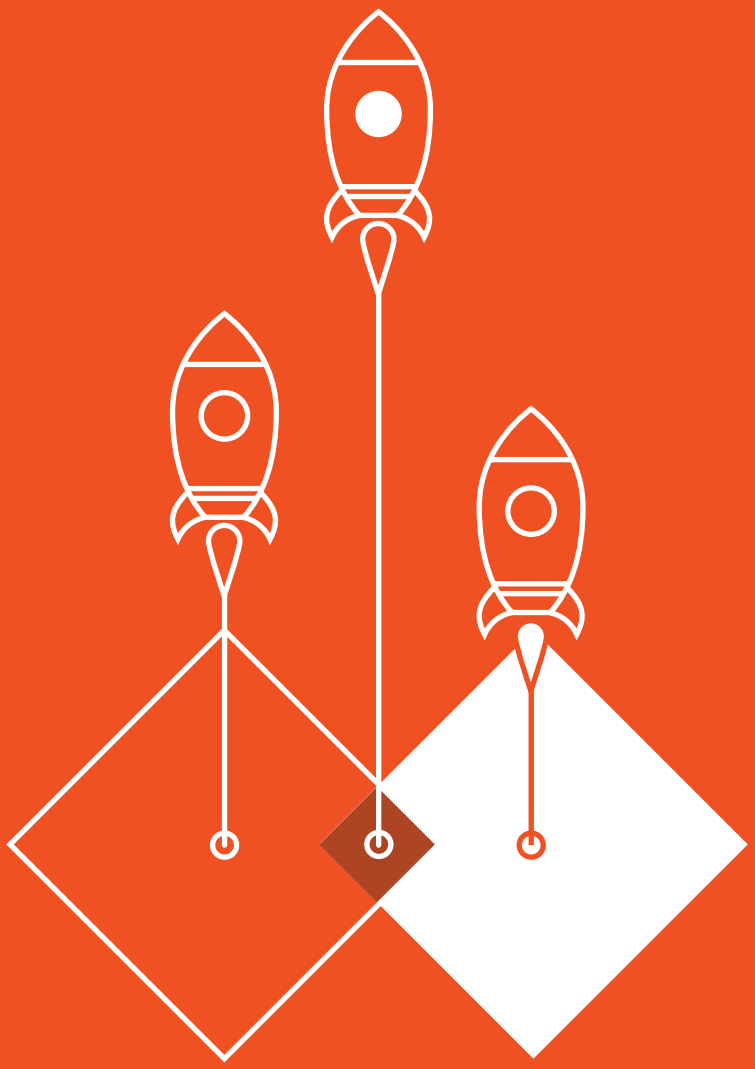
PROGRESS



USING THE JSDO WITH KENDO UI

1: Introduction

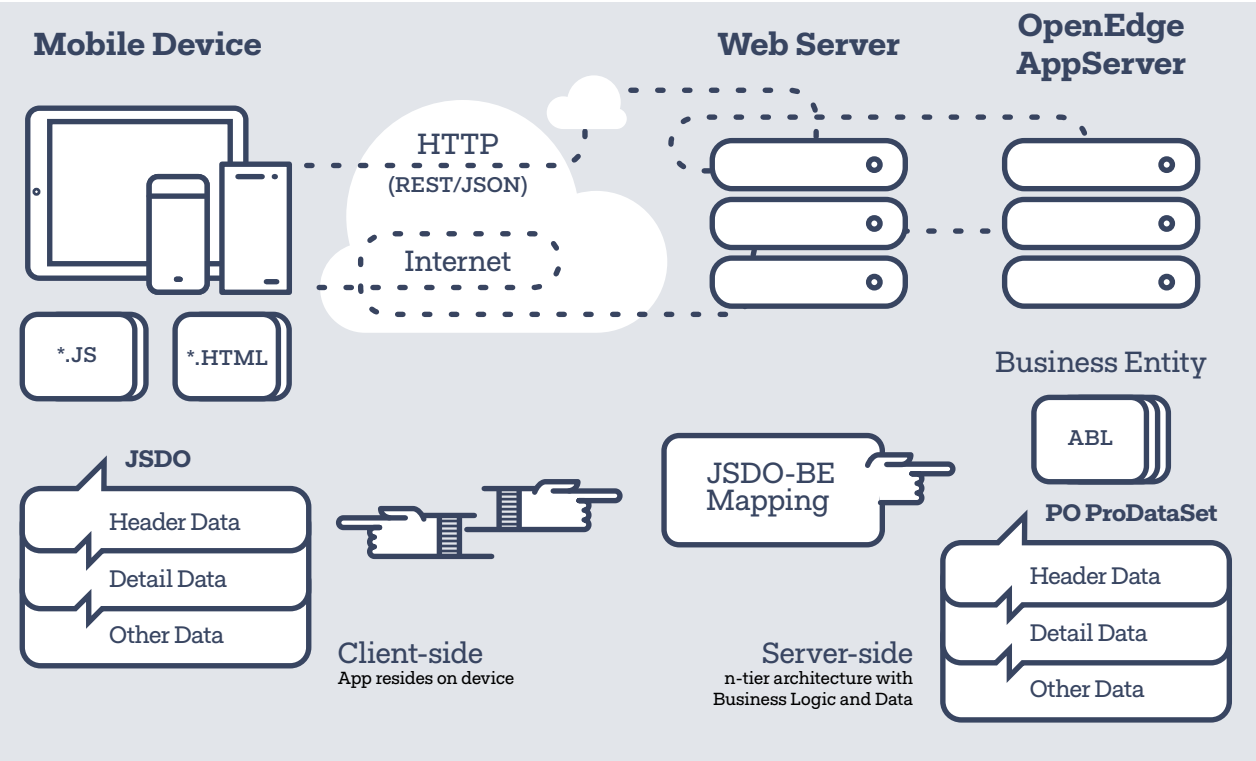
Kendo UI is a JavaScript framework that can be used to build web and mobile apps using HTML5 and JavaScript. It provides a very powerful set of UI widgets that can be bound to a data source.

The Progress JavaScript Data Object (JSDO) provides support for a complex data model and API to manipulate that data while maintaining data integrity. The JSDO catalog defines the logical schema and mapping to a remote data source. The catalog also defines an API to invoke remote business logic in addition to simple CRUD operations. The JSDO is designed to work with any Web / JavaScript framework. Similarly Kendo UI is designed to provide the best UI regardless of backend provider. Naturally the two work great together. This whitepaper describes one way to use Kendo UI with the Progress JSDO to access data and business logic on an OpenEdge AppServer.

Please note that future work is ongoing to make the communication and inter-product interaction even better, but given the openness of both technologies, it is possible to start using the two technologies immediately.

Kendo UI widgets use the Kendo UI DataSource to access local and remote data. The Kendo UI DataSource is an abstraction on top of both local data (arrays) and remote data (HTTP endpoints) that makes it much easier to request data, fill widgets with data, and then make and track changes to that data. The **transport** property in the Kendo UI DataSource configures how to perform the CRUD operations for the DataSource by defining the corresponding properties: **create**, **read**, **update**, and **destroy**.

The OpenEdge backend can be accessed through the Kendo UI DataSource using the same architecture used for Mobile in OpenEdge 11.2 and greater, the JSDO.



The JSDO manages the complex communication with an OpenEdge AppServer. It connects to the OpenEdge AppServer and executes the ABL code which accesses the data and executes the business logic. This ABL program is called a Business Entity. The JSDO only depends on JavaScript and can be integrated with many JavaScript frameworks with support for HTTP communication.

To access the OpenEdge AppServer, the CRUD operations for the Kendo UI DataSource can be configured to call the corresponding CRUD operations in the JSDO.

This document provides examples and explains how the JSDO is used with the Kendo UI components.

2: Using the JSDO from a simple HTML page

The following example shows how to access the OpenEdge backend with a simple HTML page using the JSDO:

```
<!DOCTYPE html>
<html>
<head>
    <title>Simple JSDO Usage</title>
    <script src="http://oemobiledemo.progress.com/jsdo/progress.jsdo.3.1.js"></script>
</head>
<body>
    <!-- results will be written here by JavaScript -->
    <script>
        (function () {

            var serviceURI = "http://oemobiledemo.progress.com/MobilityDemoService",
                catalogURI = serviceURI + "/static/mobile/MobilityDemoService.json";

            // create a new session object
            var session = new progress.data.Session();
            session.login(serviceURI, "", "");
            session.addCatalog(catalogURI);

            // create a JSDO
            var jsdo = new progress.data.JSDO({ name: 'dsCustomer' });
            jsdo.subscribe('AfterFill', onAfterFillCustomers, this);

            // calling fill reads from the remote OE server
            jsdo.fill();

            // this function is called after data is returned from the server
            function onAfterFillCustomers(jsdo, success, request) {
                // for each customer record returned
                jsdo.eCustomer.foreach(function (customer) {
                    // write out some of the customer data to the page
                    document.write(customer.data.CustNum + ' ' + customer.data.Name + '<br>');
                });
            }
        }());
    </script>
</body>
</html>
```

View Example

The JavaScript file **progress.jsdo.3.1.js** is included at the top of the page, just under the title. This file contains the code for the Session and JSDO objects. The current version is 3.1, however, any version can be used with Kendo UI in the way presented in this document.

Support for the JSDO is provided by two main objects:

1. The **Session** - manages authentication and session operations
2. The **JSDO** - provides the access to the OpenEdge backend by using the information in the JSDO Catalog

Once inside the JavaScript, the `serviceURI` and `catalogURI` variables are declared. These settings are used by the JSDO to access the OpenEdge service. The `serviceURI` is the URL to the web application of the Mobile Service. The `catalogURI` is the URL to the JSDO Catalog file, a JSON file containing the definition for the resources available in the OpenEdge Service. It specifies the schema and operations for each of the resources.

Developers do not need to specify any other URIs to access the data. For example, the developer can just call `fill()` to read the data from the backend and does not need to specify the URI that corresponds to the READ operation. This information is resolved internally based on the information in the catalog.

The JSDO subscribes to the `AfterFill` event. Operations to the server such as `fill()` (READ operation) and `saveChanges()` (CREATE, UPDATE, and DELETE operations) are asynchronous and need a callback to handle the response. The `AfterFill` event is called when the OpenEdge Service returns the data to the JSDO. In that event, the `foreach()` method is called to enumerate over the results and write them out to the page. The JSDO contains several other methods for handling data. Some of these methods are `getData()`, `find()`, `findById()`, `sort()`, and `addRecords()`.

Figure 2: Web Browser output shown records retrieved using the JSDO:

1. Lift TourAAA-
2. Upsilla Brent
3. Hoops
4. Go Fishing Ltd
5. Match Point Tennis
6. Match Point Tennis 2
7. Aerobics valine Ky
8. Game Set Match
9. Pihtiputaan Pyira
10. Just Joggers Limites

Update the `serviceURI` and the `catalogURI` to point to your own Mobile Service. The Online Documentation contains information on how to create a Mobile Service in Progress Developer Studio.

3: Using the JSDO from the Kendo UI Grid Component

The following example shows the JSDO working with the Kendo UI Grid. The Kendo UI Grid / Basic example from the Kendo UI demo page was used as a starting point. In addition to CRUD operations, right out of the box the Kendo UI Grid offers features such as paging, skipping records, column docking, adapting rendering and adjusting to screen size, and easy theming.

Using this example, we'll produce a visually appealing grid with basic functionality looking like this:

Figure 3: The Kendo UI Grid component showing records from an OpenEdge database

Connecting the Kendo UI Grid to the OpenEdge Service requires wiring up Kendo UI Transports to the JSDO.

```
<!DOCTYPE html>
<html>
<head>
    <title>JSDO / Kendo UI Grid Example</title>
    <link rel="stylesheet" href="http://cdn.kendostatic.com/2014.3.1316/styles/kendo.common.min.css" />
    <link rel="stylesheet" href="http://cdn.kendostatic.com/2014.3.1316/styles/kendo.default.min.css" />

    <script src="http://cdn.kendostatic.com/2014.3.1316/js/jquery.min.js"></script>
    <script src="http://cdn.kendostatic.com/2014.3.1316/js/kendo.all.min.js"></script>
    <script src="http://oemobiledemo.progress.com/jsdo/progress.jsdo.3.1.js"></script>

    <style>
        html {
            font-size: 12px;
            font-family: Arial, Helvetica, sans-serif;
        }
    </style>

</head>
<body>
    <div id="example">
        <div id="grid"></div>
    </div>
    <script>
        $(function () {


            var serviceURI = 'http://oemobiledemo.progress.com/CustomerService',
                catalogURI = serviceURI + '/static/mobile/CustomerService.json';

            // create a new session object
            var session = new progress.data.Session();
            session.login(serviceURI, '', '');
            session.addCatalog(catalogURI);

            // create a JSDO
            var jsdo = new progress.data.JSDO({ name: 'Customer' });

            // select the "grid" div with jQuery and turn it into a Kendo UI Grid
            $('#grid').kendoGrid({
                // all Kendo UI widgets use a DataSource to specify which data to display
                dataSource: {
                    transport: {
                        // when the grid tries to read data, it will call this function
                        // this could alternatively be a URL
                        read: jsdoTransportRead
                    },
                    error: function (e) {
                        console.log('Error: ', e);
                    }
                },
                height: 375,
                // setting up most of the grid functionality is as easy as toggling properties on and off
                groupable: true,
                sortable: true,
                reorderable: true,
                resizable: true,
                selectable: true,
```

```
pageable: {
                    refresh: true,
                    pageSizes: true,
                    pageSize: 10,
                    buttonCount: 5
                },
                columns: [
                  { field: 'CustNum', title: 'Cust Num', type: 'int', width: 100 },
                  { field: 'Name' },
                  { field: 'State' },
                  { field: 'Country' }
                ]
            });

            // this function is called after data is returned from the server
            function jsdoTransportRead(options) {
                jsdo.subscribe('AfterFill', function callback(jsdo, success, request) {
                    jsdo.unsubscribe('AfterFill', callback, jsdo);
                    if (success) {
                        options.success(jsdo.getData());
                    }
                    else {
                        options.error(request.xhr, request.xhr.status, request.exception);
                    }
                }, jsdo);
                jsdo.fill();
            }

        });
    </script>
</body>
</html>
```

View Example

Since Kendo UI is a JavaScript framework, the only thing required to use it is to include the common CSS file, a theme CSS file (default in this case), jQuery and the Kendo UI JavaScript library.

This example instantiates the Session and the JSDO objects, and then configures the transport in the Kendo UI DataSource properties.

All Kendo UI widgets use a DataSource to define what data they can display and manipulate. A Kendo UI DataSource can define read, update, create and destroy endpoints that are called “transports”. In this example, the read transport is set to the `jsdoTransportRead` function. This is the function the function to be called when the Kendo UI DataSource needs to perform "read". The "read" property could be a string, an object or even a function. A function reference is used so that JavaScript can be specified. Even though the function could have been defined inline, a function reference is used so that the code for jsdoTransportRead can eventually be moved into a separate JavaScript file.

The call to subscribe to the event specifies the name of the AfterFill event, the function to call back and the context. The callback is defined inline rather than just using the function reference.

The JSDO then unsubscribes the callback so that only one callback is registered for `AfterFill`. The callback function is declared inline in the jsdoTransportRead function so that the functions `options.success` and `options.error` can be called. This approach is used because the READ operation is asynchronous. The `options.success` method is called if the READ is successful. If it the READ fails, `options.error` is called and the appropriate parameters are passed so that the error handler defined for the Kendo UI DataSource can process the error.

The code in jsdoTransportRead can be changed based on the application needs. For example, if a multi-table DataSet is used, the code can be changed to point to a specific table reference within the JSDO:

```
function jsdoTransportRead(options) {
    var jsdo = this.jsdo;
    jsdo.subscribe('AfterFill',
  function callback(jsdo, success, request) {
      jsdo.unsubscribe('AfterFill', callback, jsdo);
      if (success) {
          jsdo.useRelationships = false;
          options.success(jsdo.eOrder.getData());
          jsdo.useRelationships = true;
      }
      else
          options.error(request.xhr, request.xhr.status, request.exception);
  }, jsdo);
    jsdo.fill();
}
```

The previous example uses `jsdo.eOrder.getData()` to obtain the data for eOrder in a multi-table DataSet. The `useRelationships` property in the JSDO can be set to false so that `getData()` returns all the records rather than the ones based on the relationship.

The *jsdoTransportRead* function (and the ones to support create, update and destroy) can also be customized to use any API in the JSDO, for example, `foreach()`, `sort()`, and `addRecords()` among others.

In this sample program, the code in `jsdoTransportRead()` is generic. It is not specific to a particular resource and can be moved to a separate JavaScript file so that it can be reused by other programs.

The next section shows what this would look like.

4: Wrapping The Transport Functions In A Kendo UI Class

While JavaScript itself does not have the concept of a "class" the way traditional programming languages do, classes can be created in JavaScript by using functions. Kendo UI provides a convenient extension method which makes creating classes much easier. By calling `kendo.Class.extend`, a new class can be defined, complete with constructor.

```
// Create a base class
var JSDOTransport = kendo.Class.extend({
    // The `init` method will be called when a new instance is created
    init: function (param1, param2) {
        // todo
    }
});
```

Now that the class has been defined, it is possible to move all of the logic and functions for the transports into the class in a way that will allow this class to be used over and over again with different service URIs and resource names.

```
// Begin class declaration
var JSDOTransport = kendo.Class.extend({

    // The `init` method will be called when a new instance is created
    init: function (serviceURI, catalogURI, resourceName) {

        // Create and configure the session object
        this._createSession(serviceURI, catalogURI);

        // Create the JSDO
        this.jsdo = new progress.data.JSDO({ name: resourceName });

        // Create proxies to internal methods to maintain the correct 'this' reference
        this.transport = {
            read: $.proxy(this._read, this),
            create: $.proxy(this._create, this),
            update: $.proxy(this._update, this),
            destroy: $.proxy(this._destroy, this)
        }
    },

    // methods with an "_" are private and are only to be used by the class
    _createSession: function (serviceURI, catalogURI) {
        this.session = new progress.data.Session();
        this.session.login(serviceURI, '', '');
        this.session.addCatalog(catalogURI);
    },

    // the transports needed by the DataSource
    _read: function (options) {
        var jsdo = this.jsdo;

        jsdo.subscribe('AfterFill', function callback(jsdo, success, request) {
            jsdo.unsubscribe('AfterFill', callback, jsdo);
            if (success)
                options.success(jsdo.getData());
            else
                options.error(request.xhr, request.xhr.status, request.exception);
        }, jsdo);

        jsdo.fill();
    },

    _create: function (options) {
        var jsdo = this.jsdo;
        var jsrecord = jsdo.add(options.data);

        jsdo.subscribe('AfterSaveChanges', function callback(jsdo, success, request) {
            jsdo.unsubscribe('AfterSaveChanges', callback, jsdo);
            var data;
            if (success) {
                if (request.batch
                    && request.batch.operations instanceof Array
                    && request.batch.operations.length == 1) {
                    data = request.batch.operations[0].jsrecord.data;
                }
                options.success(data);
            }
            else
                options.error(request.xhr, request.xhr.status, request.exception);
        }, jsdo);
```

```
        jsdo.saveChanges();
    },

    _update: function (options) {
        var jsdo = this.jsdo;
        var jsrecord = jsdo.findById(options.data._id);

        try {
            jsdo.assign(options.data);
        } catch (e) {
            options.error(null, null, e);
        }

        jsdo.subscribe('AfterSaveChanges', function callback(jsdo, success, request) {
            jsdo.unsubscribe('AfterSaveChanges', callback, jsdo);
            var data;
            if (success) {
                if (request.batch
                    && request.batch.operations instanceof Array
                    && request.batch.operations.length == 1) {
                    data = request.batch.operations[0].jsrecord.data;
                }
                options.success(data);
            }
            else
                options.error(request.xhr, request.xhr.status, request.exception);
        }, jsdo);

        jsdo.saveChanges();
    },

    _destroy: function (options) {
        var jsdo = this.jsdo;
        var jsrecord = jsdo.findById(options.data._id);

        try {
            jsdo.remove();
        } catch (e) {
            options.error(null, null, e);
        }

        jsdo.subscribe('AfterSaveChanges', function callback(jsdo, success, request) {
            jsdo.unsubscribe('AfterSaveChanges', callback, jsdo);
            if (success)
                options.success([]);
            else
                options.error(request.xhr, request.xhr.status, request.exception);
        }, jsdo);

        jsdo.saveChanges();
    }
});
// End class declaration
```

The class `init` method is called when a new instance of the class is instantiated. The constructor requires that the `serviceURI`, `catalogURI` and `resourceName` are provided to the initialization of the class. The class then takes care of creating a new session object and a JSDO. It also wraps all of the transport functions so that they will be available off of the class object once it has been created.

In JavaScript, the value of the 'this' keyword constantly changes, depending on where it is called within the code. This is known as "lexical scoping". Using the jQuery method $.proxy *to call internal methods ensures that the value of 'this' is actually that of the class and not something else when it is eventually used in the private _read, _create, _update and _destroy methods.*

The `_read`, `_create`, `_update` and `_delete` functions use the JSDO APIs to perform the corresponding operation and eventually call `jsdo.saveChanges()` to synchronize the changes with the OpenEdge Service. The code in the `AfterSaveChanges` callback returns the data to the Kendo UI DataSource by calling `options.success()`, or handles an error by called `options.error()`.

In the _create method, the `jsdo.add()` function adds the record to memory on the JSDO. Calling `saveChanges()` commits those changes to the OpenEdge server and then handles the response from the server making sure that the DataSource and the server contain the same data.

In the `_update` method, the record that needs to be updated is retrieved by calling the `jsdo.findById()` `method`. The retrieved item is then updated and calling `saveChanges()` commits that update to the OpenEdge server. The callback from the server is then handled to ensure that the DataSource and the server now both contain the same data.

In the _destroy method, the `jsdo.findById()` function is once again used to find the record that needs to be deleted. It is then removed in memory by calling the `jsdo.remove()` method. Calling `saveChanges()` once again saves the changes from memory to the OpenEdge Server. Lastly, an empty array is passed back to the Kendo UI DataSource since there are no records to return from a destroy operation.

The word "destroy" is used in place of the word "delete" on the Kendo UI DataSource Transports because "delete" is a keyword in JavaScript.

It is now possible to instantiate the class, providing the proper parameters. In the following example, the JSDOTranport class has been moved into a different file and included in the HTML page.

```
<!DOCTYPE html>
<html>
<head>
    <title>Progress / Kendo UI Demo</title>
    <link rel="stylesheet" href="http://cdn.kendostatic.com/2014.3.1316/styles/kendo.common.min.css" />
    <link rel="stylesheet" href="http://cdn.kendostatic.com/2014.3.1316/styles/kendo.default.min.css" />

    <script src="http://cdn.kendostatic.com/2014.3.1316/js/jquery.min.js"></script>
    <script src="http://cdn.kendostatic.com/2014.3.1316/js/kendo.all.min.js"></script>

    <script src="http://oemobiledemo.progress.com/jsdo/progress.jsdo.3.1.js"></script>
    <script src="jsdoTransport.js"></script>
</head>
<body>
    <div id="example">
        <div id="grid"></div>
    </div>
    <script>
        $(function () {
            var serviceURI = 'http://oemobiledemo.progress.com/CustomerService',
                catalogURI = serviceURI + '/static/mobile/CustomerService.json',
                resourceName = 'Customer';

            // create a new instance of the JSDO transport class for 'Customer'
            // the class was included as part of the jsdoTransport file
            var customer = new JSDOTransport(serviceURI, catalogURI, resourceName);
```

```
            $("#grid").kendoGrid({
                dataSource: {
                    // define transports as the class functions
                    transport: customer.transport,
                    schema: {
                        model: {
                            id: '_id'
                        }
                    },
                    error: function (e) {
                        console.log('Error: ', e);
                    }
                },
                height: 400,
                groupable: true,
                reorderable: true,
                resizable: true,
                sortable: true,
                pageable: {
                    refresh: true,
                    pageSizes: true,
                    pageSize: 10,
                    buttonCount: 5
                },
                editable: 'inline',
                toolbar: ['create'],
                columns: [
                    { field: 'CustNum', title: 'Cust Num', type: 'int', width: 100 },
                    { field: 'Name' },
                    { field: 'State' },
                    { field: 'Country' },
                    { command: ['edit', 'destroy'], title: ' ', width: '250px' }
                ]
            });
        });
    </script>
</body>
</html>
```

The code is now much cleaner. All that is needed to create proper Kendo UI DataSource transports that use a JSDO is to create a new instance of the JSDOTransport class and pass in the `serviceURI`, `catalogURI` and `resourceName` parameters.

5: Sharing a Kendo UI DataSource Between Widgets

While a Kendo UI DataSource can be defined in a widget configuration object (as we have seen so far), it can also be defined as a stand-alone object. This allows it to be used by more than one widget. Aside from rich data management widgets, Kendo UI also includes an extensive charting a data visualization library. These widgets use new HTML5 SVG standards to draw animated and interactive charts right in the browser. They also take care of handling older browsers (back to IE 7) where SVG is not supported.

In order to add a chart to this page which displays the same data as the grid, it is first necessary to pull the DataSource declaration out into a separate object.

```
var serviceURI = base + 'http://oemobiledemo.progress.com/CustomerService',
    catalogURI = base + '/static/mobile/CustomerService.json',
    resourceName = 'Customer';

// create a new instance of the JSDO transport class for 'Customer'
var customer = new window.app.JSDOTransport(serviceURI, catalogURI, resourceName);

// create a datasource that can be shared between widgets
var customerDS = new kendo.data.DataSource({
    transport: customer.transport,
    schema: {
        model: {
            id: '_id'
        }
    },
    error: function (e) {
        console.log('Error: ', e);
    }
});

// the grid's dataSource can now be set directly to the customerDS object
$("#grid").kendoGrid({
    dataSource: customerDS,
    height: 350,
    groupable: true,
    reorderable: true,
    resizable: true,
    sortable: true,
    pageable: {
        refresh: true,
        pageSizes: true,
        pageSize: 10,
        buttonCount: 5
    },
    editable: "inline",
    toolbar: ["create"],
    columns: [
      { field: "CustNum", title: "Cust Num", type: "int", width: 100 },
      { field: "Name" },
      { field: "State" },
      { field: "Country" },
      { command: ["edit", "destroy"], title: " ", width: "250px" }
    ]
});
```

It is now possible to add any other Kendo UI widget or data visualization component and use the exact same DataSource.

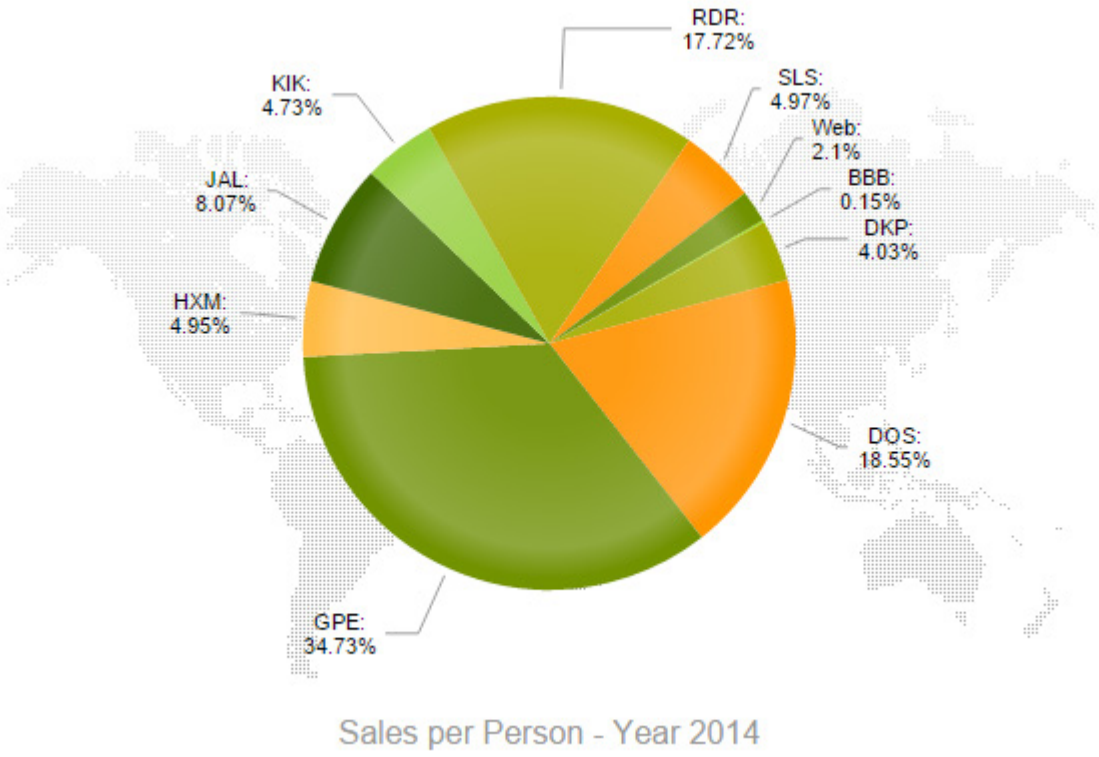
6: Using the JSDO with Kendo UI Charts

Kendo UI has an extensive charting and data visualization library. OpenEdge data retrieved using the JSDO can be displayed using Kendo UI Charts. The data retrieved by the JSDO is processed in JavaScript then passed to a Kendo UI Chart using the format that it expects. The Kendo UI DataSource is not used in these examples.

The following examples are included in the zip file:

- Pie Chart: The Sales per Person for Year 2014 chart, uses the JSDO to call an INVOKE operation called `MonthlySales()` that returns the monthly sales per `SalesRep`, the data is processed to calculate the total for the sales and produce the series that the Kendo UI Pie Chart component uses.

Figure 4: The Kendo UI Pie Chart component showing sales information from an OpenEdge database



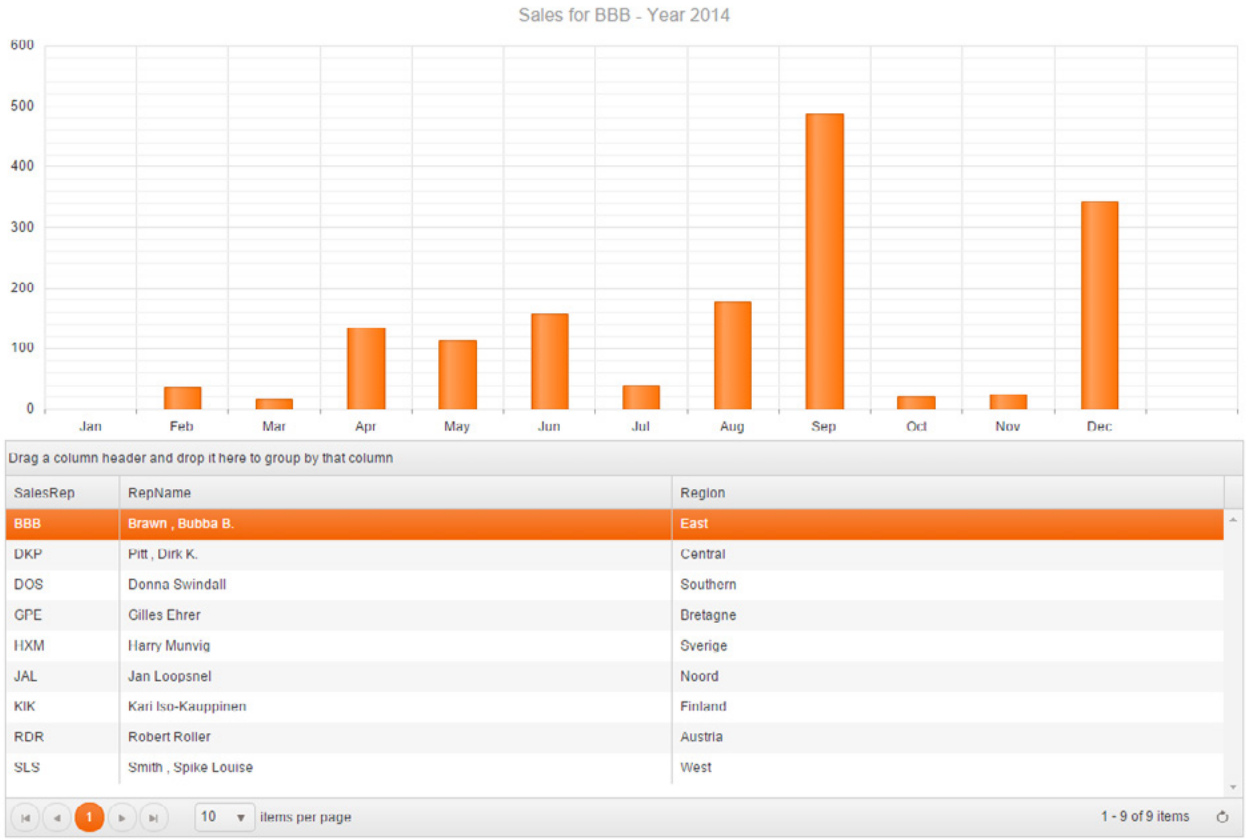
Notes:

This example is available at: http://oemobiledemo.progress.com/jsdo/example015.html

The source code for these and other sample programs using Kendo UI can be downloaded from the following location: https://community.progress.com/community_groups/openedge_development/m/documents/1502.aspx

Bar Chart: The Monthly Sales for Year 2014 chart, combines a Kendo UI Grid with a Kendo UI Bar Chart. It uses the same INVOKE operation as the previous example, but processes the data to calculate the total sales per month for the `SalesRep` selected in the grid. If no `SalesRep` is selected, a total for all the `SalesRep` is shown.

Figure 5: A sample app combining the Kendo UI Bar Chart and Grid components



Notes:

This example is available at: http://oemobiledemo.progress.com/jsdo/example016.html

7: References

The following links point to the documentation for Kendo UI and for OpenEdge related to this white paper:

- http://docs.telerik.com/kendo-ui/framework/datasource/overview
- http://docs.telerik.com/kendo-ui/api/javascript/data/datasource
- http://demos.telerik.com/kendo-ui/datasource/index
- http://demos.telerik.com/kendo-ui/grid/index
- http://demos.telerik.com/kendo-ui/grid/hierarchy
- http://demos.telerik.com/kendo-ui/pie-charts/index
- http://documentation.progress.com/output/OpenEdge114/openedge114/#page/dvmad/OpenEdge.049.html#
- http://documentation.progress.com/output/OpenEdge114/openedge114/#page/pdsoe/OpenEdge.0287.html#wwconnect_header
- http://documentation.progress.com/output/OpenEdge115/openedge115/#page/pdsoe/building-mobile-applications.html#wwconnect_header