

# HD2dC04wl – Customers Application- Lite

## Product and Focus

HANA Platform/oData

## Target Audience

Undergraduate/Graduate  
Beginner to Intermediate

## Author

Ross Hightower

## MOTIVATION

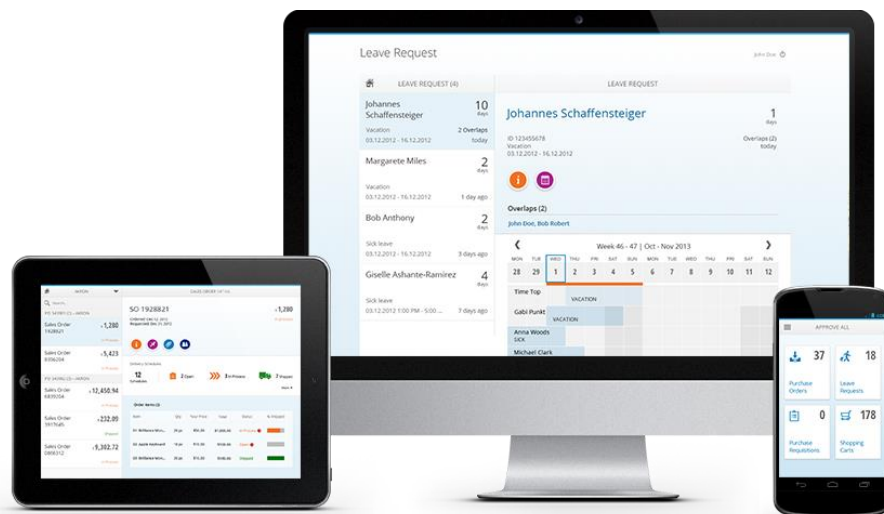
This case describes the user of oData services to create a simple SAPUI5 application.

## PREREQUISITES

HD1C01 – Hello World MVC

HD1dC02I – Create the Persistence Model - Lite

HD1dC03I – oData Services - Lite<sup>1</sup>



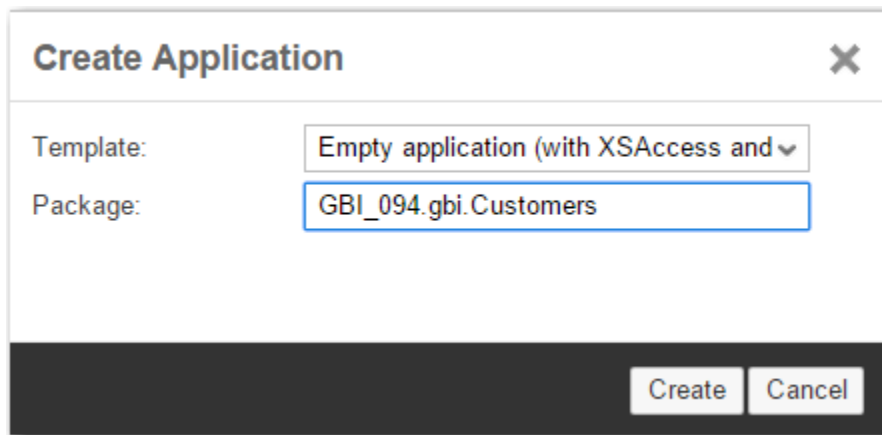
<sup>1</sup> This case can be used with either the Lite or full version of the Core Data Services curriculum

## Master/Detail Application

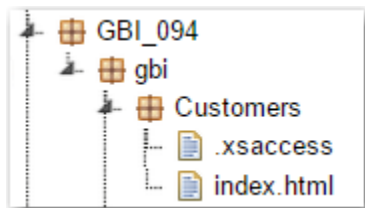
This cases builds on the previous two cases in this series to develop a master/detail application which allows the user view information about customers.

### Create the Application Packages


Login to the WDW and locate the gbi package you created in case HD1dC2w. Right-click the gbi package and choose **Create Application**. Choose the SAP UI5 Hello World Template and then add **Customers** to the Package.



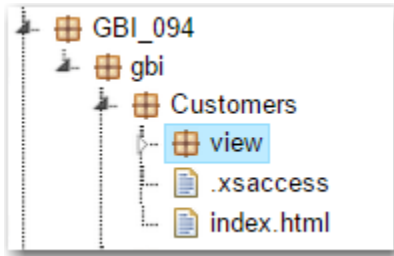
The package is created and basic application is created.



Note there is no .xsapp file. That file is unnecessary because in the oData services case we created that file in the gbi package. It applies to the entire directory structure below it. However, the .xsaccess file was created because it is possible to create different levels of access for different packages.

You can run the application if you want by selecting the index.html file and clicking . This is the standard SAPUI5 Hello World app.

Now create the **view** package shown in the image below.



The basic structure of the application is complete. Now let's add some content.

### Create the Application

The application follows a standard structure for an SAPUI5 application. The index.html file bootstraps the SAPUI5 libraries and creates a Component which encapsulates the application. The definition of the component is included in a file called Component.js. The name and location of this file is standard and cannot be altered. The various view and controller files are located in a package called view.

For a detailed explanation of the code see the case HD1C01 – Hello World MVC

### index.html

Replace the code in the index.html file with this code.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <title>GBI Customers</title>

  <script id="sap-ui-bootstrap"
    type="text/javascript"
    src="/sap/ui5/1/resources/sap-ui-core.js"
    data-sap-ui-theme="sap_bluecrystal"
    data-sap-ui-libs="sap.m"
    data-sap-ui-xx-bindingSyntax="complex"
    data-sap-ui-resourceroots = '{
      "gbi" : "./"
    }'></script>

  <script>

    new sap.m.Shell("Shell",{
      app: new sap.ui.core.ComponentContainer({
        name: 'gbi'
      })
    }).placeAt('uiArea');
```

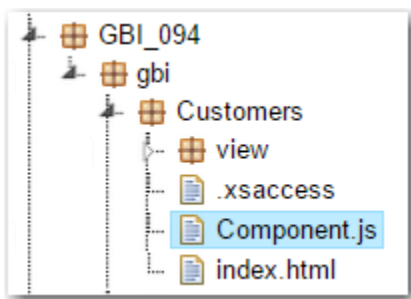
```
</script>

</head>
<body class="sapUiBody">
  <div id="uiArea"></div>
</body>
</html>
```

Listing 1

### Component.js

Create the Component.js file in the Customer package.



And add the following code.

```
jQuery.sap.declare("gbi.Component");

sap.ui.core.UIComponent.extend("gbi.Component",{

    metadata: {

        routing: {

            config: {
                viewType: "XML",
                viewPath: "gbi.view",
                targetControl: "splitApp",
                clearTarget: false,
                transition: "slide"
            },

            routes: [
                {
                    pattern: "",
                    name: "Customers",
                    view: "Master",
                    targetAggregation: "masterPages",
```

```
        subroutes : [
            {
                pattern : "Orders/{entity}",
                name : "Details",
                view : "Details",
                targetAggregation : "detailPages"
            }
        ]
    }
}

},

init: function() {

    jQuery.sap.require("sap.m.routing.RouteMatchedHandler");
    jQuery.sap.require("sap.ui.core.routing.HashChanger");

    //call createContent
    sap.ui.core.UIComponent.prototype.init.apply(this, arguments);

    this._router = this.getRouter();

    //initialize the router
    this._routeHandler = new sap.m.routing.RouteMatchedHandler(this._router);
    this._router.initialize();

},

createContent: function() {

    var oView = sap.ui.view({
        id: "app",
        viewName: "gbi.view.App",
        type: "JS",
        viewData: {component: this}
    });

    var oModel = new
sap.ui.model.odata.ODataModel("http://hana.ucc.uwm.edu:8004/GBI_002/gbi/services/gbi.xsodat
a");

    oView.setModel(oModel,'gbi');

    return oView;

}
```

```
});
```

*Listing 2***Update the highlighted portion of the code to reflect your oData service URL.**

This code declares a component called `gbi.Component` and then extends it. Note the `gbi` at the beginning of the component name corresponds to `./` as declared in the `index.html` file. This means that SAPUI5 will look for the file in the same package as the `index.html` file.

```
jQuery.sap.declare("gbi.Component");  
  
sap.ui.core.UIComponent.extend("gbi.Component", {
```

Next, metadata for the component is defined. In this case the only metadata is configuration data for a router. The router object will be used to navigate between views. The `config` section defines default values for the router. We will use views defined using XML and locate them in the view package. The views will be displayed in a `splitApp` control which is used to organize a master/detail application. The `clearTarget` attribute indicates that the application shouldn't delete the contents of the target control before navigation and the `transition` defines the type of transition to use when navigation occurs. You can find the configuration options [here](#).

```
metadata: {  
  routing: {  
    config: {  
      viewType: "XML",  
      viewPath: "gbi.view",  
      targetControl: "splitApp",  
      clearTarget: false,  
      transition: "slide"  
    },  
    routes: [  
      {  
        pattern : "",  
        name : "Customers",  
        view : "Master",  
        targetAggregation : "masterPages",  
        subroutes : [  
          {  
            pattern : "Orders/{entity}",  
            name : "Details",  
            view : "Details",  
            targetAggregation : "detailPages"  
          }  
        ]  
      }  
    ]  
  }  
}
```

A SplitApp control has two aggregations: masterPages and detailPages. An aggregation is a collection to which multiple items can be bound. These collections can be bound to the views which make up the visual interface. The masterPages aggregation is shown on the left side of the application and the detailPages aggregation is shown on the right side of the screen. Each aggregation can have multiple views assigned to it and the routes define when each view is shown.

The first route in the image above has an empty pattern which causes this route to load automatically when the application loads. The name of the route is Customers and the view that will be loaded is Master.xml.view which will be loaded into the masterPages (left) aggregation.

The second route, a subroute of the Customers route is triggered with the pattern Orders/{entity}. The value {entity} will contain the id of the selected Customer when the route is invoked. The route loads the Details.view.xml view into the detailPages aggregation.

The init function is called when the Component is first created. The first two lines in this function tell SAPUI5 to load the two libraries that implement the router. Next, the constructor for the prototype of the Component class is called. The result is to initialize the component.

Finally, the router is created then initialized. The operation of the router will become clearer a little later.

```
init: function() {  
  
    jQuery.sap.require("sap.m.routing.RouteMatchedHandler");  
    jQuery.sap.require("sap.ui.core.routing.HashChanger");  
  
    //call createContent  
    sap.ui.core.UIComponent.prototype.init.apply(this, arguments);  
  
    this._router = this.getRouter();  
  
    //initialize the router  
    this._routeHandler = new sap.m.routing.RouteMatchedHandler(this._router);  
    this._router.initialize();  
  
},
```

The createContent function creates the content of the component which consists of a single view called App. The App view encapsulates the other views in the application. It then creates the application model using the URL of the oData service document.

```
createContent: function() {  
  
    var oview = sap.ui.view({  
        id: "app",  
        viewName: "gbi.view.App",  
        type: "JS",  
        viewData: {component: this}  
    });  
  
    var oModel = new sap.ui.model.odata.ODataModel("http://  
    oview.setModel(oModel, 'gbi');  
  
    return oview;  
  
}
```



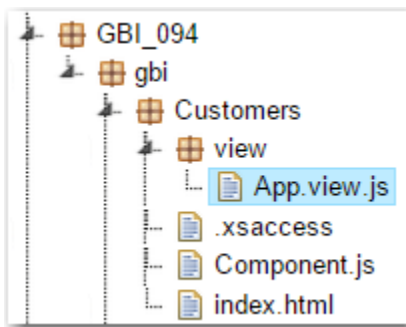
### App.view.js

Create the App.view.js file in the view package and add the following code.

```
sap.ui.jsview("gbi.view.App", {  
  
    createContent : function() {  
  
        this.setDisplayBlock(true);  
  
        return new sap.m.SplitApp("splitApp",{  
        }  
    }  
});
```

Listing 3

This code creates the application object based on the SplitApp master/detail structure.



### Master.view.xml

Create a file called Mater.view.xml in the view package and add the following code.

```
<mvc:View controllerName="gbi.view.Master" xmlns:mvc="sap.ui.core.mvc"  
    xmlns="sap.m">  
    <Page title="Customers">  
    <List  
        id="ShortCustomerList"  
        headerText="Customers"  
        items="{gbi>/Customers}" >  
        <StandardListItem  
            type="Active"  
            press="handleListItemPress"  
            title="{gbi>ID.CustomerID}"  
            description="{gbi>CompanyName}"    />  
        </List>  
    </Page>  
</mvc:View>
```

Listing 4

This code implements the initial view visible in the left side (master) of the screen. It consists of a List control that is bound to the Customers collection. The list items show the CustomerID and CompanyName fields. Setting the type attribute to Active makes the list items clickable and the press attribute assigns a function called handleListItemPress to handle the click event. This function is defined in the Master.controller.js file.

#### Master.controller.js

Create a file called Master.controller.js in the view package and insert the following code.

```
sap.ui.controller("gbi.view.Master", {

    onInit: function() {

        this.router = sap.ui.core.UIComponent.getRouterFor(this);
    },

    handleListItemPress: function(oItem){

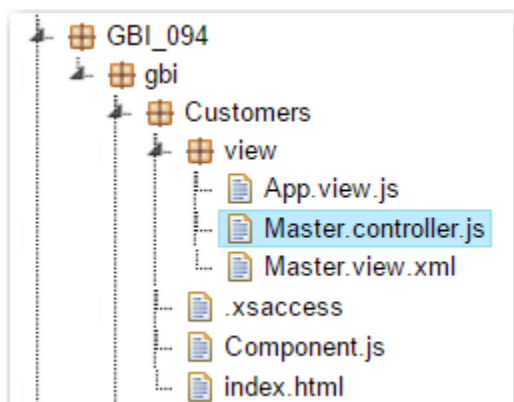
        var entity = oItem.getBindingContext("gbi").getPath().split("");

        this.router.navTo("Details", {
            from: "Master",
            entity: entity[1]
        });

    }

});
```

Listing 5



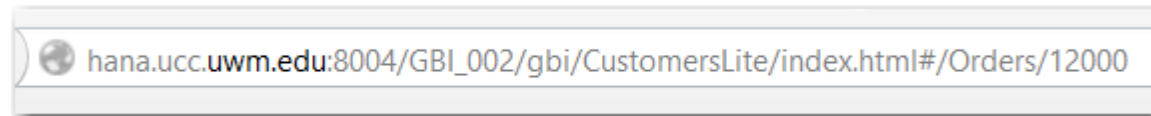
In the onInit function, a reference to the application's router object is retrieved.

The `handleListItemPress` event is invoked when a list item is clicked. The `oItem` argument refers to the list item that was clicked. The argument is used to retrieve the binding context of the clicked list item. One of the properties of the binding context is the path within the oData collection to the object. This path will be of the form `"/Customers('1000')"`. The `getPath` function retrieves this path and the `split` function breaks the path into parts delimited by the `'` character. The result is that entity is an array with three elements and the second element will be 1000.

The `navTo` method of the router object is used to navigate to the route that has the name `Details` and 1000 is passed as the parameter named `entity`. This matches the route (from the `Component.js` file):

```
pattern : "Orders/{entity}",
name : "Details",
view : "Details",
targetAggregation : "detailPages"
```

The route loads the `Details` view into the `detailPages` collection. If you look at the URL when this route is invoked you will see the pattern indicated by the route.



The 12000 is the id of the customer whose list item was clicked.

#### [Details.view.xml](#)

Create a file called `Details.view.xml` in the view package and insert the following code.

```
<mvc:View
    controllerName="gbi.view.Details"
    xmlns:l="sap.ui.layout"
    xmlns:core="sap.ui.core"
    xmlns:mvc="sap.ui.core.mvc"
    xmlns:f="sap.ui.layout.form"
    xmlns="sap.m">
    <Page title="Customer Details" >
        <l:Grid
            defaultSpan="L12 M12 S12"
            width="auto">
            <l:content>
                <f:SimpleForm id="idCusotmerForm"
                    minWidth="1024"
                    maxContainerCols="2"
                    editable="false"
                    layout="ResponsiveGridLayout"
                    title="Customer Details"
```

```

labelSpanL="4"
labelSpanM="4"
emptySpanL="0"
emptySpanM="0"
columnsL="2"
columnsM="2">
<f:content>

    <core:Title text="Customer" />
    <Label text="Number" />
    <Text text="{gbi>ID.CustomerID}" />
    <Label text="Name" />
    <Text text="{gbi>CompanyName}" />
    <Label text="Sales Org" />
    <Text text="{gbi>SalesOrgID}" />
    <core:Title text="Address" />
    <Label text="Address" />
    <Text text="{gbi>Address.Address}" />
    <Label text="City" />
    <Text text="{gbi>Address.City}" />
    <Label text="Region" />
    <Text text="{gbi>Address.Region}" />
    <Label text="Postal Code" />
    <Text text="{gbi>Address.Postal_code}" />
    <Label text="Country" />
    <Text text="{gbi>Address.Country}" />

</f:content>
</f:SimpleForm>
</l:content>
</l:Grid>

<Table id="idOrdersTable"
    inset="false"
    itemPress = "handleTableRowPress"
    items='{gbi>Orders}'>
    <columns>
        <Column>
            <header>
                <Text text="ID" />
            </header>
        </Column>
        <Column>
            <header>
                <Text text="Created At" />
            </header>
        </Column>
        <Column>
            <header>

```

```
<Text text="Amount" />
</header>
</Column>
<Column>
    <header>
        <Text text="Currency" />
    </header>
</Column>

</columns>
<items>
    <ColumnListItem>

        <Text text="{gbi>ID.OrderID}" />
        <Text text="{gbi>CreatedAt}" />
        <Text text="{gbi>GrossAmount.Amount}" />
        <Text text="{gbi>GrossAmount.Currency}" />

    </ColumnListItem>
</items>
</Table>

</Page>
</mvc:View>
```

Listing 6

This view consists of a SimpleForm control at the top that shows the customer details. The form is enclosed within a Grid control that controls the form's width. Below the form is a table that shows the customer's orders.

#### [Details.controller.js](#)

Create a file called Details.controller.js in the view package and insert the following code.

```
sap.ui.controller("gbi.view.Details", {
    onInit: function() {

        this.router = sap.ui.core.UIComponent.getRouterFor(this);
        this.router.attachRoutePatternMatched(this.onRouteMatched, this);

    },

    onRouteMatched : function(oEvent) {
        var oParameters = oEvent.getParameters();

        var oView = this.getView();
```

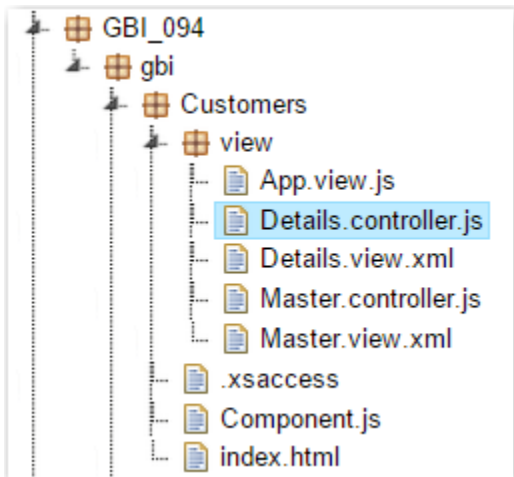
```
        var sEntityPath = "/" + oParameters.arguments.entity + "";
        var oModel = oView.getModel('gbi');
        var context = new sap.ui.model.Context(oModel , sEntityPath);

        oView.setBindingContext(context,'gbi');

    }

});
```

Listing 7



In the onInit function, a reference to the application's router is retrieved and then the onRouteMatched function is assigned to run when the router routes to this view.

In the onRouteMatched function the parameters of the route matched event are retrieved. The parameters include the entity value (which contains the customer id) that was passed to the route by the Master.controller.js code. This value is used to create the variable sEntityPath which will contain a value like /Customers('12000'). You should remember from the oData case that this is what is added to the end of the service document URL to retrieve the data from customer with ID equal to 12000. Next, a reference to the gbi model is retrieved and the model and sEntityPath are used to create a binding context that references the customer. The binding context is bound to the Details view. This makes the customer data available to the SimpleForm control.

Getting the customer's orders in the table makes use of the association between the Customers service and the SalesOrders service. The image below shows the definition of the Customers service from the gbi.xsodata file.

```
"GBI_094.gbi.data::GBI_094.MASTERDATA.CUSTOMERS" as "Customers"
    navigates ("CustomerOrders" as "Orders")
```

Notice that, in order to retrieve a customer's orders, you would use something like this:

```
<service document URL>/Customers('12000')/Orders
```

We just saw how the code in `onRouteMatched` bound the view to the customer (e.g. `/Customers('12000')`). If you look at the binding for items in the Table control in the `Details.view.xml` file you will see that it is bound to `Orders`. Since the Table control is embedded in the view, the combined bindings mean the table items are bound to something like `/Customers('12000')/Orders`.

## Run the Application

Run the application by clicking the `index.html` file and clicking the run icon. When the application loads initially you will see a list of customers in the master page section. When you click a customer, the customer's details and orders are shown.

Customers	Customer Details			
Customers	Customer Details			
1000 Rocky Mountain Bikes	Customer		Address	
10000 Silicon Valley Bikes	Number: 12000		Address: 601 108th Ave	
	Name: Northwest Bikes		City: Seattle	
11000 DC Bikes	Sales Org: UW00		Region: WA	
			Postal Code: 98004	
			Country: US	
12000 Northwest Bikes	ID	Created At	Amount	Currency
13000 Airport Bikes	101347	Mon Dec 01 2014	\$4,060.00	USD
14000 Alster Cycling	101606	Sun Apr 20 2014	\$102,352.79	USD
15000 Bavaria Bikes				
16000 Capital Bikes				
17000 Cruiser Bikes				
18000 Drahtesel				

## Exercise

Create the master/detail application using Sales Orders and Sales Order Details depicted in the image below.

Orders	Order Details																																										
Orders	Order Details																																										
100752 8000	<div><div>Number: 101317 Created By: Sales User Created At: Mon Nov 10 2014 18:00:00 GMT-0600 (Central Daylight Time)</div><div>Customer ID: 15000 Gross Amount: 12686.56 EUR Discount: 1268.66 Status: Processed</div></div>																																										
100753 13000																																											
100796 3000																																											
101180 10000	<table><tr><th>Item</th><th>Product</th><th>Quantity</th><th>UnitOfMeasure</th><th>Revenue</th><th>Discount</th></tr><tr><td>10</td><td>DXTR1100</td><td>1</td><td>ST</td><td>2487.56</td><td>124.38</td></tr><tr><td>110</td><td>RAAL1120</td><td>1</td><td>ST</td><td>1368.16</td><td>68.41</td></tr><tr><td>130</td><td>RACA1120</td><td>1</td><td>ST</td><td>3482.58</td><td>174.13</td></tr><tr><td>170</td><td>RHMT1000</td><td>1</td><td>ST</td><td>41.46</td><td>2.07</td></tr><tr><td>30</td><td>PRTR1100</td><td>1</td><td>ST</td><td>2653.4</td><td>132.67</td></tr><tr><td>50</td><td>PRTR3100</td><td>1</td><td>ST</td><td>2653.4</td><td>132.67</td></tr></table>	Item	Product	Quantity	UnitOfMeasure	Revenue	Discount	10	DXTR1100	1	ST	2487.56	124.38	110	RAAL1120	1	ST	1368.16	68.41	130	RACA1120	1	ST	3482.58	174.13	170	RHMT1000	1	ST	41.46	2.07	30	PRTR1100	1	ST	2653.4	132.67	50	PRTR3100	1	ST	2653.4	132.67
Item	Product	Quantity	UnitOfMeasure	Revenue	Discount																																						
10	DXTR1100	1	ST	2487.56	124.38																																						
110	RAAL1120	1	ST	1368.16	68.41																																						
130	RACA1120	1	ST	3482.58	174.13																																						
170	RHMT1000	1	ST	41.46	2.07																																						
30	PRTR1100	1	ST	2653.4	132.67																																						
50	PRTR3100	1	ST	2653.4	132.67																																						
101317 15000																																											
101347 12000																																											
101433 11000																																											
101434 16000																																											
101477 16000																																											
101478 13000																																											
101606 12000																																											