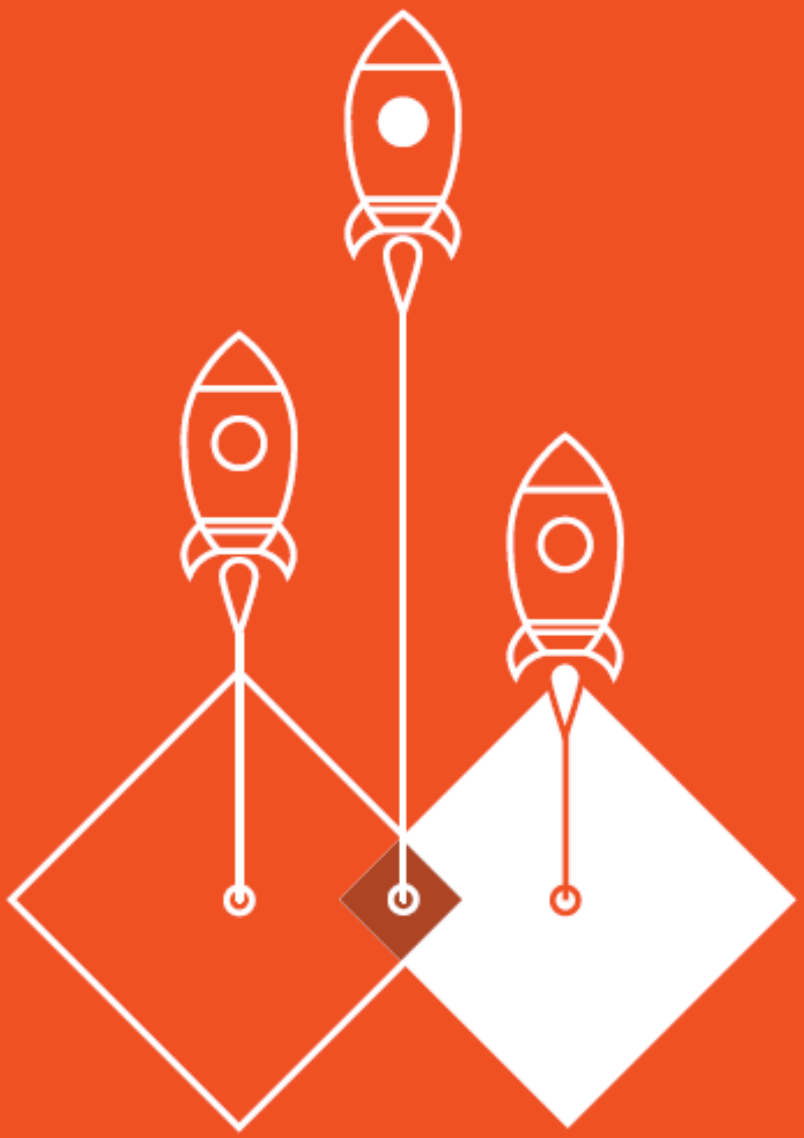
PROGRESS



ИСПОЛЬЗОВАНИЕ JSDO С KENDO UI

ВВЕДЕНИЕ

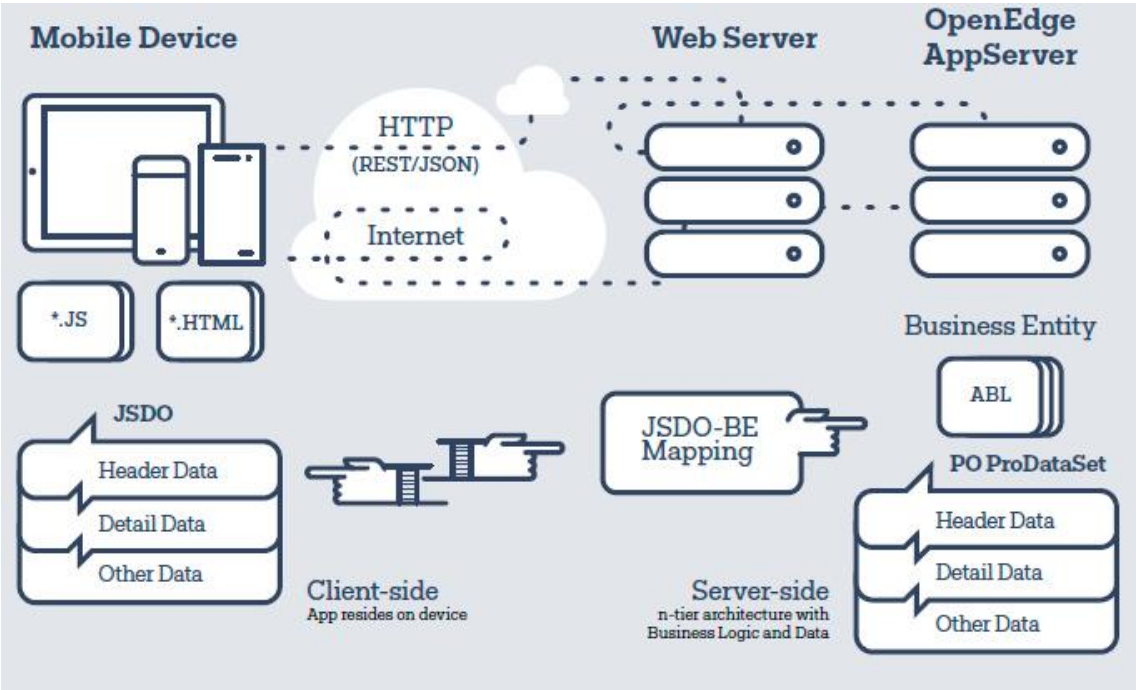
Kendo UI – это JavaScript-фреймворк, который может быть использован для создания веб- и мобильных приложений с использованием HTML5 и JavaScript. Он обеспечивает набор мощных виджетов, которые могут быть связаны с источниками данных.

Обратите внимание, работы по улучшению взаимодействия этих технологий продолжаются, но, учитывая их открытость, можно начать использовать их прямо сейчас.

Progress JavaScript Data Object (JSDO) предоставляет собой комплексную модель данных и API для управления этими данными при сохранении их целостности. JSDO-каталог определяет логическую схему и связь с удалённым источником данных. В каталоге также есть API для вызова удалённой бизнес-логики в дополнение к простым операциям CRUD (Create, Read, Update, Delete). JSDO разработан для работы с любым Web/JavaScript фреймворком. И естественно Progress JSDO и Kendo UI отлично работают вместе. В этом документе описывается один из способов их совместного использования для доступа к данным и бизнес логике на OpenEdge AppServer.

Виджеты Kendo UI используют Kendo UI DataSource для доступа к локальным и удалённым данным. Kendo UI DataSource является абстракцией над локальными и удалёнными данными, что позволяет гораздо легче запрашивать данные, заполнять ими виджеты, а затем отслеживать изменения этих данных. **Транспортное свойство** в Kendo UI DataSource определяет, как выполнять операции CRUD в DataSource, определив соответствующие свойства: **создание**, **чтение**, **обновление**, и **удаление**.

OpenEdge сервер может быть доступен через Kendo UI DataSource с использованием той же архитектуры, которая использовалась для Mobile в OpenEdge 11.2 и выше – JSDO.



JSDO управляет комплексным взаимодействием с OpenEdge AppServer. Он подключается к OpenEdge AppServer и выполняет ABL-код, который получает доступ к данным и запускает бизнес-логику. Такая ABL -программа называется бизнес-сущностью. JSDO зависит только от JavaScript и может быть интегрирован со многими JavaScript фреймворками с поддержкой HTTP-взаимодействий.

Чтобы получить доступ к OpenEdge AppServer, операции CRUD для Kendo UI DataSource могут быть настроены для вызова соответствующих CRUD-операций в JSDO. Этот документ содержит примеры и объясняет, как JSDO используется с компонентами пользовательского интерфейса Kendo.

Использование JSDO из простой HTML страницы

Следующий пример демонстрирует, как получить доступ к OpenEdge серверу прямо из простой HTML-страницы с помощью 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>
```

Посмотреть пример онлайн

В этом коде JavaScript-файл **progress.jsdo.3.1.js**, включённый в верхней части приведённого примера сразу под тегом <title>, содержит код для объектов сессии и JSDO. Текущая версия - 3.1. Однако, и любая другая версия может быть использована с Kendo UI в данном документе.

Поддержка для JSDO обеспечивается двумя основными объектами:

1. **Сессия** – управляет аутентификацией и операциями сессии.
2. **JSDO** – обеспечивает доступ к серверу OpenEdge, используя информацию в каталоге JSDO.

Объявляем JavaScript-переменные serviceURI и catalogURI. Эти настройки используются JSDO для доступа к сервису OpenEdge. **serviceURI** – это адрес веб-приложения в мобильном сервисе OpenEdge. **catalogURI** – это адрес файла JSDO-каталога (файл JSON, содержащий определение ресурсов доступных в OpenEdge-сервисе). Это определяет схему и операции для каждого из ресурсов.

Разработчикам нет необходимости указывать любые другие URI-адреса для доступа к данным. Например, разработчик может просто вызвать функцию `fill()`, чтобы считать данные с сервера и ненужно указывать URI, который соответствует операции чтения. Эта информация получена на основе данных, содержащихся в каталоге.

JSDO подписывается на событие AfterFill. Операции к серверу, такие как `fill()` (операция чтения) и `saveChanges()` (операции создания, обновления и удаления), являются асинхронными и требуют callback-функцию для обработки ответа сервера. Событие AfterFill вызывается, когда OpenEdge-служба возвращает данные в JSDO. В этом событии метод `foreach()` вызывается для перебора присланных результатов и отображения их на этой странице. JSDO также содержит ряд других методов для обработки данных. Например, `getData()`, `find()`, `findById()`, `sort()`, и `addRecords()`.

Рисунок 2: отображение записей, извлекаемых с помощью JSDO, в Веб-браузере

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

Попробуйте заменить serviceURI и catalogURI на свой собственный мобильный сервис. Информацию о том, как создать мобильный сервис в Progress Developer Studio вы найдёте в онлайн-документации.

Использование JSDO из компонента UI Grid

В следующем примере демонстрируется работа JSDO с Kendo UI Grid. Простой пример с демонстрационной страницы Kendo UI был использован в качестве отправной точки. В дополнение к CRUD-операциям, прямо из коробки Kendo UI Grid предлагает такие функции как пагинация (разбитие на страницы), пропуск записей, разбитие на колонки, адаптации размера к разрешению экрана и лёгкая смена тем оформления.

Используя этот пример, мы получаем визуально привлекательную сетку с базовой функциональностью:

Рисунок 3: Компонент Kendo UI Grid с записями из OpenEdge-базы данных Sports

Drag a column header and drop it here to group by that column

Cust Num	Name	State	Country
1	Lift Tours123	LA	USA
2	Urpon Frisbee	Uusima	Finland
3	Hoops	GA	USA
4	Go Fishing Ltd	Middlesex	United Kingdom
5	Match Point Tennis	MA	USA
6	Fanatical Athletes	AL	United Kingdom
7	Aerobics valine Ky	Uusimaa	Finland
8	Game Set Match	AL	USA
9	Pihtiputaan Pyora	Etela-Savo	Finland
10	Just Joggers Limited	Sussex	United Kingdom

1 2 3 4 5 ... 10 items per page 1 - 10 of 1122 items

Подключение Kendo UI Grid к OpenEdge-сервису требует настройки Kendo UI Transports в 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>
```

Посмотреть пример онлайн

Так как Kendo UI это JavaScript-фреймворк, то единственное что необходимо для его использования, это включить общий файл CSS, файл CSS темы (по умолчанию в данном примере), библиотеки JQuery и Kendo UI JavaScript.

Этот пример создаёт объекты сессии и JSDO, а затем настраивает транспорт в свойствах DataSource Kendo UI.

Все виджеты Kendo UI используют DataSource, чтобы определить какими данными они могут манипулировать. Kendo UI DataSource может определить конечные точки чтения, обновления, создания и удаления, которые называются «Транспорты». В этом примере транспортом чтения установлена функция jsdoTransportRead – это функция, которая будет вызвана, когда Kendo UI DataSource должен прочитать данные. Свойство «Read» может быть строка, объект или даже функция. Функция может быть определена сразу, либо это может быть ссылка на функцию jsdoTransportRead, которая помещена в отдельный файл JavaScript.

В вызове `subscribe()` указываются событие AfterFill, callback-функция и контекст. Callback-функция определяется сразу, а не просто с помощью ссылки на функцию.

JSDO затем отписывается от обратного вызова, так что только одна callback-функция регистрируется на AfterFill. Callback-функция объявлена «инлайново» (сразу идёт описание функции) в функции jsdoTransportRead так, что методы options.success и options.error тоже вызываются в нужный момент. Этот подход используется потому, что операция чтения асинхронная. Метод options.success вызывается, если чтение выполнено успешно. Если операция чтения не удаётся, вызывается метод options.error и соответствующие параметры передаются таким образом, что обработчик ошибок определённый для Kendo UI DataSource может обрабатывать ошибку.

Код в jsdoTransportRead может быть изменён в зависимости от потребностей приложения. Например, если используется многотабличный DataSet, код может быть изменён, чтобы указывать на конкретную таблицу в 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();
}
```

В предыдущем примере используется jsdo.eOrder.getData() для получения данных для eOrder в многотабличном DataSet. Свойство useRelationships в JSDO может быть установлено равным false, так что GetData() будет возвращать все записи, а не только на основании связей.

В этом примере код в jsdoTransportRead() является универсальным. Это не относится к конкретному ресурсу и может быть перемещён в отдельный файл JavaScript, так что его можно использовать повторно другими программами.

Функция jsdoTransportRead (и другие которые поддерживают создание, обновление и удаление) также может быть настроена для использования любого API в JSDO, например, foreach(), sort(), addRecords() и другие.

В следующем разделе показано, как это будет выглядеть.

Упаковка транспортной функции класс Kendo UI

В то время как сам JavaScript не имеет понятия «класса» как у традиционных языков программирования, классы могут быть созданы в JavaScript с помощью функций.

Kendo UI обеспечивает удобный метод расширения, который позволяет создавать классы гораздо проще. С помощью kendo. Class.extend, новый класс может быть определён, вместе с конструктором.

```
// 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
    }
});
```

Теперь, когда класс был определён, возможно переместить всю логику и транспортные функции в класс, что позволит использовать этот класс снова и снова с различными URI сервисов и именами ресурсов.

```
// 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
```

Метод класса init() вызывается, когда создаётся новый экземпляр класса. Конструктор требует, чтобы serviceURI, catalogURI и resourceName были переданы в конструктор класса. Затем класс заботится о создания нового объекта сеанса и JSDO. Также в нем описаны все транспортные функции, так что они будут доступны после того как создан объект класса.

В JavaScript значение переменной 'this' постоянно меняется в зависимости от места его использования в коде. Это известно, как «лексический контекст». Использование JQuery метода $.proxy для вызова внутренних функций гарантирует, что значение 'this' на самом деле относиться к данному классу, а не что-то ещё, когда оно, в конечном счёте используется в private методах _read, _create, _update и _destroy.

В _read, _create, _update и _delete функциях используются API-интерфейсы JSDO для выполнения соответствующей операции и в конечном итоге вызывают jsdo.saveChanges() для синхронизации изменений с OpenEdge-сервисом. Код в функции обратного вызова AfterSaveChanges возвращает данные в Kendo UI DataSource, выполнив options.success(), или обрабатывает ошибки методом options.error().

В методе _create функция jsdo.add() добавляет запись в память JSDO. Вызов SaveChanges() посылает эти изменения на OpenEdge сервер, а затем обрабатывает ответ от сервера, убедившись, что DataSource и сервер содержат одинаковые данные.

В методе _update запись, которая должна быть обновлена, извлекается путём вызова jsdo.findById(). Извлечённая запись изменяется, а затем вызывается метод SaveChanges() посылающий это изменение на OpenEdge сервер. Обратный вызов от сервера обрабатывается, чтобы убедиться, что источник данных и сервер теперь оба содержат одинаковые данные.

В методе _destroy функция jsdo.findById() вновь используется для поиска записи, которая должна быть удалена. Затем её удаляют из памяти с помощью вызова jsdo.remove(). Вызов SaveChanges() ещё раз сохраняет изменение из памяти на OpenEdge сервер. Наконец, пустой массив передаётся обратно в DataSource Kendo UI, так как операция удаления не возвращает никаких записей.

Ключевое слово «destroy» используется вместо «delete», потому что «delete» является ключевым словом в JavaScript.

Теперь можно создать экземпляр класса, обеспечивая надлежащие параметры. В следующем примере класс JSDOTranport был перемещён в другой файл и включён в страницу HTML.

```
<!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>
```

Код стал гораздо чище. Всё, что нужно для создания надлежащего транспорта Kendo UI DataSource, который используют JSDO, это создать новый экземпляр класса JSDOTransport и передать в параметрах serviceURI, catalogURI и resourceName.

Совместное использование Kendo UI DataSource между виджетами

В то время как Kendo UI DataSource может быть определён в виджете как объект конфигурации (как мы видели ранее), он также может быть определён как автономный объект(stand-alone). Это позволяет использовать его большим количеством виджетов. Помимо богатой коллекции виджетов управления данными, Kendo UI также включает в себя обширную библиотеку визуализации данных.

Эти виджеты используют новые стандарты HTML5 SVG для отображения анимированных и интерактивных графиков прямо в браузере. Они также заботятся о работе в старых браузерах (IE 7), где SVG не поддерживается.

Для того чтобы добавить диаграмму на той же странице, которая отображает данные в Grid, то в первую очередь необходимо вынести объявление DataSource в отдельный объект.

```
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" }
    ]
});
```

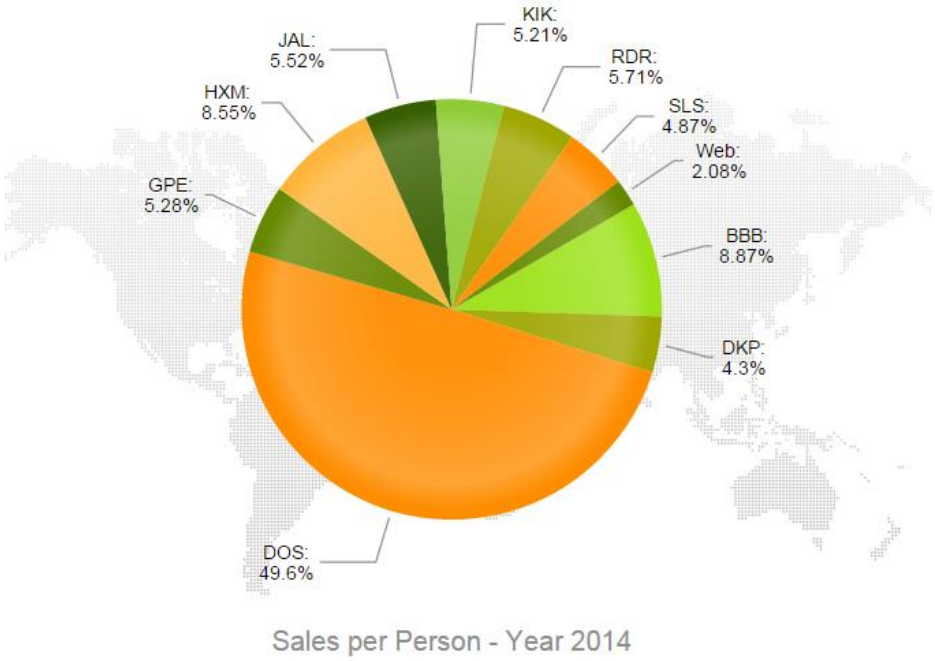
Теперь можно добавить любой виджет Kendo UI или компонент визуализации данных используя один и тот же источник данных.

Использование JSDO с Kendo UI Charts

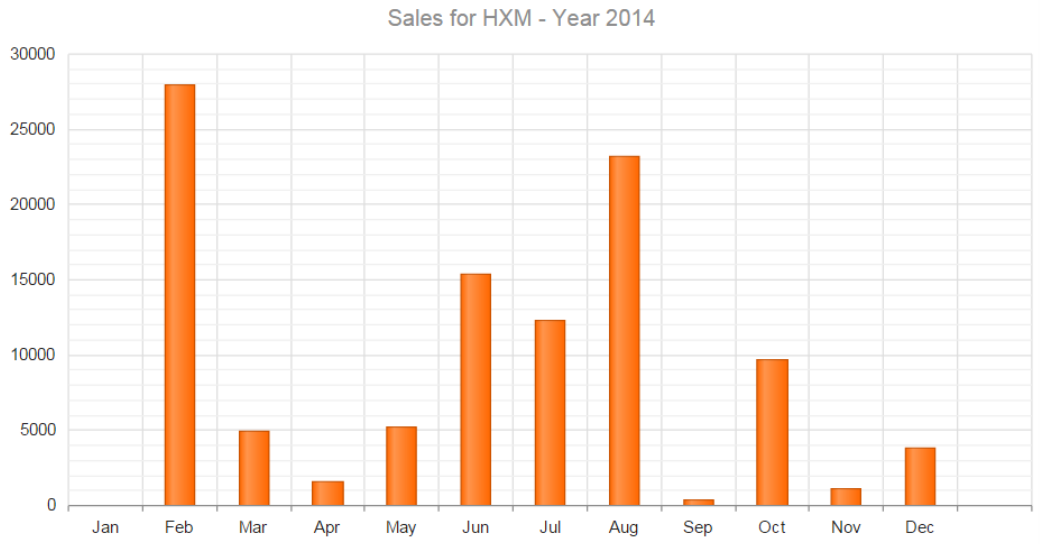
Kendo UI имеет обширную библиотеку для построения графиков и визуализации данных. Данные OpenEdge полученные с помощью JSDO могут отображается с помощью Kendo UI Charts. Данные извлекаемые JSDO обрабатываются в JavaScript, затем передаются в Kendo UI Charts с использованием ожидаемого виджетом формата. Kendo UI DataSource не используется в этих примерах.

Следующая круговая диаграмма использует JSDOдля вызова операции с именем MonthlySales(), которая возвращает ежемесячные продаж по SalesRep и с помощью Kendo UI Charts отображаются в графическом виде.

Этот пример доступен по адресу следующей ссылке.



Следующая гистограмма, график продаж за 2014 год, сочетает в себе Kendo UI Grid и Kendo UI Bar Chart. Здесь используется таже INVOKE-операция, что и в предыдущем примере, но обрабатываются данные для расчёта общего объёма продаж за месяц по SalesRep выбранного в Grid. Этот пример доступен по следующей ссылке.



Drag a column header and drop it here to group by that column

SalesRep	RepName	Region
BBB	Brawn , Bubba B.	East
DKP	Pitt , Dirk K.	Central
DOS	Donna Swindall	Southern
GPE	Gilles Ehrer	Bretagne
HXM	Harry Munvig	Sverige
JAL	Jan Loopsnel	Noord
KIK	Kari Iso-Kauppinen	Finland
RDR	Robert Roller	Austria
SLS	Smith , Spike Louise	West

1 10 items per page 1 - 9 of 9 items

Ссылки

Список ссылок на разделы документации по Keno UI и OpenEdge, который были использованы для написания этой статьи:

- 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/OpenEdge115/openedge115/#page/pdsoe/building-mobile-applications.html#wwconnect_header