HD3C15 – End-to-End Customers Application

|  |  |
| --- | --- |
| **Product and Focus**  HANA Platform/oData | **MOTIVATION**  This case describes the user of oData services to create a simple SAPUI5 application.  **PREREQUISITES**  HD3C05 – Create the Persistence Model |
| **Target Audience**  Undergraduate/Graduate Beginner to Intermediate |
| **Author**  Ross Hightower |
| https://bgoerke.files.wordpress.com/2013/05/section1.png | |

# Core Data Services

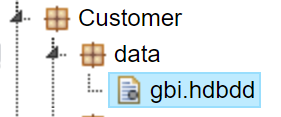
This case uses the core data services (CDS) infrastructure to create a persistence model that will be used to create services and an application. CDS is a semantically rich layer above SQL. CDS artifacts are design time objects that HANA uses to create the persistence objects in the HANA repository. The way that CDS works is by interpreting the descriptions of database objects that you create and then creating the SQL DDL statements to create the objects in the HANA database.

There are advantages to using CDS rather than using SQL directly, the two most important are that it’s easy to transport the CDS artifacts to other systems and allowing HANA to create the SQL allows it to optimize the SQL statements.

The persistence model created in this exercise will consist of three database tables. This case will describe the creation process of some of those entities but will leave others for you to complete.

## Create the Database

Create the structure shown below.

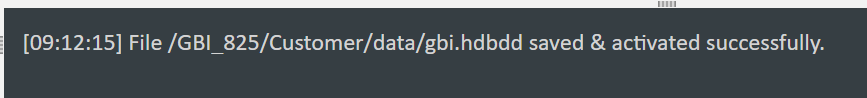


Copy the code shown below into the gbi.hdbdd file. Change the highlighted part to your id. The namespace at the top is the path through the packages to the gbi.hdbdd file. Note also that the gbi in the line context gbi refers to the name of the .hdbdd file.

|  |
| --- |
| namespace **GBI\_825**.Customer.data;  @Schema: **'GBI\_825'**  context gbi {    }; |

Listing 1

When you save the file you can see the result in the console below the editor. Errors will appear in red.



### Create the Customers table

Copy the highlighted code in the dbi.hdbdd file as shown below. Make sure you insert it between the opening and closing brackets for the context.

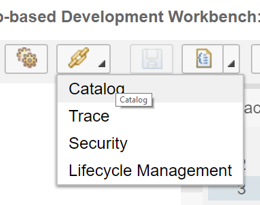
|  |
| --- |
| namespace GBI\_825.Customer.data;  @Schema: 'GBI\_825'  context gbi {    **@Catalog.tableType: #COLUMN**  **entity CUSTOMERS {**  **key ID : String(10);**  **CompanyName : String(35) not null;**  **Address : String(35);**  **City : String(20);**  **Region : String(2);**  **Country : String(2);**  **Postal\_code : String(5);**  **SalesOrgID : String(4);**  **};**  }; |

Listing 2

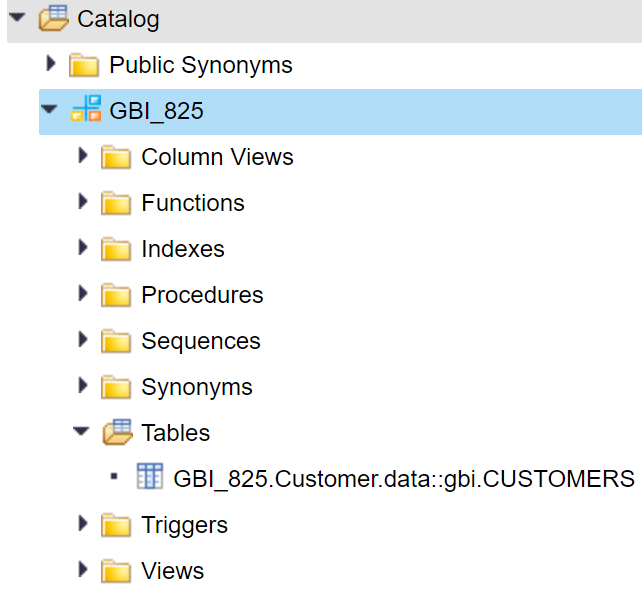
This code creates a column oriented table with nine fields. The primary key is ID.

Save the file and make sure there are no errors.

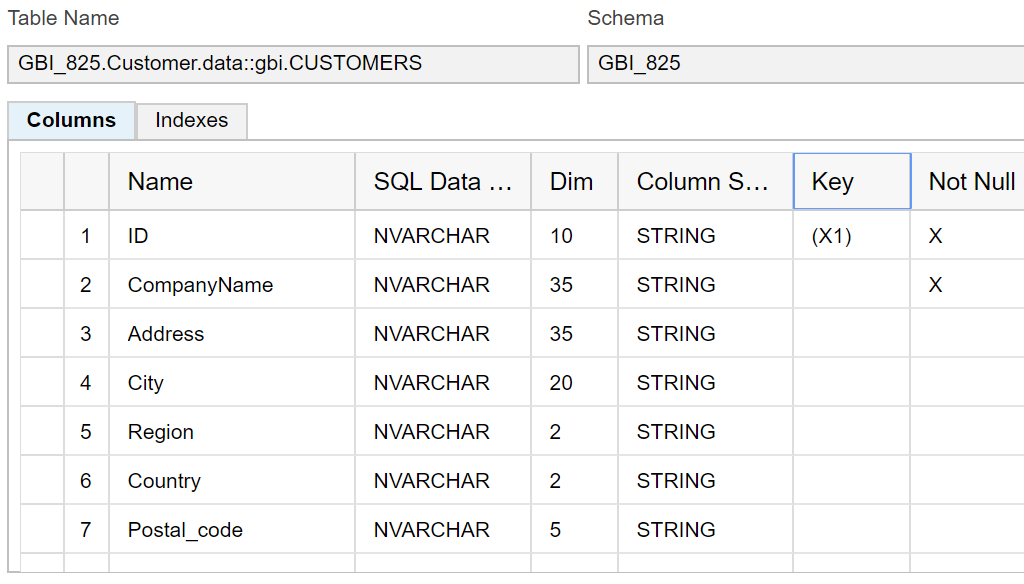
Open the Catalog editor using the icon shown below.



Drill into your database schema (it has the same name as your id) and open the tables folder.



Note the full name of the Customer table is the namespace defined at the top of the .hdbdd file then two colons followed by the context (gbi) and the name of the table. You can open the file structure by double-clicking the table.



Note the Key column indicates the primary key and the x in the Not Null column for the CompanyName field is there because that condition is specified in the table definition code.

You can see the data in the table by clicking the Open Content button. There is no data now but there will be later.

### Create the Sales Order Table

Use the following code to create the Sales Order table. Place this code immediately below the Customer table code.

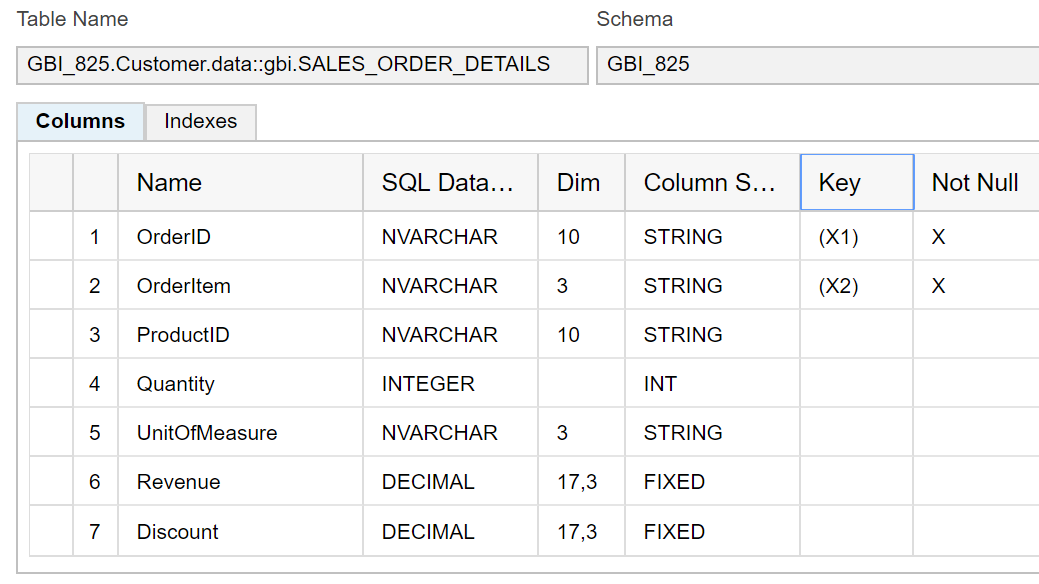
|  |
| --- |
| @Catalog.tableType: #COLUMN  entity SALES\_ORDERS {  CreatedAt : LocalDate;  CreatedBy : String(20);  CustomerID : String(10) not null;  key ID : String(10);  GrossAmount : Decimal(17,3);  Discount : Decimal(17,3);  Status : String(15) default 'New';  requiredDate : LocalDate;  shipDate : LocalDate;  }; |

Listing 3

Save the file and make sure there are no errors. If you look in the Catalog editor you will have to right-click your schema and select refresh to see the new table.

## Exercise 1

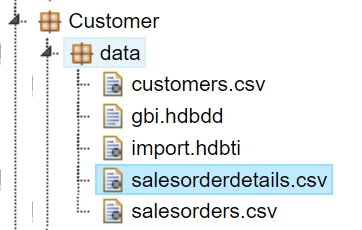
Now create the Sales Order Details table using the structure shown below.



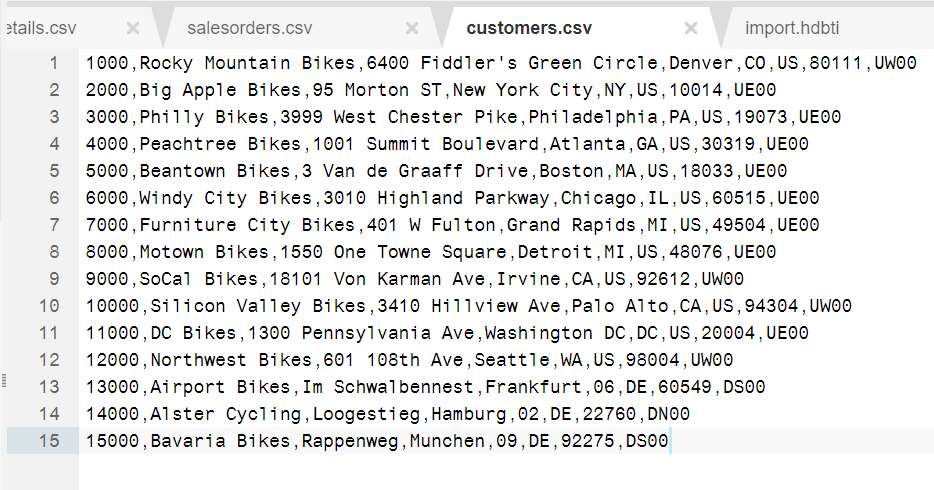
To get the INT data type use Integer in your table definition.

## Import Data into the Tables

Create the following files in the data package.



You can find the data to paste into the .csv files on D2L.



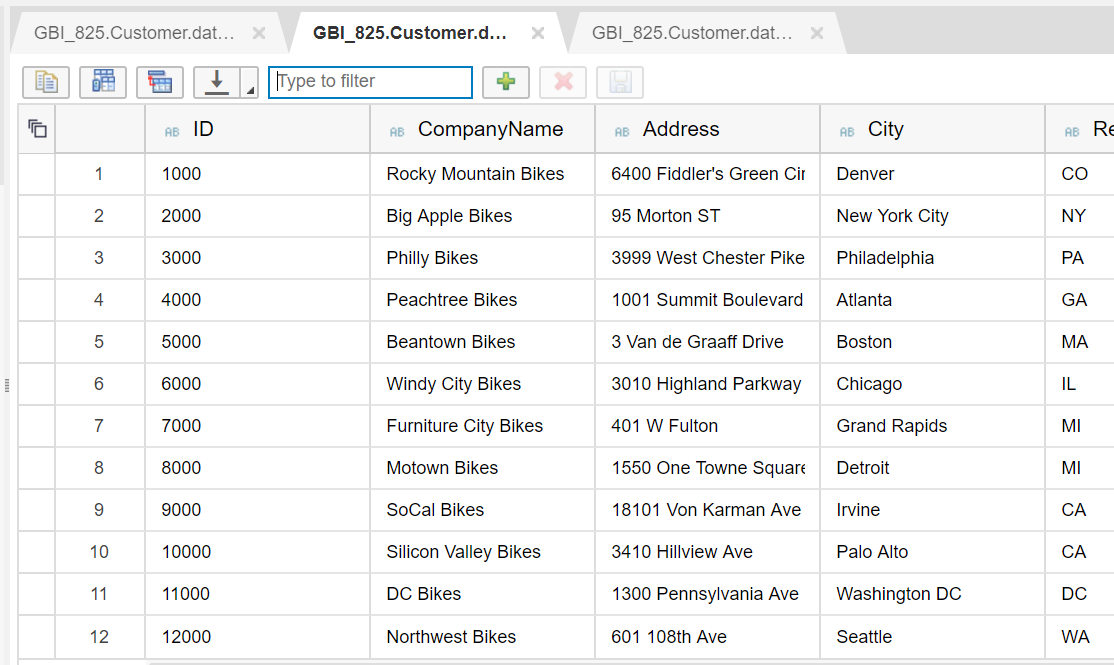
Paste the following code into the .hdbti file. Update the highlighted parts for your situation.

|  |
| --- |
| import = [  {  table = "**GBI\_825.Customer.data::gbi.CUSTOMERS**";  schema = "**GBI\_825**";  file = "**GBI\_825.Customer.data:customers.csv**";  header = false;  }  ]; |

Listing 4

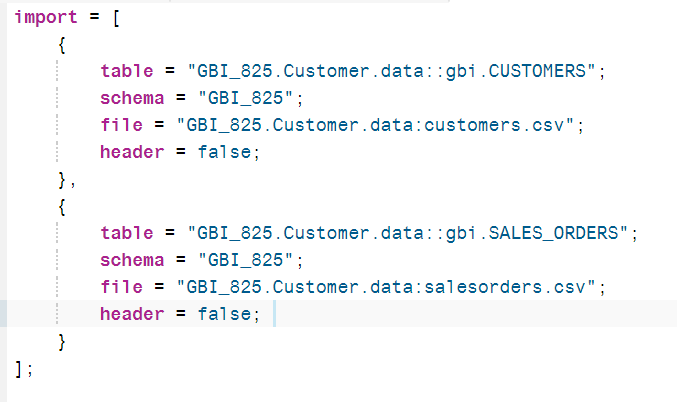
Save the file and make sure there are no errors in the console.

If you click Open Content for the Customer table in the Catalog editor you should have data.



## Exercise 2

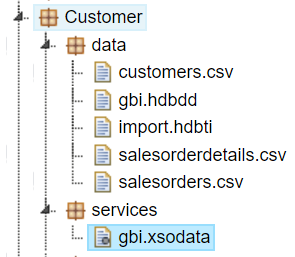
Now add the code to import data into the Sales Order and Sales Order Details tables. You can add the code to the .hdbti file as shown below separating each table’s code with a comma.



## Create oData Services

OData has been called ODBC of the Web because it is becoming a standard, cross-platform way to access and update data. It relies on web technologies such as RESTful services, Uniform Resource Identifiers (URI) for resource location, HTTP verbs (GET, PUT, POST and DELETE) for operations and JavaScript Object Notation (JSON) for data representation. OData is quickly becoming a widely used protocol because it is fast, efficient and flexible. Not only is the data easy to consume using oData, creating oData services on HANA is also very simple. A single line of code provides the ability to read, create, update and delete table. You can learn more about oData [here](http://odata.org) and [here](http://help.sap.com/saphelp_hanaplatform/helpdata/en/b8/0f8b626b3d44f882e8f2c3ff45952d/content.htm?frameset=/en/8d/78eb6f6e8f4307be939a6bc684b9ca/frameset.htm&current_toc=/en/34/29fc63a1de4cd6876ea211dc86ee54/plain.htm&node_id=244&show_children=false).

Create the package and file shown below.

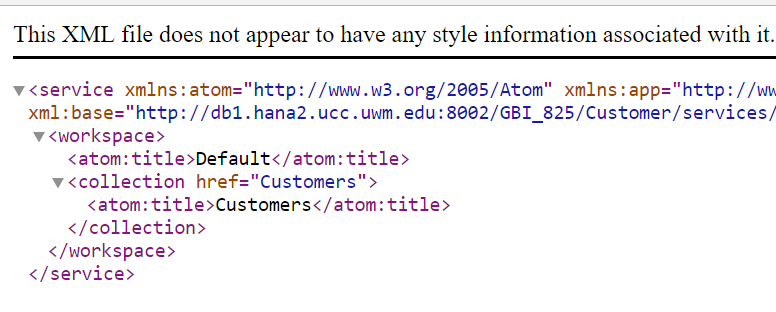


Paste the following code into the gbi.xsodata file. The highlighted part is the name of the Customer table. Update the code so it uses your table name.

|  |
| --- |
| service {  "**GBI\_825.Customer.data::gbi.CUSTOMERS**" as "Customers";  } |

Listing 5

This code creates an oData service based on the Customer table. The endpoint (URI) for the service is Customers. You can use this service to get the customer data, create new records, update existing records and delete records. To see it work, click Run.

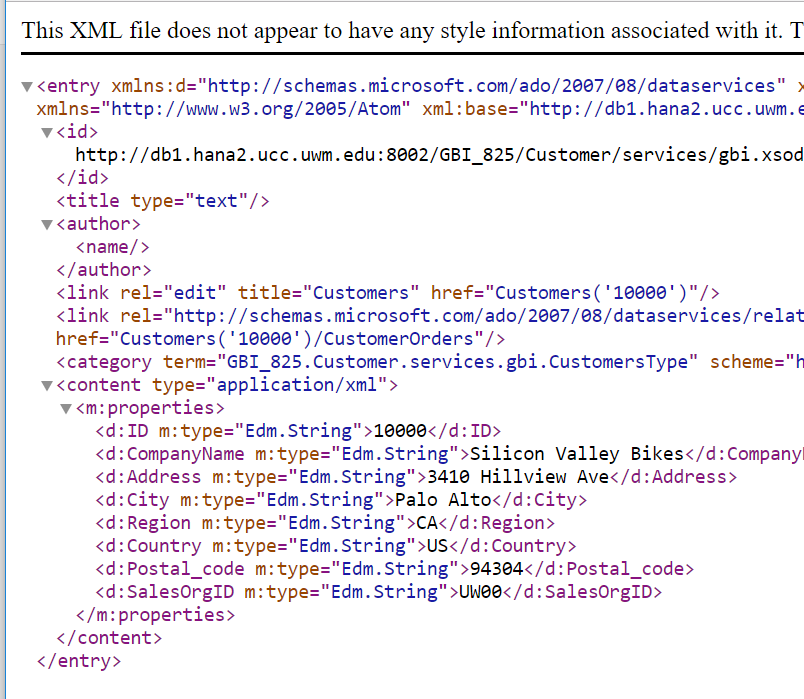


This is the output from the service document. Note it lists one service available called Customers. To retrieve the Customers data, add /Customers to the end of the url.

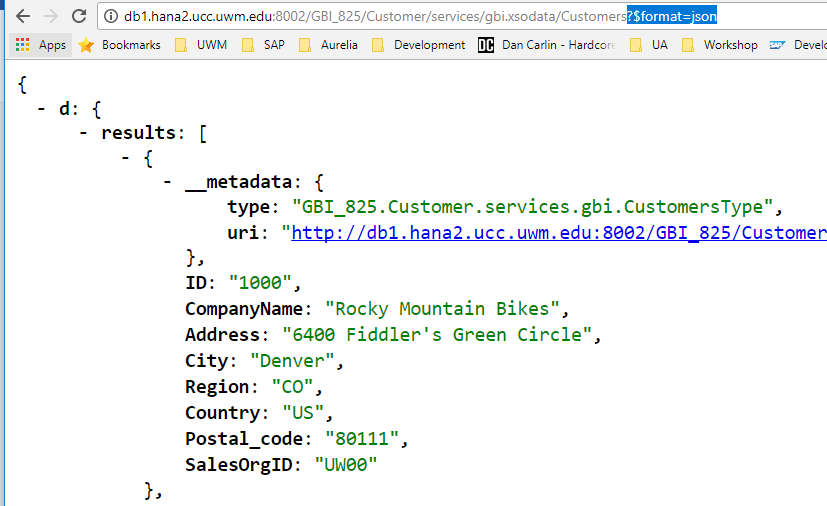


You can retrieve an individual customer by adding the primary key inside parentheses like this:

<http://db1.hana2.ucc.uwm.edu:8002/GBI_825/Customer/services/gbi.xsodata/Customers('10000')>



To see the data in JSON format, add ?$format=json to the end of the URI.



## Exercise 3

Create services for the Sales Order and Sales Order Details tables. You can add them to the gbi.xsodata file below the service for the Customer table. When you are done, the service document will look like this (you can get the endpoints of the services from this image).



### Add Associations Between Services

Now we have to create associations between the tables based on the keys in the tables. To create an association between the ID field in Customers and the CustomerID field in Sales Orders add the highlighted code shown below:

|  |
| --- |
| "GBI\_825.Customer.data::gbi.CUSTOMERS" as "Customers"  **navigates ("CustomersSO" as "CustomerOrders");**    **association "CustomersSO" principal "Customers"("ID") multiplicity "1"**  **dependent "SalesOrders"("CustomerID") multiplicity "\*";** |

Listing 6

The code that starts with “association…” defines a one-to-many association between Customers and Sales Orders. CustomersSO is the name of the association. The code “Customers”(“ID”) specifies the Customers endpoint and the ID field within the Customers table. The code multiplicity “1” indicates that the Customers table is the one side of the one-to-many association. The code “SalesOrders”(“CustomerID”) specifes the SalesOrder endpoint as the dependent service and that the CustomerID is the foreign key in the Sales Order table that is linked to ID in the Customer table. The code multiplicity “\*” indicates the SalesOrder service is the many side of the relationship.

The code that starts “navigates …” that is added to the Customers service assigns the association to the service using the association name (CustomersSO) and assigns the endpoint CustomerOrders to the association.

To use the association, first retrieve a specific customer like this:

<http://db1.hana2.ucc.uwm.edu:8002/GBI_825/Customer/services/gbi.xsodata/Customers('10000')>

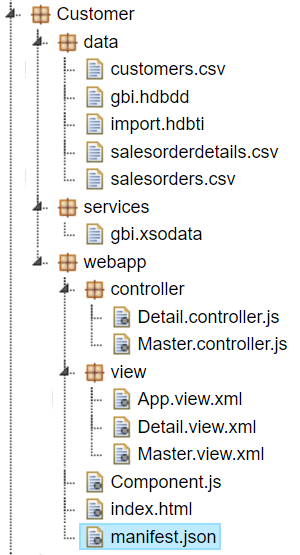
Then add the endpoint for the association by adding /CustomerOrders to the end of the URI.

## Exercise 4

Now create an association between SalesOrders and SalesOrderDetails. The primary key in Sales Orders is ID and the foreign key in Sales Order Details is OrderID. The endpoint for the association should be Details.

# Create the Application

Add the following structure in the Customers package.



### 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>Customers</title>    <!-- 1.) Load SAPUI5 (from local server), select theme and control library -->  <script id="sap-ui-bootstrap"  type="text/javascript"  src="https://openui5.hana.ondemand.com/1.42.6/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='{ "ui5": "./" }'></script>    <script>  sap.ui.getCore().attachInit(function() {  new sap.m.Shell("shell",{  app : new sap.ui.core.ComponentContainer({  height : "100%",  name : "ui5"  })  }).placeAt("uiArea"); });  </script>    </head>  <body class="sapUiBody">    <!-- This is where you place the UI5 button -->  <div id="uiArea"></div>    </body>  </html> |

Listing 7

### Component.js

Create the Component.js file in the Customer package and add the following code.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/UIComponent"  ], function (UIComponent) {  "use strict";  return UIComponent.extend("ui5.Component", {  metadata : {  manifest: "json"  },  init : function () {  // call the init function of the parent  UIComponent.prototype.init.apply(this, arguments);    this.getRouter().initialize();  }  });  }); |

Listing 8

### manifest.json

Add the following code the manifest.json file.

|  |
| --- |
| {  "\_version": "1.1.0",  "sap.app": {  "\_version": "1.1.0",  "id": "ui5",  "type": "application",  "i18n": "i18n/i18n.properties",  "title": "{{appTitle}}",  "description": "{{appDescription}}",  "applicationVersion": {  "version": "1.0.0"  },  "ach": "CA-UI5-FST",  "dataSources": {  "employeeRemote": {  "uri": "/here/goes/your/serviceUrl/",  "type": "OData",  "settings": {  "odataVersion": "2.0",  "localUri" : "localService/metadata.xml"  }  }  }  },  "sap.ui": {  "\_version": "1.1.0",  "technology": "UI5",  "deviceTypes": {  "desktop": true,  "tablet": true,  "phone": true  },  "supportedThemes": ["sap\_belize"]  },  "sap.ui5": {  "\_version": "1.1.0",  "rootView": "ui5.view.App",  "dependencies": {  "minUI5Version": "1.30",  "libs": {  "sap.m": { }  }  },  "models": {    },  "routing": {  "config": {  "routerClass": "sap.m.routing.Router",  "viewType": "XML",  "viewPath": "ui5.view",  "controlId": "app"  },  "routes": [  {  "pattern": "",  "name": "Master",  "target": "Master"  },  {  "pattern": "Detail/{item}",  "name": "Detail",  "target": "Detail"  }  ],  "targets": {  "Master": {  "viewName": "Master",  "controlAggregation": "masterPages"  },  "Detail": {  "viewName": "Detail",  "controlAggregation": "detailPages"  }  }  }  }  } |

Listing 9

### App.view.xml

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

|  |
| --- |
| <mvc:View  xmlns="sap.m"  xmlns:mvc="sap.ui.core.mvc"  displayBlock="true">  <SplitApp class="myAppDemoWT" id="app"/>  </mvc:View> |

Listing 10

### Add the Model

Locate the models section in the manifest.json file and update it with the following code. Edit the service to use your service document URI.

|  |
| --- |
| "models": {  **"gbi": {**  **"type": "sap.ui.model.odata.ODataModel",**  **"uri": "http://db1.hana2.ucc.uwm.edu:8002/GBI\_825/Customer/services/gbi.xsodata"**  **}**  }, |

Listing 11

### Master.view.xml

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

|  |
| --- |
| <mvc:View controllerName="ui5.controller.Master" xmlns:mvc="sap.ui.core.mvc"  xmlns="sap.m">  <Page title="Customers">  <List  id="CustomerList"  headerText="Customers"  items="{gbi>/Customers}" >  <StandardListItem  type="Active"  press="handleListItemPress"  title="{gbi>ID}"  description="{gbi>CompanyName}" />  </List>  </Page>  </mvc:View> |

Listing 12

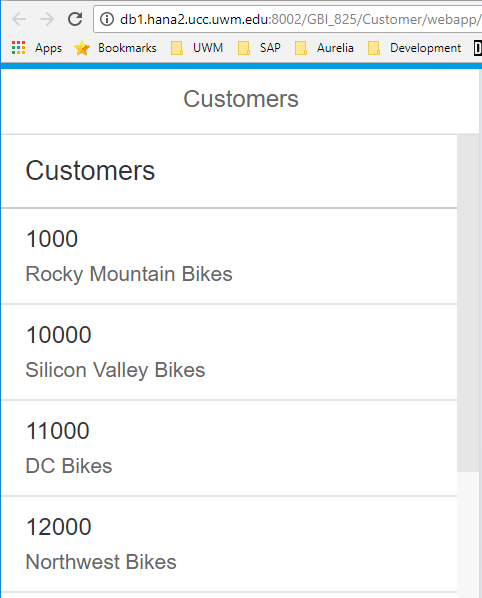
### Master.controller.js

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

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("ui5.controller.Master", {    });  }); |

Listing 13

If you run the app now it should load a list of customers on the master side of the app.



This list control specifies an event handler called handleListItemPress for the press event. To add this event handler, add the following code to the Master.controller.js file.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("ui5.controller.Master", {    **handleListItemPress: function(oEvent){**  **var oItem = oEvent.getSource();**  **var entityArray = oItem.getBindingContext("gbi").getPath().split("'");**  **var oRouter = sap.ui.core.UIComponent.getRouterFor(this);**  **oRouter.navTo("Detail", {**  **item: entityArray[1]**  **});**  **}**  });  }); |

Listing 14

This code retrieves the path of the clicked item. The path will be formatted like :

/Customers('10000')

If you’re paying attention, you will realize this is the string you add to the end of the OData service document URI to retrieve customer 10000.

http://db1.hana2.ucc.uwm.edu:8002/GBI\_825/Customer/services/gbi.xsodata/Customers('10000')

We need to extract the customer ID (10000) to pass as a parameter to the Detail route. We can use the JavaScript split array method to split the string on the ‘ character. This returns an array with 3 elements:

0: /Customers(

1: 10000

2: )

We pass element 1 to the route.

### Detail.view.xml

Add the following code to the Detail.view.xml file. This code adds a form that will show the customer details.

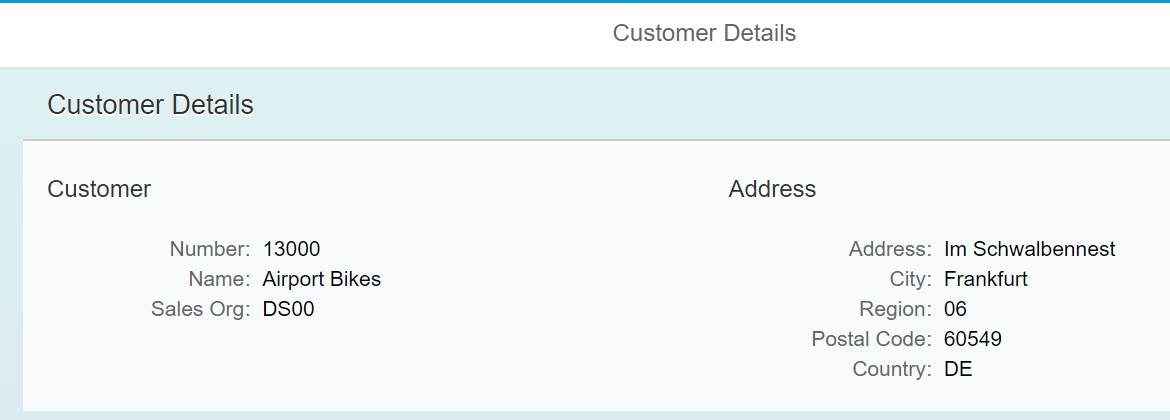
|  |
| --- |
| <mvc:View  controllerName="ui5.controller.Detail"  xmlns:l="sap.ui.layout"  xmlns:core="sap.ui.core"  xmlns:mvc="sap.ui.core.mvc"  xmlns:f="sap.ui.layout.form"  xmlns="sap.m">  <Page title="Customer Details" >  <l:Grid  defaultSpan="L12 M12 S12"  width="auto">  <l:content>  <f:SimpleForm 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}" />  <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}" />  <Label text="City" />  <Text text="{gbi>City}" />  <Label text="Region" />  <Text text="{gbi>Region}" />  <Label text="Postal Code" />  <Text text="{gbi>Postal\_code}" />  <Label text="Country" />  <Text text="{gbi>Country}" />  </f:content>  </f:SimpleForm>  </l:content>  </l:Grid>  </Page>  </mvc:View> |

Listing 15

### Detail.controller.js

Add the following code to Detail.controller.js. This code binds the correct path to view.

|  |
| --- |
| sap.ui.define([  "sap/ui/core/mvc/Controller"  ], function (Controller) {  "use strict";  return Controller.extend("ui5.controller.Detail", {    onInit: function () {  var oRouter = sap.ui.core.UIComponent.getRouterFor(this);  oRouter.getRoute("Detail").attachPatternMatched(this.\_onObjectMatched, this);  },    \_onObjectMatched: function (oEvent) {  this.getView().bindElement({  path: "/Customers('" + oEvent.getParameter("arguments").item + "')",  model: "gbi"  });  }  });  }); |

Listing 16

If you run the application, you can now click on a customer on the Master view and see the details on the top of the Detail view.

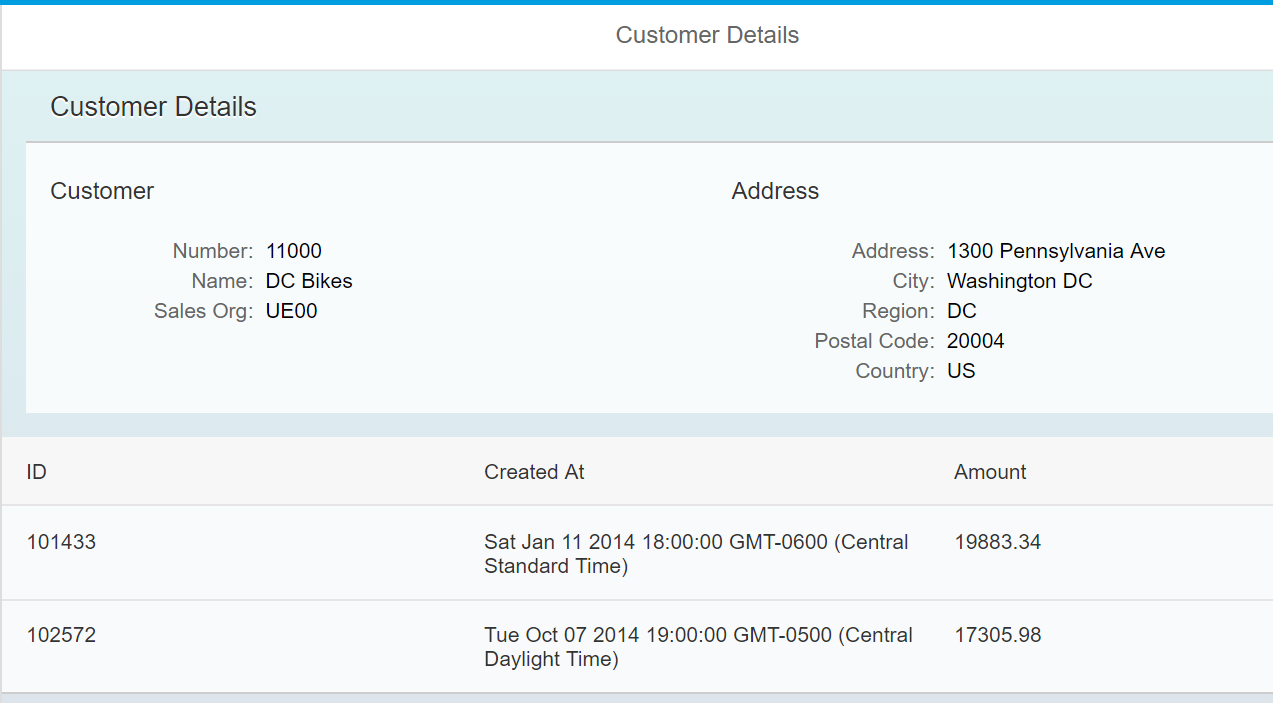
### Detail.view.xml

Now we can take advantage of the CustomerOrders association service to load the customer’s sales orders into a table below the details form. Add the highlighted code to the Details.view.xml file. Note the binding of the items attribute of the Table control. This is an aggregation binding. CustomerOrders is the endpoint for the association service between Customers and Sales Orders. Since the view is bound to the customer (something like /Customers(‘10000’)), the binding for the Table will be /Customers(‘10000’)/CustomerOrders.

|  |
| --- |
| </l:content>  </l:Grid>    **<Table id="idOrdersTable"**  **inset="false"**  **itemPress = "handleTableRowPress"**  **items='{gbi>CustomerOrders}'>**  **<columns>**  **<Column>**  **<header>**  **<Text text="ID" />**  **</header>**  **</Column>**  **<Column>**  **<header>**  **<Text text="Created At" />**  **</header>**  **</Column>**  **<Column>**  **<header>**  **<Text text="Amount" />**  **</header>**  **</Column>**  **</columns>**  **<items>**  **<ColumnListItem>**    **<Text text="{gbi>ID}" />**  **<Text text="{gbi>CreatedAt}" />**  **<Text text="{gbi>GrossAmount}" />**    **</ColumnListItem>**  **</items>**  **</Table>**  </Page>  </mvc:View> |

Listing 17

Now you will see the sales orders for the selected customer.



## Format Values

One last thing to do is to format the dates and amounts in the tables.

### Detail.controller.js

Add the following functions to the Detail.controller.js file.

|  |
| --- |
| dateFormatter: function(date){  return new Date(date).toLocaleDateString();  },  currencyFormatter : function(value){  var d = ".";  var t = ",";  var c = 2;  var p = "$";  c = isNaN(c = Math.abs(c)) ? 2 : c;  var s = value < 0 ? "-" : "";  var i = parseInt(value = Math.abs(+value || 0).toFixed(2)) + "";  var j = (j = i.length) > 3 ? j % 3 : 0;  return p + s + (j ? i.substr(0, j) + t : "") + i.substr(j).replace(/(\d{3})(?=\d)/g, "$1" + t) + (c ? d + Math.abs(value - i).toFixed(2).slice(2) : "");  } |

Listing 18

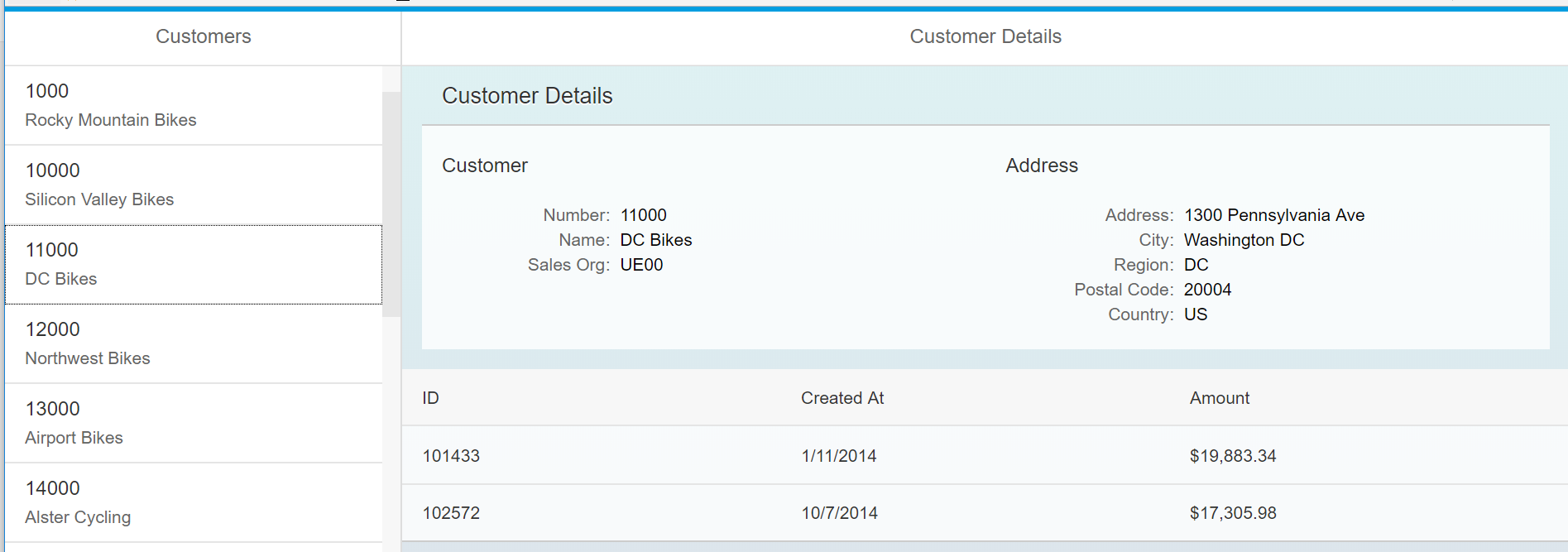
### Detail.view.xml

Next, we have to update the view XML so UI5 will know to call these formatter functions. Locate the two table columns and edited them as shown.

|  |
| --- |
| <ColumnListItem>  <Text text="{gbi>ID}" />  **<Text text="{**  **path: 'gbi>CreatedAt',**  **formatter: '.dateFormatter'**  **}"**  **/>**  **<Text text="{**  **path: 'gbi>GrossAmount',**  **formatter: '.currencyFormatter'**  **}" />**  </ColumnListItem> |

Listing 19

Much better!



Your done!