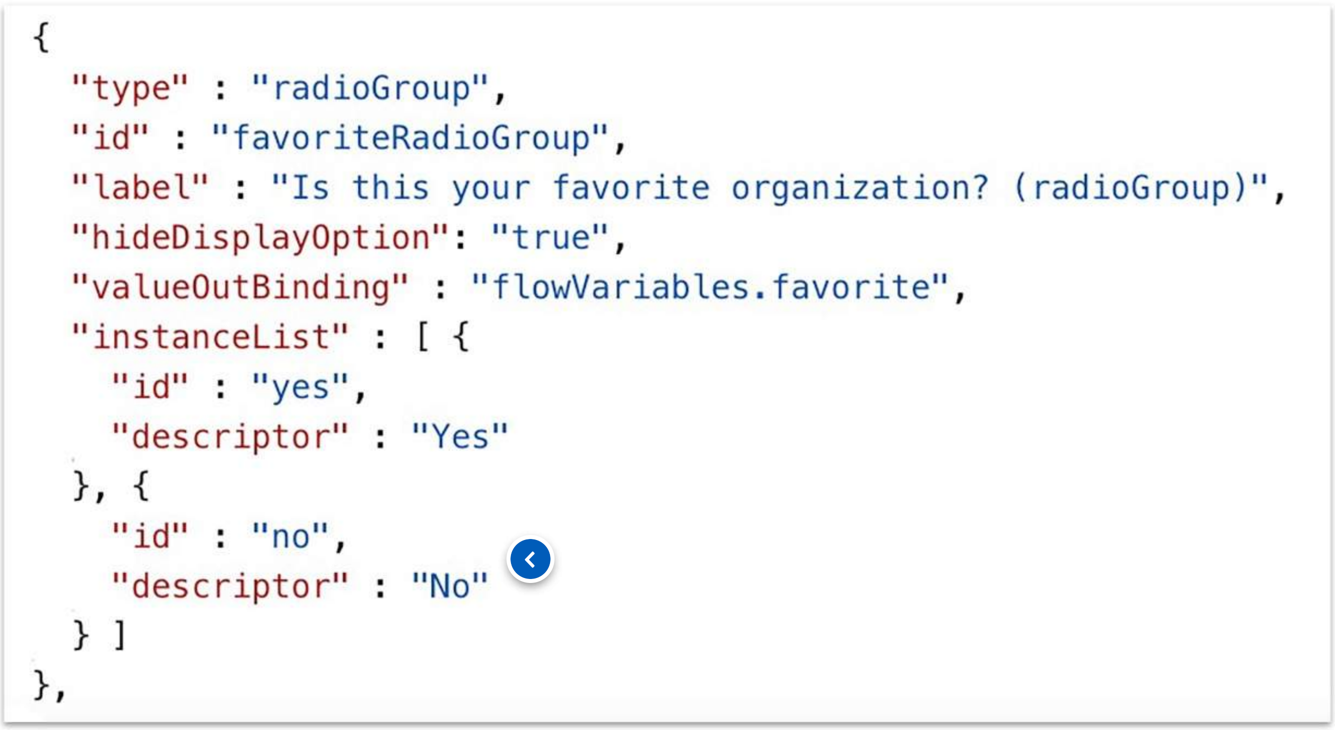


```
{
  "type" : "radioGroup",
  "id" : "favoriteRadioGroup",
  "label" : "Is this your favorite organization? (radioGroup)",
  "hideDisplayOption": "true",
  "valueOutBinding" : "flowVariables.favorite",
  "instanceList" : [ {
    "id" : "yes",
    "descriptor" : "Yes"
  }, {
    "id" : "no",
    "descriptor" : "No"
  } ]
},
```

### instanceList

When you want to display hard coded values in a **radioGroup** (or other list widgets), use **instanceList**.

Define the options you want displayed in the list with **id** and **descriptor** tags. The descriptor tag should be set to the value you want displayed on the list.



```
{
  "type" : "radioGroup",
  "id" : "favoriteRadioGroup",
  "label" : "Is this your favorite organization? (radioGroup)",
  "hideDisplayOption": "true",
  "valueOutBinding" : "flowVariables.favorite",
  "instanceList" : [ {
    "id" : "yes",
    "descriptor" : "Yes"
  }, {
    "id" : "no",
    "descriptor" : "No"
  } ]
},
```

### id and descriptor

The **radioGroup** widget, as well as other list widgets, display values represented by an **id** and **descriptor**. The **id** is a unique value that distinguishes the value from others in the list. The **descriptor** is the value that is displayed on the list.

PREVIEW YOUR CHANGES IN APP PREVIEW

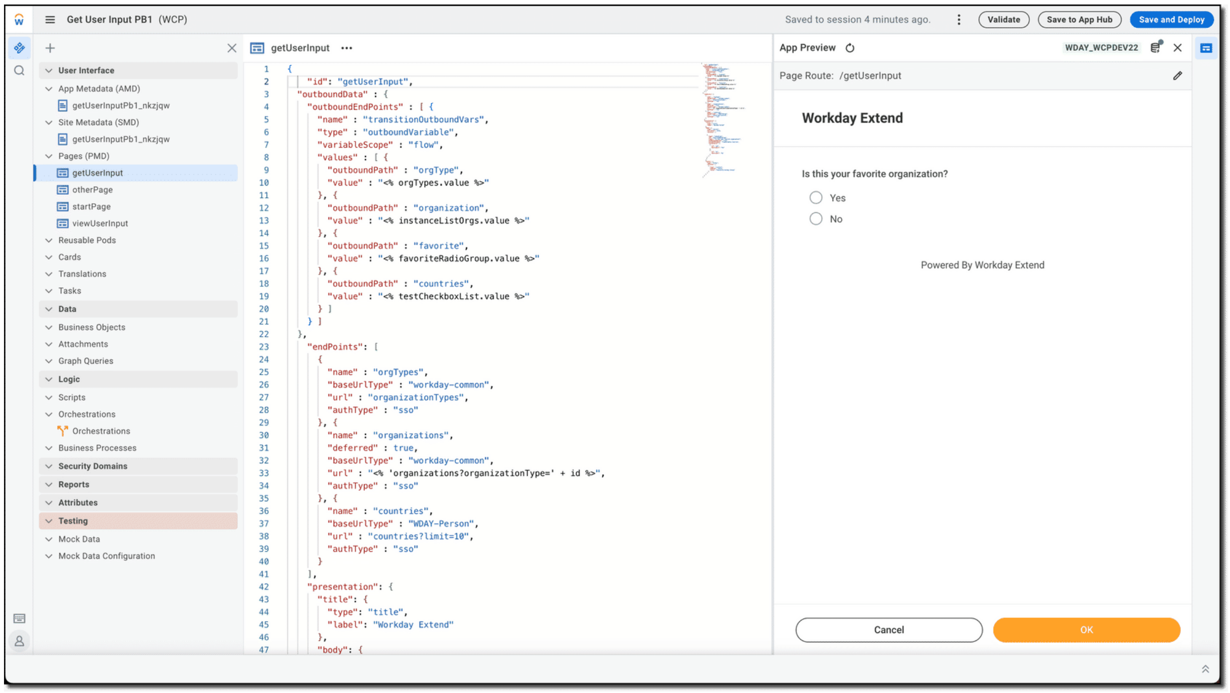
2

## Preview Your Changes in App Preview

Preview the **getUserInput page** and check that the **radioGoup** works:

1. Open the **getUserInput page** and click the **App Preview icon** in the upper-right corner of the **Content Panel**.

2. Make sure that the **radioGroup list** is displayed properly, similar to the image below.



```
{
  "id": "getUserInput",
  "outboundData" : {
    "outboundEndPoints" : [ {
      "name" : "transitionOutboundVars",
      "type" : "outboundVariable",
      "variableScope" : "flow",
      "values" : [ {
        "outboundPath" : "orgType",
        "value" : "<% orgTypes.value %>"
      }, {
        "outboundPath" : "organization",
        "value" : "<% instanceListOrgs.value %>"
      }, {
        "outboundPath" : "favorite",
        "value" : "<% favoriteRadioGroup.value %>"
      }, {
        "outboundPath" : "countries",
        "value" : "<% testCheckboxList.value %>"
      } ]
    } ]
  },
    "endPoints": [
      {
        "name" : "orgTypes",
        "baseUrlType" : "workday-common",
        "url" : "organizationTypes",
        "authType" : "sso"
      }, {
        "name" : "organizations",
        "deferred" : true,
        "baseUrlType" : "workday-common",
        "url" : "<% 'organizations?organizationType=' + id %>",
        "authType" : "sso"
      }, {
        "name" : "countries",
        "baseUrlType" : "WDAY-Person",
        "url" : "countries?limit=10",
        "authType" : "sso"
      }
    ],
    "presentation": {
      "title": {
        "type": "title",
        "label": "Workday Extend"
      },
      "body": {
```

Open App Preview to preview your code changes.

## Download the Code

The .rtf file below contains the final code for this activity. You can use the file as a reference for each task.

RTF

**Activity2.2.rtf**

4.8 KB

**In the next activity, you'll configure a checkBoxList that will use an instanceLoop to populate a list of countries.**

ACTIVITY 2.3: ADD A DROPDOWN LIST OF ORGANIZATION TYPES

# Activity 2.3: Add a Dropdown List That Displays Organization Types

CJ Christopher Johnson

In this task, you'll create **dropdown** widget to display a list of organization types on the **getUserInput** edit page. To populate the list, you'll use the **values** attribute to retrieve values from an API endpoint.

You'll remember from the previous task, list widgets need an **id** and **descriptor** to retrieve and display values. By default, the **values** attribute will retrieve ***all*** values in the **id** and **descriptor** fields located at the root-level of the endpoint data. (Later in this tutorial, you'll learn how to retrieve and display data from nested fields.)

In this activity, you'll:

- Add a **dropdown** widget to the **getUserInput** page.
- Use the **values** attribute to specify the endpoint data that will populate the dropdown list with organization types.
- Add an **onChange** event handler to trigger a PMD script that will populate a searchable list of organization names based on the organization type that the user selects.

START ACTIVITY

1

## Add a Dropdown List to the getUserInput Page

Users will start the flow by selecting an organization type from a dropdown list, which will then trigger an onChange event that will use a PMD script to populate an instanceList with the names of organizations of the same organization type chosen by the user.

1. In the **getUserInput page**, find the **body section > children array**.
2. Before the **radioGroup** list that you created in the last task, add a **dropdown** widget with these attributes:
   - "type": "dropdown"
   - "label": "Pick an Organization Type from the dropdown list."
   - "id": "organizationType"
3. After the "id", add a **values** attribute and set it to **"<% orgTypes.data %>"**. This will the retrieve ALL the values from the **id** and **descriptor** fields of the endpoint data.
4. After the values attribute, add a **"selectedValues"** tag and set it to **"2eb21d19e0a110106fecb6dd8e390931",** which is the first id the list of values in the data endpoint. This tag specifies the first id that will be displayed in the widget.
5. Add an **onChange event** that will use this Java expression to trigger a function that will populate a list of organizations: **"<% populateInstanceList(self.value) %>".** In the next task, you'll create the populateInstanceList function.
6. After the onChange event, add "**valueOutBinding**" and set it to **"flowVariables.orgType"**. The **valueOutBinding** assigns a flow variable to the value that the user will select from the list. The flow variable will persist the data and make it available to other pages during the lifecycle of the flow.

7. Click **Save to App Hub** to validate and save your code changes.

```
{
  "type" : "dropdown",
     "label" : "Organization Type",
     "id" : "organizationType",
     "values" : "<% orgTypes.data %>",
     "selectedValues" : "2eb21d19e0a110106fecb6dd8e390931",
    "onChange" : "<% populateInstanceList(self.value) %>",
     "valueOutBinding" : "flowVariables.orgType"
 },
```

Code Block — Copy this code to add a dropdown widget that will display a list of organization types.

**Remember:** When the **values** attribute is used to specify endpoint data, it will retrieve the entire dataset stored in the root-level **id** and **descriptor** fields. If you want to override the descriptor to display a value from another field, set the displayKey attribute to the field with the value that you want displayed. You can also override the id to display a value from another field using the **idKey** attribute.

## CREATE A PMD SCRIPT THAT WILL POPULATE A LIST OF ORGANIZATION NAMES

## Create a PMD Script Will Populate a List of Organization Names

Next, we want to populate a second list that will display names of organizations that are the same organization type that the user selected from the first dropdown list. To do this, you'll add an onChange attribute to the dropdown list created in the previous task. Then you'll write a PMD script that will be triggered by an onChange event.

1. In the **getUserInput page**, add an **onChange event** that will trigger a PMD script named **populateInstanceList**.
2. In the getUserData page, find the **page id,** which is located at the top of the page.
3. After the **"id" : "getUserInput"** tag and before the **"outboundData"** tag, add a **PMD script** that will populate an **instanceList** of organizations.
4. **Validate** and **save** your code.

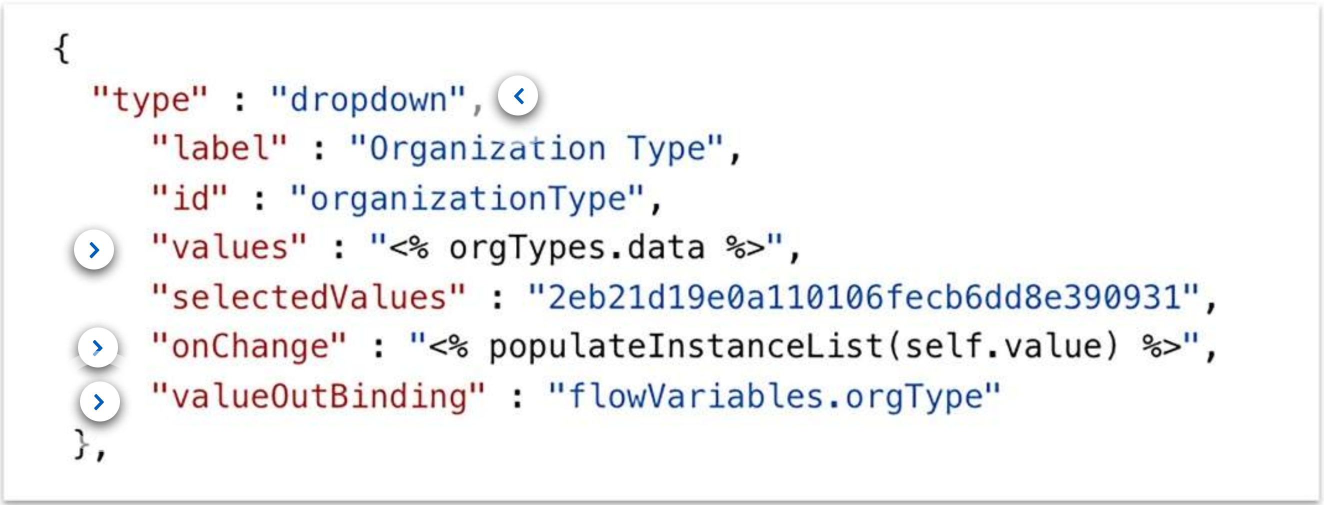
```
"script": "<%
           var populateInstanceList = function(orgId) {

               if( !(empty orgId)) {
                   console.info('organization type ID is ' + orgId);

                       var listData = organizations.invoke({'id': orgId[(
                       console.info('endpoint response is ' + listData.da
                       instanceListOrgs.setValues(listData.data.map(x =>
               } else {
                   instanceListOrgs.setValues([]);
               }
           };
           %>",
```

Code Block — Copy this code to add a PMD script that will populate the instanceList.

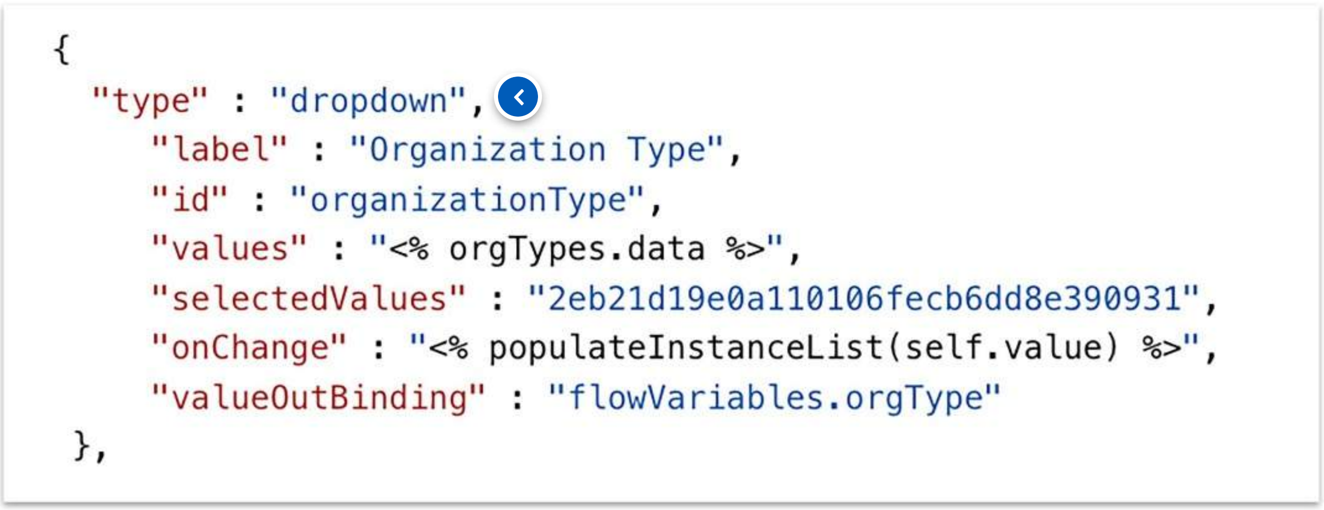
**It's recommended that you use a PMD script to reshape data from root-level and nested-fields. With a PMD script, you can call PMD functions that will efficiently manipulate data when it's displayed on a page or submitted to outbound endpoints.**

## Annotated Dropdown List Configuration

Dropdown lists can allow you the ability to choose a selection from a REST API call. Click the numbers in the below screenshot to learn more about the structure of an dropdown PMD widget.



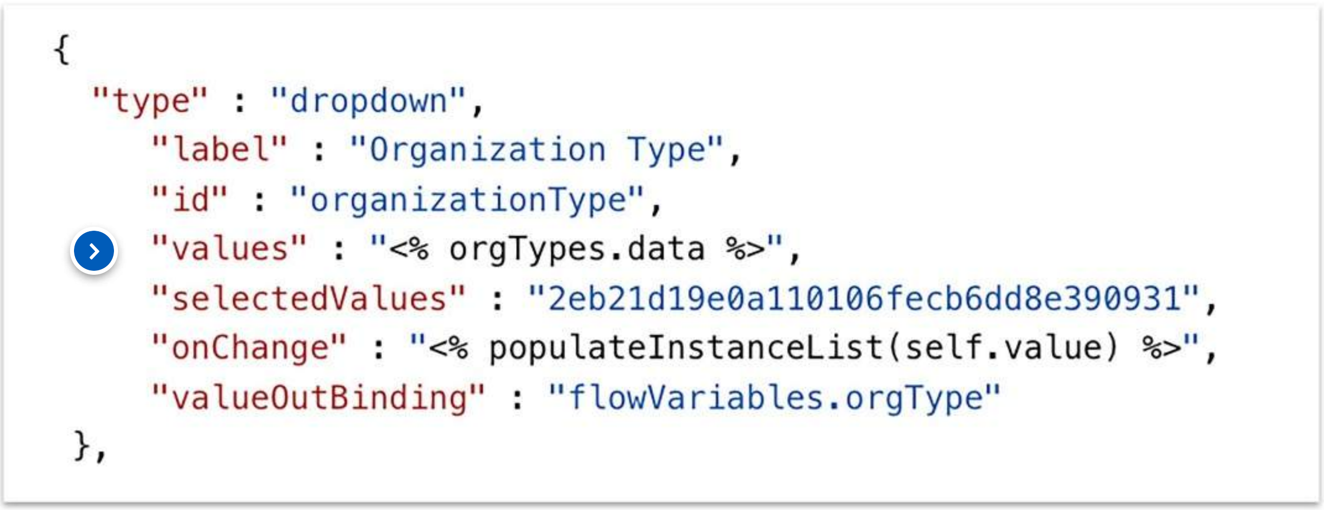
```
{
  "type" : "dropdown",
     "label" : "Organization Type",
     "id" : "organizationType",
     "values" : "<% orgTypes.data %>",
     "selectedValues" : "2eb21d19e0a110106fecb6dd8e390931",
     "onChange" : "<% populateInstanceList(self.value) %>",
     "valueOutBinding" : "flowVariables.orgType"
 },
```



```
{
  "type" : "dropdown",
     "label" : "Organization Type",
     "id" : "organizationType",
     "values" : "<% orgTypes.data %>",
     "selectedValues" : "2eb21d19e0a110106fecb6dd8e390931",
     "onChange" : "<% populateInstanceList(self.value) %>",
     "valueOutBinding" : "flowVariables.orgType"
 },
```

### dropdown widget

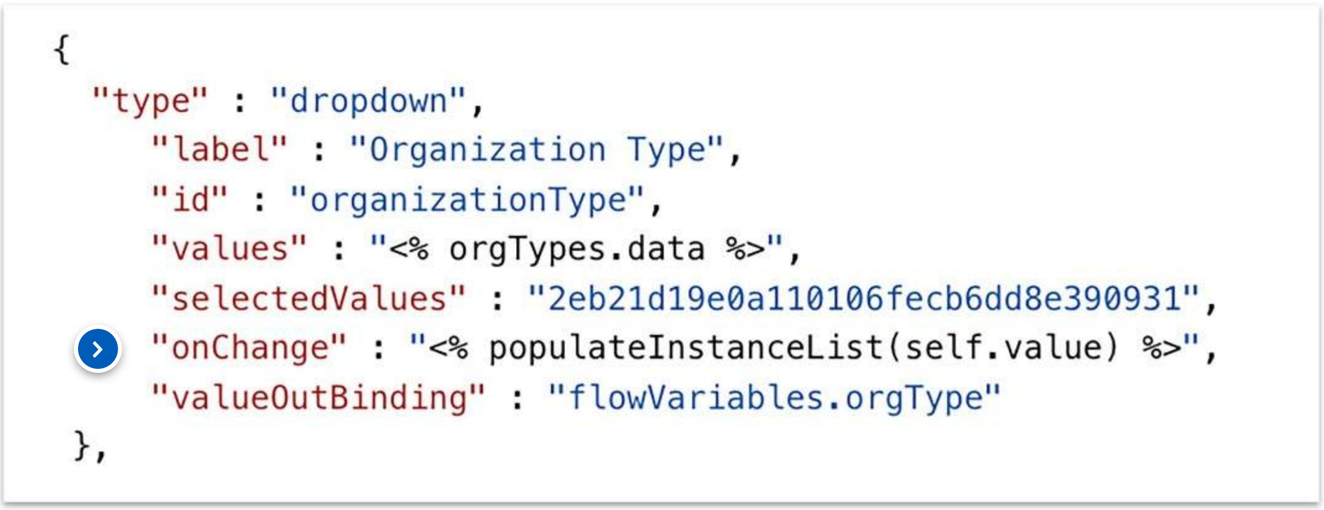
The **dropdown** widget creates a prompt that enables users to select an option from a list of options.
Use dropdown on edit pages when you want the user to select a single option from a dropdown list. You can also use the dropdown tag with the dropdownEditButton.



```
{
  "type" : "dropdown",
     "label" : "Organization Type",
     "id" : "organizationType",
     "values" : "<% orgTypes.data %>",
     "selectedValues" : "2eb21d19e0a110106fecb6dd8e390931",
     "onChange" : "<% populateInstanceList(self.value) %>",
     "valueOutBinding" : "flowVariables.orgType"
 },
```

## Values

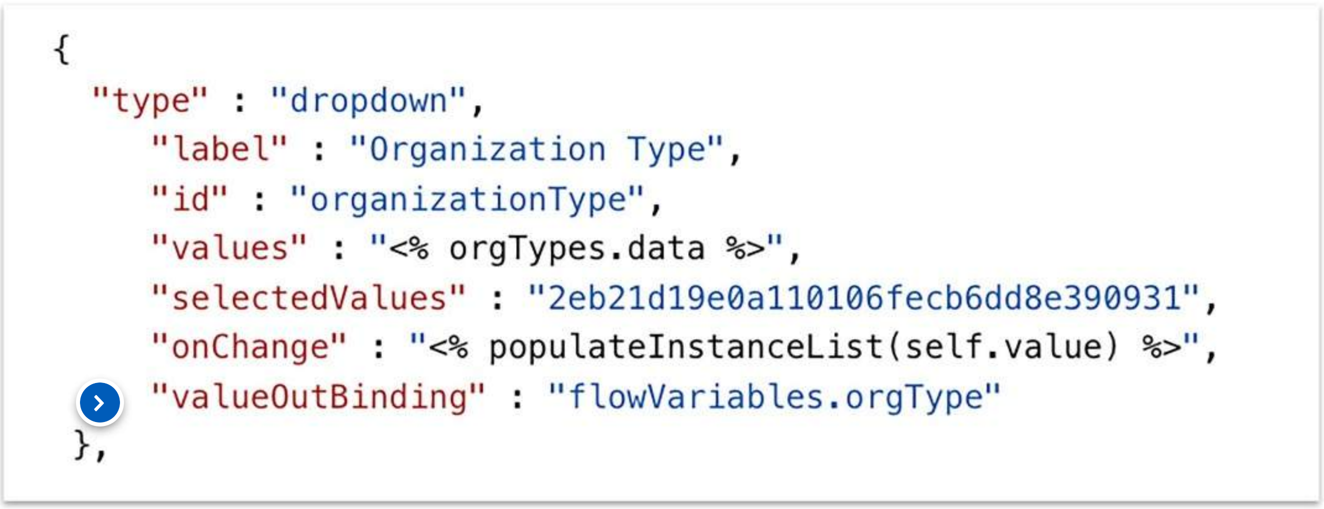
Specify **values** with endpoint data, which must be a list containing the id and descriptor fields. Using the values attribute is the most common way to populate a list.



```
{
  "type" : "dropdown",
     "label" : "Organization Type",
     "id" : "organizationType",
     "values" : "<% orgTypes.data %>",
     "selectedValues" : "2eb21d19e0a110106fecb6dd8e390931",
     "onChange" : "<% populateInstanceList(self.value) %>",
     "valueOutBinding" : "flowVariables.orgType"
 },
```

### onChange will trigger the PMD script

When the user selects an option from the dropdown list, the populateInstanceList function will run and populate the next list with organizations that are the org type that the user picked from the list.



```
{
  "type" : "dropdown",
     "label" : "Organization Type",
     "id" : "organizationType",
     "values" : "<% orgTypes.data %>",
     "selectedValues" : "2eb21d19e0a110106fecb6dd8e390931",
     "onChange" : "<% populateInstanceList(self.value) %>",
     "valueOutBinding" : "flowVariables.orgType"
 },
```

### valueOutBinding

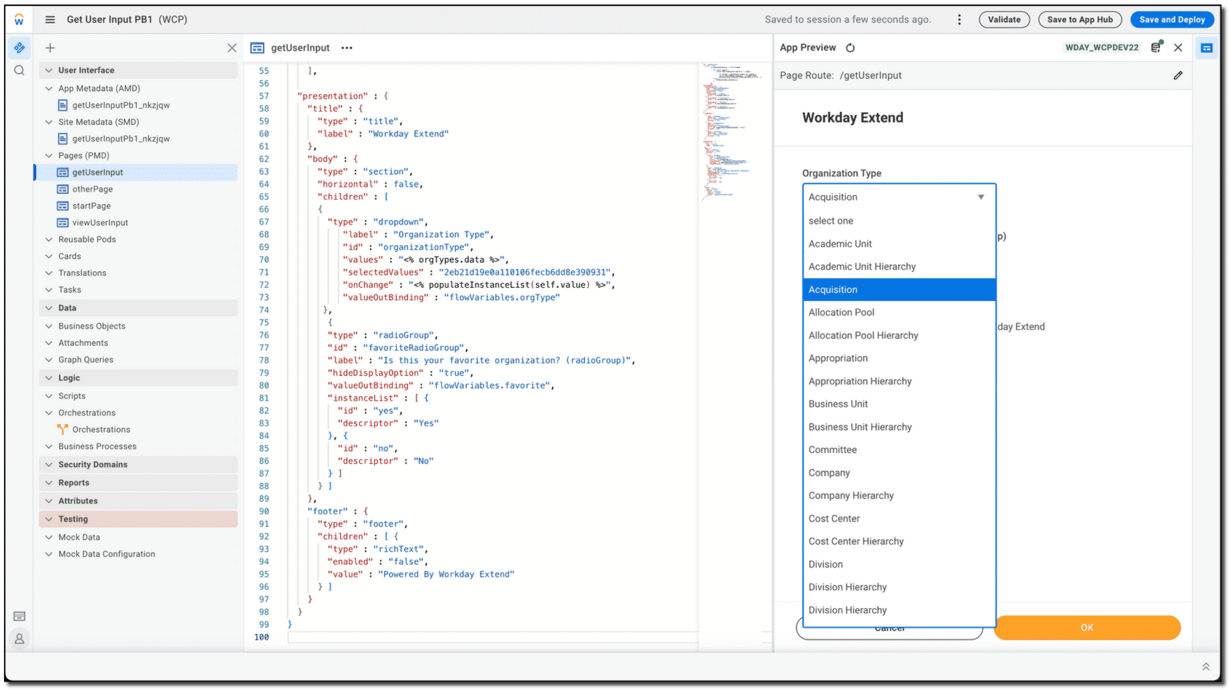
Use the **valueOutBinding** attribute to define the variable within the context of the page widget.

PREVIEW YOUR CHANGES IN APP PREVIEW

3

## Preview Your Changes in App Preview

1. Click the **App Preview icon** in the upper-right corner of the **Content Panel**.
2. If the **"Data Source Required"** message appears in the App Preview panel, click the **Manage Data Source icon** (right-corner of the window), and login to your development tenant.
3. Click the dropdown list so that you can check that it correctly displays a list of organization types, as shown in the image below.



```
    ],

  "presentation" : {
    "title" : {
      "type" : "title",
      "label" : "Workday Extend"
    },
    "body" : {
      "type" : "section",
      "horizontal" : false,
      "children" : [
      {
        "type" : "dropdown",
           "label" : "Organization Type",
           "id" : "organizationType",
           "values" : "<% orgTypes.data %>",
           "selectedValues" : "2eb21d19e0a110106fecb6dd8e390931",
           "onChange" : "<% populateInstanceList(self.value) %>",
           "valueOutBinding" : "flowVariables.orgType"
      },
       {
        "type" : "radioGroup",
        "id" : "favoriteRadioGroup",
        "label" : "Is this your favorite organization? (radioGroup)",
        "hideDisplayOption" : "true",
        "valueOutBinding" : "flowVariables.favorite",
        "instanceList" : [ {
          "id" : "yes",
          "descriptor" : "Yes"
        }, {
          "id" : "no",
          "descriptor" : "No"
        } ]
      } ]
    },
    "footer" : {
      "type" : "footer",
      "children" : [ {
        "type" : "richText",
        "enabled" : "false",
        "value" : "Powered By Workday Extend"
      } ]
    }
  }
}
```

Open App Preview and test the dropdown list to make sure that it is populated with organization types.

## Download the Code

The .rtf file below contains the final code for this activity. You can use the file as a reference for each task.

RTF

**Activity2.3.rtf**

6.3 KB

**In the next activity, you'll use the instanceList widget that will enable users to search, and select from, a list of organizations.**

## ACTIVITY 2.4: CONFIGURE A LIST THAT WILL DISPLAY ORGANIZATION NAMES

*Lesson 6 of 6*

# Knowledge Check

CJ Christopher Johnson

This quiz will check your understanding of the concepts discussed in this tutorial.

*Question*

**01/10**

To configure a PMD as an edit page that will accept user inputs, which of the following must be done? (Select all that apply.)

- ☐ Add outboundEndPoints.
- ☐ Add EndPoints.
- ☐ Add “pageType” : “edit page” to the page’s presentation component.
- ☐ Add dataProviders to the page.

***Question***

**02/10**

From the list below, choose the two attributes that you must put in the dropDownButton's values array in order to navigate to another page in your app?

- ☐ commandHttpMethod
- ☐ label
- ☐ enabled
- ☐ taskReference

*Question*

**03/10**

What taskReference attribute would you use to send key-value parameters to the next page?

- ○ type
- ○ taskId
- ○ parameterBindings
- ○ values

*Question*

**04/10**

Which of the following tags would you use to store a value that a user selects from a list, such as a dropdown or checkBoxList widget?

- ○ instanceLoop
- ○ outboundEndPoint
- ○ valueOutBinding
- ○ values

*Question*

**05/10**

Which of the following are reasons to use instanceList? (Select all that apply.)

---

- ☐ Create a multilevel list that enables users to drill down a folder-like structure and select a value from the last level (leaf node).
- ☐ Reshape data retrieved from root-level and nested fields.
- ☐ Display a list on an edit page from which a user can search and select values.
- ☐ Instantly populate another list with values.
- ☐ Associate a related actions menu with each item on a list that is displayed on a view page.
- ☐ Loop through a data source and display the returned instances in a list.

*Question*

**06/10**

What are the two attributes that you would use to define a collection of variables in the values array of an outboundVariable?

---

○ name and label

○ variableScope and flow

○ flowVariable and values

○ outboundPath and value

*Question*

**07/10**

What tag would you use to iterate through the data and return a subset of values located in root-level and nested fields?

- ○ instanceList
- ○ instanceListLoopTag
- ○ flow
- ○ loopFlow

*Question*

**08/10**

The most common way to populate a list is to use the list widget’s values attribute to specify endpoint data. From the list below, identify the fields located at the root-level of the endpoint data that are used to populate the list. (Select all that apply.)

---

- ☐ descriptor
- ☐ name
- ☐ label
- ☐ id

*Question*

**09/10**

If you want override the root-level descriptor field of endpoint data so that you can display values from another field, what attribute would you use?

---

○ flowVariable

○ displayKey

○ name

○ enabled

*Question*

**10/10**

As a best practice, what is the best method to reshape data retrieved from endpoint data?

- ○ instanceListLoopTag
- ○ PMD script with the values attribute
- ○ instanceLoop
- ○ on