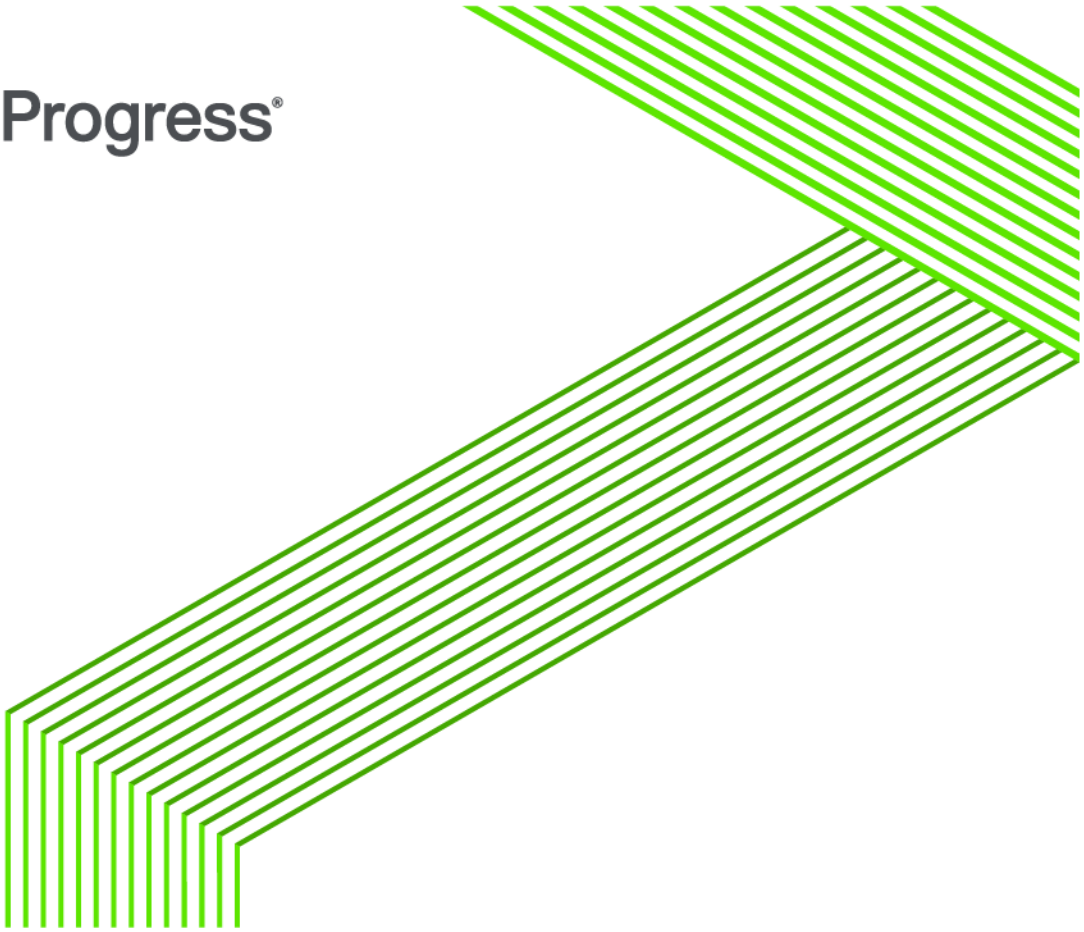
Progress®



Kendo UI® Builder by Progress®:
Sample Workflow

Kendo UI® Builder by Progress

Copyright

© 2017 Telerik AD. All rights reserved.

July 2017

Last updated with new content: Version 2.0

Updated: 2017/07/15

Table of Contents

1

Overview

Using Progress® Developer Studio for OpenEdge® (Developer Studio) and Kendo UI® Builder by Progress® (Kendo UI Builder), you can separate the user interface (UI) from the business logic that can together be deployed and run on a Progress Application Server (PAS) instance.

You can use Kendo UI Builder to design and develop a web UI for an OpenEdge application using in-built templates. These templates use a data source definition, Data Service Catalog, that is implemented through a Data Object Service. The OpenEdge application architecture provides a service interface for the CRUD operations in the Data Service Catalog. These templates then generate the UI metadata for CRUD support without requiring detailed input from you.

The layout and properties of web pages that is saved as metadata is used to generate the web app. When you use metadata and templates, the amount of JavaScript code that you need to write is minimized.

You can upgrade the UI for future versions of the web app, using the customizable templates and metadata from which the web app is generated.

Note: For more information on the architecture and components of OpenEdge and Kendo UI Builder, see the *Kendo UI Builder by Progress: Modernizing OpenEdge Applications*.

For details, see the following topics:

- Prerequisites

Prerequisites

Here are the prerequisites for creating a sample web app:

- OpenEdge 11.6.3 (64-bit) or later and Kendo UI Builder 2.0 must be installed on your machine.

  Note: If you do not have OpenEdge and Kendo UI Builder on the same machine, you have to copy the web app folder into the associated Web UI project location. For more information on working with a Web UI project with OpenEdge Service Pack 11.6.3, see *OpenEdge Service Pack 11.6.3: New Information*, and with later OpenEdge releases, see the Progress Developer Studio for OpenEdge online help.

- A copy of sports2000 database in `C:\OpenEdge\WRK\db` with name sports2000 must be created.
- AdminServer must be started. To start the AdminServer:
  - Enter `PROADSV -start` in the Proenv command line, or
  - Select **Control Panel** > **Administrative Tools** > **Services** and then set the AdminService for OpenEdge to **Started**.
- In Developer Studio, an OpenEdge Explorer connection must be set up and the Servers view must show the Progress Application Server for OpenEdge (PAS for OpenEdge) instance, oepas1. For more information, see the Developer Studio online help.

2

Creating a sample web app using Kendo UI Builder by Progress

You can create your OpenEdge web app using Developer Studio and Kendo UI Builder with the following steps:

1. Create an ABL Web App project with an ABL Service of Data Object type with either the REST RPC or the WebHandler service provider. The service provider that you select, specifies the request and response transport and message protocol that accesses the annotated Business Entity used by the Data Object Service.

   Note:

   An ABL Service of Data Object type is an OpenEdge Data Object Service that provides web access to your ABL business logic through one or more OpenEdge Data Objects that use the ABL Business Entities. An OpenEdge web app can then access this standard interface that OpenEdge application architecture provides, using a JavaScript Data Object (JSDO). A JSDO hides the underlying details of the network request and response protocol of the web app. A Data Object Service then manages all web access between an instance of the JSDO in the web app and the Data Object running on the OpenEdge application server.

   For more information on service providers, see the *Progress Data Objects Guide and Reference* guide. For more information on Data Objects Services, see the *OpenEdge Development: Web Services* guide. For more information on Business Entities, ABL Service of Data Object type (Data Object service), and OpenEdge projects, see the Progress Developer Studio for OpenEdge online help.

2. Create a Business Entity with semantic type field-level annotations. These annotations help you customize the options for how the Kendo UI Designer uses the table field values provided by the Data Object resource that you implement with the Business Entity.

Note: ABL Business Entities are annotated ABL class- or procedure-based objects that provide a standard web interface for your data and business logic. For an overview of OpenEdge Data Object Services and how to implement ABL Business Entities as Data Objects, see the *OpenEdge Development: Web Services* guide.

3. Edit the ABL Service of Data Object type to use the annotated Business Entity. The associated Data Service Catalog is generated. This Catalog is a JavaScript Object Notation (JSON) file that contains metadata describing the schema and operations supported by each Data Object managed by the Data Object Service.

   Note: The JSDOs that the Kendo UI Designer and its generated web apps create to access Data Object resources rely on the Catalog for each Data Object Service.

4. Deploy the Data Object Service on PAS for OpenEdge.
5. Create a web app using the Kendo UI Designer

   The web app you create is based on the selected Data Object Service metadata and UI templates for the supported Kendo UI components. You create each module of the web app, and each view in the module, using the built-in UI templates, such as for a grid or form, with a predefined Kendo UI Builder configuration. You then add data providers and data sources that bind data to each view by associating a Data Object resource table as a data source. You can then define app function and presentation by setting properties of the app, each module, and its views, then preview the result with real data from the data sources that are bound to the views.

   Note: For more information on using Kendo UI Designer, see the Kendo UI Builder by Progress: Using Kendo UI Designer guide. For more information on the architecture and components of OpenEdge and Kendo UI Builder, see the *Kendo UI Builder by Progress: Modernizing OpenEdge Applications* guide.

6. Preview and test the web app by invoking the Kendo UI Generator. It takes the saved JSON UI metadata and referenced UI templates as input, and generates a deployable web app and the HTML5/CSS and JavaScript files. The HTML5/CSS and JavaScript files are then saved to the location you specified, which can be a Web UI project in Developer Studio. You can then invoke a preview of the app from the Designer in your default browser, which is run and managed by a webpack-dev-server.
7. Create a Web UI project in the same location that has your web app and deploy the contents of this project on PAS for OpenEdge. You can use this deployment as a development build for testing on a development instance of PAS for OpenEdge or as a release build for delivery on a production instance of PAS for OpenEdge. In addition, you can export the Web UI project as a Web UI application, which creates a WAR file for your web app that you can deploy to a web server.

   Note: For more information on PAS for OpenEdge, see *Progress Application Server for OpenEdge: Introducing PAS for OpenEdge* and *Progress Application Server for OpenEdge: Application Migration and Development* guides. For more information on working with a Web UI project with OpenEdge Service Pack 11.6.3, see *OpenEdge Service Pack 11.6.3: New Information*, and with later OpenEdge releases, see the Progress Developer Studio for OpenEdge online help.

In the ensuing topics, we create a sample web app, named OrderEntryWebAp, for customer order entries. First, we create an ABL Web App project named OrderEntry, which contains the Data Object Service, OrderEntryService. The service uses an annotated Business Entity. The JSON file for the Data Service Catalog is generated. The data provider is then used by Kendo UI Builder when we create the web app, OrderEntryWebApp. We will also create modules containing views that we associate with (bind to) the data provider. We will finally deploy the web app on PAS for OpenEdge using a Web UI project.

Note: The steps for creating and deploying a web app oscillate between using Developer Studio and Kendo UI Builder.

For details, see the following topics:

- Creating a new project
- Creating and annotating a Business Entity
- Editing the Data Object Service
- Publishing the Data Object Service
- Creating the web app in Kendo UI Designer
- Creating a Web UI project
- Launching the web app

Creating a new project

Now we create a new ABL Web App project, OrderEntry, that contains the Data Object Service, OrderEntryService.

To create the project:

1. Launch Developer Studio.
2. Select **File** > **New** > **OpenEdge Project**. The **Create an OpenEdge Project** page opens.
3. In the **Project name** field, enter OrderEntry.
4. Clear the **Use default** check box and enter `C:\OpenEdge\WRK\OrderEntry` as the path.
5. Select **ABL Web App** from the **Project type configuration** drop-down list.
6. Click **Next**. The **Provide ABL Web App deploy details** page opens.
7. Retain the default settings; the **Deploy as WebApp** option button must be selected, OrderEntry must be the **Business Logic Module name** field and AppServer must be the **ABL Source folder** field.
8. Clear any selection in the Server name.

   Note: We will publish the ABL Web App project and the Data Object Service manually after we create them.

9. Click **Next**. The **Create an ABL Service** page opens.
10. Create an ABL Service of Data Object type.
11. Click **Next**. The **Select AVM and layout options** page appears.

12. Clear the **Use TTY for runtime** check box and select the **Use project-specific AVM** option button.
13. Click **Next**. The **Define PROPATH** page appears.
14. Click **Next**. The **Select Database Connections** page appears.
15. Click the **Configure database connections** link on the top right. The **Database Connections** dialog box appears.
16. Configure a database connection for this project.
17. Select the **Database Connections** tab and select the **sports2000** check box.

    The Project Explorer displays the new project, OrderEntry, and the ABL Service, OrderEntryService, under the Defined Services node. The AppServer directory is also created where we will add our ABL code. Notice that the ABL Console view displays messages that the AVM and the database server are started.

Note: The ABL Web App project can be deployed on PAS for OpenEdge only so we wll deploy the sample web app on PAS for OpenEdge. You can create a Data Object Service with Data Object project and then deploy on classic servers (OE Web Server).

Creating an ABL Service of Data Object type

We now create an ABL Service of Data Object type using the **Create an ABL Service** page.

1. Select Data Object as the **Service type**.
2. Select REST RPC as the **Service provider**.

Note: For the sample web app we use a REST provider for Web transport and messaging protocol for client apps to access Data Object resources (Business Entities) on the OpenEdge application server. For more information on service providers, see the *Progress Data Objects and Reference* guide.

---

3. Retain OrderEntryService for the **Service name** field and **/OrderEntryService** for the **Service relative URI** field.

   The Sample URI appears as http://*`host[:port]`*/OrderEntry/rest/OrderEntryService/*`Resource URI`* at the bottom of the page.
4. Click **Next**. The ABL Service of Data Object type is created.

Configuring a database connection

We now set up a database connection for the ABL Web App project.

1. Click **New…** to create a database connection. The **Add OpenEdge Database Connection** page opens.
2. In the **Connection name** field, enter sports2000.
3. In the **Physical name** field, enter `C:\OpenEdge\WRK\db\sports2000.db`.
4. In the **Host name** field, enter localhost.
5. In the **Service/Port** field, enter 6015.
6. Click **Next**. The **Define a SQL connection** page opens.
7. Click **Next**. The **Add SQL Connection Profile** page opens.

8. Click **Next**. The **Define Database Server Configuration** page opens.
9. Select the **Auto-start database server** check box.
10. In the **Service/Port** field, enter 6015.
11. Click **OK**. The database connection for the ABL Web App project is created.

Creating and annotating a Business Entity

We now create a Business Entity in the AppServer layer to access the customer data that we want to make available to our OpenEdge web app.

1. Right-click the OrderEntry project in the Project Explorer and select **New** > **New Business Entity**.The **Create a Business entity class** page of **New Business Entity** wizard appears.
2. Retain \OrderEntry\AppServer for the **Package root** field.
3. Enter CustomerBE in the **Business Entity name** field.
4. Click **Next**. The **Select a schema file** page opens
5. Select the **CRUD and Submit** option button in **Operations**.
6. Select the **Select database table** option button. The **Connection** field displays sports2000.
7. Click the **Table** drop-down list and select Customer.
8. Click **Finish**. The `CustomerBE.cls` file appears under the AppServer folder in the OrderEntry project in Project Explorer.
9. Double-click the `CustomerBE.cls` file to edit. The class is already annotated as a Data Object Service Interface by the **New Business Entity** wizard using the annotation:

```
@progress.service.resource FILE(name="CustomerBE", URI="/CustomerBE",
schemaName="dsCustomer", schemaFile="OrderEntry/AppServer/customerbe.i")
```

10. Scroll down to the `ReadCustomerBE` method definition. The `Read` method uses a filter (or a where clause) and returns a dataset of the records matching the filter. We use this method to retrieve the records from the Web UI project.

```
@openapi.openedge.export(type="REST", useReturnValue="false",
writeDataSetBeforeImage="true").
    @progress.service.resourceMapping(type="REST", operation="read",
URI="?filter=~{filter~}", alias="", mediaType="application/json").
    METHOD PUBLIC VOID ReadCustomerBE(
      INPUT filter AS CHARACTER,
      OUTPUT DATASET dsCustomer):

     SUPER:ReadData(filter).

    END METHOD.
```

11. Open the `Customerbe.i` file under the AppServer folder in the OrderEntry project in Project Explorer. All the columns from the database table `sports2000.Customer` including the indexes and primary key definitions are displayed as follows:

```
@openapi.openedge.entity.primarykey (fields="CustNum").

DEFINE TEMP-TABLE ttCustomer BEFORE-TABLE bttCustomer
FIELD CustNum AS INTEGER INITIAL "0" LABEL "Cust Num"
```

```
FIELD Country AS CHARACTER INITIAL "USA" LABEL "Country"
FIELD Name AS CHARACTER LABEL "Name"
FIELD Address AS CHARACTER LABEL "Address"
FIELD Address2 AS CHARACTER LABEL "Address2"
FIELD City AS CHARACTER LABEL "City"
FIELD State AS CHARACTER LABEL "State"
FIELD PostalCode AS CHARACTER LABEL "Postal Code"
FIELD Contact AS CHARACTER LABEL "Contact"
FIELD Phone AS CHARACTER LABEL "Phone"
FIELD SalesRep AS CHARACTER LABEL "Sales Rep"
FIELD CreditLimit AS DECIMAL INITIAL "1500" LABEL "Credit Limit"
FIELD Balance AS DECIMAL INITIAL "0" LABEL "Balance"
FIELD Terms AS CHARACTER INITIAL "Net30" LABEL "Terms"
FIELD Discount AS INTEGER INITIAL "0" LABEL "Discount"
FIELD Comments AS CHARACTER LABEL "Comments"
FIELD Fax AS CHARACTER LABEL "Fax"
FIELD EmailAddress AS CHARACTER LABEL "Email"
INDEX Comments  Comments  ASCENDING
INDEX CountryPost  Country  ASCENDING  PostalCode  ASCENDING
INDEX CustNum IS  PRIMARY  UNIQUE  CustNum  ASCENDING
INDEX Name  Name  ASCENDING
INDEX SalesRep  SalesRep  ASCENDING .


DEFINE DATASET dsCustomer FOR ttCustomer.
```

12. Right-click anywhere on the editor and select **Progress OpenEdge** > **Define Service Interface**. The **Define Service Interface** page opens.
13. Select all the operations in the **ABL routines** section and click **Next**. The **Edit Annotation** page opens.
14. Click the **Field Annotations** tab.
15. From the **Temp Table** drop-down list, select ttCustomer.
16. From the **Field** drop-down list, select Phone.
17. From the **Annotation** drop-down list, select Semantic Type.
18. From the **Semantic Type** drop-down list, select PhoneNumber.
19. Click **Apply** and then click **Finish**.

    Notice that the `Customerbe.i` file is annotated.
20. Save and close the `CustomerBE.cls` and the `Customerbe.i` files.

Editing the Data Object Service

We created a Data Object Service, OrderEntryService, while creating the ABL Web App project. This service appears in the Defined Services node of the project in the Project Explorer. Now we edit the service to use the annotated Business Entity that we created.

1. Right-click the OrderEntryService service in the **Project Explorer** > **OrderEntry** > **Defined Services** and click **Edit**. The **Edit an ABL Service** page opens.
2. Click **Next**. The **Create a Data Object service** page opens.
3. Select the `CustomerBE.cls` resource.
4. Click **Finish**.

   The .json file that contains metadata describing the schema and operations of the `CustomerBE` class is generated.

Note:

The *service*.json file is generated for a REST-based Data Object Service and *service*.json and *service*.gen files are generated for a WebHandler-based Data Object Service. Since we created a REST-based Data Object Service, the .json file and customer data is generated.

The URL in the *service*.json file for a WebHandler-based service is web/pdo/*serviceURI*, and for a REST-based service is rest/*serviceURI*.

Publishing the Data Object Service

The Data Object Service, OrderEntryService, uses the annotated Business Entity to implement the business logic. We now publish the service.

1. Click the **Servers** tab in the bottom pane of the **OpenEdge Server** view. The oepas1 node is in the [Stopped] state.
2. Double-click the oepas1 node. The **Overview** view appears in a new tab.
3. Click the **Open launch configuration** link. The **Edit launch configuration properties** page opens.
4. Click the **Databases** tab.
5. Select the **Show all** option button to display the sports2000 database that we connected to.
6. Select sports2000 from the list.
7. Click **Apply** and then click **OK**.
8. Select the oepas1 node in the **Servers** view and start the server by selecting the **Start** icon.

   The status is updated to Starting oepas1… in the bottom right corner of the page. When the server is ready. the state changes to [Started, Synchronized].
9. To add OrderEntryService to the oepas1 instance, right-click the oepas1 node and click **Add and Remove…**. The **Add and Remove** dialog box appears.
10. Click **Add All** to add all the services to the oepas1 instance.
11. Click **Finish**.

    The service is added and published to the server. Expand the oepas1 node to see the services. The status of the services appear as [Synchronized][Published].

Accessing the catalog details

To check that the Data Object service published correctly, we access the Data Object Service Catalog and the customer data.

1. In the **Servers** view, expand the oepas1 server instance.
2. Right-click the OrderEntryService and click **Open Catalog URL**. The URL http://localhost:8810/OrderEntry/static/OrderEntryService.json opens and displays the content of the CustomerBE dataset records.

Creating the web app in Kendo UI Designer

Now we create an OpenEdge web app, OrderEntryWebApp, using the Kendo UI Designer.

Note: For more information on the architecture and components of OpenEdge and Kendo UI Builder, see the *Kendo UI Builder by Progress: Modernizing OpenEdge Applications* guide. For more information on using Kendo UI Designer, see the Kendo UI Builder by Progress: Using Kendo UI Designer guide.

Creating the web app

We now create a simple web app, OrderEntryWebApp.

1. Click **Start** > **Progress** > **Kendo UI Builder** > **Kendo UI Builder**. The Kendo UI Builder launches as a stand-alone Electron application.
2. Click **Create App**. The **Create App** dialog box appears.
3. In the **App Name** field, enter OrderEntryWebApp.
4. In the **Location** field, enter `C:\OpenEdge\WRK`.
5. Click **Create App**. A new web app, OrderEntryWebApp, appears.

Adding the data provider

A data provider defines a single data service. We now add a data provider.

1. Open OrderEntryWebApp.
2. Click **Add Data Provider**. The **Add Data Provider** dialog box appears.
3. In the **Name** field, enter CustomerBE.
4. In the **Service URI** field, enter http://localhost:8810/OrderEntry.
5. In the **Catalog URI** field, enter http://localhost:8810/OrderEntry/static/OrderEntryService.json.
6. Select the **Auto-create Data Sources** check box.
7. Select **Anonymous** in the **Authentication Model** field.
8. Click **Add Data Provider**. The **Add Data Provider** dialog box closes and the CustomerBE data provider is added to the Data Providers list.

Modifying a data source

A data source represents tables from the data service, which you can bind to views. We now modify a data source that the data provider, CustomerBE, will provide.

1. Open OrderEntryWebApp.
2. Click **Edit** next to Customer BE data provider. The **Edit Data Provider** dialog box opens. The CustomerBE data source appears.
3. Click **Edit** next to Customer BE data source.

4. Click **Exclude All** to move all the fields to the **Excluded Fields** section. We will add only a few fields from the CustomerBE table. Only the selected fields appear in our view.
5. Drag and drop the CustName, Name, and State fields onto the **Included Fields** section.
6. Select **Customer** field in the **Included Fields** section.
7. In the **Properties** panel, in the **Label** field, enter Customer
8. Click **Save**. The **Edit Data Source** dialog box closes.
9. Click **Save**. The **Edit Data Provider** dialog box closes and the data source is added to the data provider.

Creating a module

A module is the basic unit of a web app. Each module contains one or more views that provide the functionality for a common set of features.

The module, Application, is added by default. It contains application-level pages, including the landing page and authentication page.

We now create a module, with the name, OrderEntry, and with the run-time label, Order Entry. The module name is used to form the name of a folder that stores folders and files for views created in the module, and the module label is used to identify the module in the app landing page at run time.

1. Open OrderEntryWebApp.
2. Click **Add Module**. The **Add Module** dialog box appears.
3. In the **Name** field, enter OrderEntry.
4. In the **Label** field, enter Order Entry. Spaces are allowed in the **Label** field.
5. Change the icon and color by clicking the down arrow on the **Icon/Color** image to identify the module on the web app design page (and also on the app landing page at run time). Select an icon, an icon color, and the background color.
6. Click **Add Module**. The OrderEntry module is added to the Modules list.

Creating a view

A view provides the UI for a specific function or feature within a module. We now create a view for the OrderEntry module.

1. Open OrderEntryWebApp.
2. Click **Edit** next to the **OrderEntry** module. The view design page opens.
3. Click **Add View**. The **Add View** dialog box appears.
4. In the **Name** field, enter customer-list, which is used to form the name of a folder that stores files for the view.
5. In the **Label** field, enter Customer List, which identifies the view in the web app at run time.
6. Select the **Data Grid** view option.
7. Click **Add View**. The Data Grid view is created and the view design page for the view is displayed with a simulated rendering of the view and a **VIEW PROPERTIES** pane to the right.
8. In the **VIEW PROPERTIES** pane, do the following:

a) Enter Customer List in the **Grid Title** field.

b) Select CustomerBE in the **Data Provider** field.

c) Select CustomerBE in the **Data Source** field.

9. Select the **Enable Sorting** check box.
10. Click **Save** in the toolbar above the view simulation. The view instance metadata is saved with the settings you have specified.

Viewing the metadata

The UI definition of the web app is stored as metadata. The metadata with the web app UI framework and component templates generates the web app.

1. Navigate to the `C:\OpenEdge\WRK\OrderEntryWebApp\meta` directory.
2. Open the `modules` folder. You see a file for each module, `Application.json` and `OrderEntry.json`. Close the `modules` folder.
3. Open the `OrderEntry.json` file. Notice that the children array has one child with a customer-list name, a Customer List label, and a data-grid viewType. Other properties within the child specify other property settings for the data-grid view, such as the settings for the dataProvider and dataSource properties.
4. Under the `meta` directory again, open the `dataProviders` directory.
5. Open the `CustomerBE.json` file for the data provider we created. Notice that the catalogUris, serviceUri, and the children array property that specifies the data sources for the CustomerBE data provider are displayed.

Previewing the web app

We now preview and build OrderEntryWebApp so that it can be deployed on PAS for OpenEdge.

1. Open OrderEntryWebApp.
2. Click **Generate** on the app design page. The creates a `customer-list` folder for the view in an `order-entry` module folder under `C:\OpenEdge\WRK\OrderEntryWebApp\app\src\modules`.
3. Click **Preview**.

Note: You browser settings may not allow pop-ups. Change the pop-up setting.

A new tab opens in your browser showing the landing page for the web app. If you click the module labeled Order Entry, you see the Customer List view that you just created and configured.

Creating a Web UI project

We now create a Web UI project to customize the code for the web app and also deploy it on PAS for OpenEdge.

1. Launch Developer Studio
2. Select **File** > **New** > **OpenEdge project** from the main menu bar. The **New OpenEdge Project** wizard opens and displays the **Create an OpenEdge Project** page.
3. In the **Project name** field, enter OrderEntryWebApp.
4. Clear the **Use default** check box and enter `C:\OpenEdge\WRK\OrderEntryWebApp` as the path.
5. From the **Project type configuration** drop-down list, select Web UI and click **Next**. The **Provide Web UI deploy** details page opens.
6. Retain OrderEntryWebApp in the **WebApp name** field.
7. Select the oepas1 check box in **Supported servers**. This server is the destination for your published web app and Web UI module.
8. Click **Finish**.

Customizing the Web UI project

The Web UI project is populated with many directories and content that belong to the Kendo UI Builder project we created earlier.

The `C:\OpenEdge\WRK\OrderEntryWebApp\app\src` directory contains custom files for the OpenEdge web app and its contents are picked up automatically when we build a web app in Kendo UI Builder. Do not change any files in any other directories.

Open the `\src\modules\order-entry\customer-list\controller.public.js` file in the project. This file contains most sample event handlers for the OpenEdge web app. You can see default method stubs for the `onShow` and `onRowSelect` events.

Open the `\src\modules\order-entry\customer-list\router-events.js` file in the project. This file contains primarily the `onInit` and `onHide` sample event handlers for the OpenEdge web app. You can see default method stubs for this events.

We now add the following content to the file to add specific functions:

Function/Event	Code snippet	

onInit	`onInit: function($stateParams) {`
	`  return $q(function(resolve, reject) {`
	`  console.log("onInit");`
	`  resolve({});`
	`  });`
	`},`
		When t
		the onI

onShow	`onShow: function ($scope, customData) {`
	`  // this.scope = $scope;`
	
	`  var that = this;`
	`  console.log('onShow');`
	`  try {`
	`  $scope.model.options.filterable = true;`
	
	`  $scope.$on('kendoWidgetCreated', function (event, widget) {`
	`  var dataSource;`
	
	`  console.log("Event: kendoWidgetCreated");`
	`  if (event.targetScope.vm.widget === widget) {`
	`  dataSource = event.targetScope.vm.widget.dataSource;`
	`  dataSource.bind('change', function () {`
	
	`  that.selectRecords(event.targetScope.vm.widget, 'MA', 'lightgreen');`
	`  });`
	`  }`
	`  });`
	`  } catch (ex) {`
	`  console.log("Exception: ", ex);`
	`  }`
	`  },`
		When t
		all the
		light-gr

onHide	`onHide: function(customData) {`
	`  console.log("onHide");`
	`},`
		When t
		onHide

onRowSelect	`onRowSelect: function (e) {`
	`  console.log('rowSelect', e);`
	`  try {`
	`  this.selectRecords(e.sender, 'MA', 'lightblue');`
	`  } catch (ex) {`
	`  console.log("Exception: ", ex);`
	`  }`
	`  }`
		When a
		view w
		with sta
		color.

selectRecords		Functio

Function/Event	Code snippet

	selectRecords: function (grid, state, color) {
	var dataSource,
	data,
	i,
	row;
	
	try {
	dataSource = grid.dataSource;
	data = dataSource.view();
	for (i = 0; i < data.length; i += 1) {
	if (data[i].State === state) {
	row = grid.tbody.find("tr[data-uid='" +
	data[i].uid + "']");
	row.css("background-color", color);
	grid.expandRow(row);
	}
	}
	} catch (ex) {
	console.log("Exception: ", ex);
	}
	}

After updating, the `controller.public.js` and `router-events.js` files have the following content:

```
/* global angular */
(function(angular) {

angular
   .module('viewFactories')
   .factory('orderEntryCustomerList', ['$q', function($q) {

   function OrderEntryCustomerList() {
   }

   OrderEntryCustomerList.prototype = {

   /* The resolve method could return arbitrary data,
which will be available in the "viewShowHandler" and "viewHideHandler" handler as the
customData argument */

   onInit: function($stateParams) {
      return $q(function(resolve, reject) {
         console.log("onInit");//
      resolve({});
   });
   },

   selectRecords: function (grid, state, color) {
      var dataSource,
      data,
      i,
      row;

   try {
      dataSource = grid.dataSource;
      data = dataSource.view();
      for (i = 0; i < data.length; i += 1) {
         if (data[i].State === state) {
            row = grid.tbody.find("tr[data-uid='" + data[i].uid + "']");
            row.css("background-color", color);
            grid.expandRow(row);
            }
         }
      } catch (ex) {
         console.log("Exception: ", ex);
      }
   },
   /* "customData" is the data return by the viewInitHandler handler*/
    onShow: function ($scope, customData) {
                    // this.scope = $scope;

                    var that = this;
                    console.log('onShow');
                    try {
                        $scope.model.options.filterable = true;

                        $scope.$on('kendoWidgetCreated', function (event, widget) {
                            var dataSource;

                            console.log("Event: kendoWidgetCreated");
                            if (event.targetScope.vm.widget === widget) {
                                dataSource = event.targetScope.vm.widget.dataSource;
                                dataSource.bind('change', function () {
                                  that.selectRecords(event.targetScope.vm.widget, 'MA',
 'lightgreen');
                                });
                            }
                        });
                    } catch (ex) {
                        console.log("Exception: ", ex);
                    }
                },
```

```
    /* "customData" is the data return by the viewInitHandler handler*/
    onHide: function(customData) {
       console.log("onHide");
    },
    /* Kendo event object*/
    onRowSelect: function (e) {
                    console.log('rowSelect', e);
                    try {
                        this.selectRecords(e.sender, 'MA', 'lightblue');
                    } catch (ex) {
                        console.log("Exception: ", ex);
                    }
                }
};

    return new OrderEntryCustomerList();
    }]);

})(angular);
```

Save and close each file.

Regenerate the OrderEntryWebApp using Kendo UI Builder. Then click **Publish** and choose the **Debug** option.

In the **Servers** view in Developer Studio, expand the oepas1 server instance. The oepas1 server instance displays the republish state. The modifications to the `controller.public.js` and `router-events.js` files are published. After some time, you will notice that the app, OrderEntryWebApp, is deployed on the oepas1 server instance.

Launching the web app

You can launch the web app from the Developer Studio to test it.

1. In the **Servers** view, expand the oepas1 node.
2. Right-click OrderEntryWebApp and click **Launch App**. The http://localhost:8810/OrderEntryWebApp/#/home URL opens and launches the app in your default browser.
3. Click the OrderEntry module. It lists the customer details with the selected fields.

   Observe that Row 1 and Row 5 are highlighted in light-green color and `onInit`, `onShow`, and `Event:kendoWidgetCreated` messages are displayed in the Console tab.

   Select any row with state MA, observe that the color for all the rows with state MA is changed to light-blue color. This is because we customized the `controller.public.js` and `router-events.js` files in the previous section.

3

Copyright

© 2017 Telerik AD. All rights reserved.

July 2017

Last updated with new content: Version 2.0