Mobile Analytics Kit
Developer Guide: HTML5

SAP Mobile Platform 2.3

Document ID: DC01731-01-0230-01

Last Revised: January 2013

© 2013 SAP AG or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries. Please see http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark for additional trademark information and notices.

Contents

Introduction 7
Architecture 7
Creating a Chart Using MAKit 7
Creating a Static Web Project in Eclipse 7
Importing MAKit JavaScript and CSS Files to the Project 7
Adding a Metadata File 8
Adding an HTML File 8
Adding a Chart 8
Feeding Data via the Query Function 9
Viewing the Chart 10
Sorting the Chart Category 12
Refreshing the Chart 12
Creating a Semantic Zooming Chart 12
Adding a Metadata File 12
Feeding Data 13
Viewing the Zooming Chart 15
Creating a Drilldown Group 15
Adding a Metadata File 16
Feeding Data 16
Viewing the Drilldown Chart 18
Creating a What-If Chart 19
Adding a Metadata File 19
Viewing the What-If Chart 20
Showing the What-If View by JavaScript API 20
Creating a Dynamic Analytics Chart 20
Adding a Global DA Variables Metadata File 20
Adding a Metadata File 21
Loading Global DA Variables Metadata File and Binding DAChanged Event 21
Viewing the DA Chart 22
Showing the Dynamic Analytics View by JavaScript API 22
Advanced Features 22
Creating a Dynamic Chart 22
Creating a Localized Chart 23
Sharing a Chart 23
Creating a Multiple Categories Chart 23
Creating a Combination Chart 24

Using the “Lite” Version 25
Best Practices 26
Performance 26
Usability 26
Functions and Events 26
$MA 26
Functions 26
$MA.Chart 26
Properties 27
Functions 28
Events 30
$MA.DataTable 30
Functions 32
$MA.DrillDownGroup 32
Functions 32
Events 33
Tested Operating Systems and Browsers for MAKit 33
Expressions 33
Syntax 33
Expression Types 34
Runtime Functions 35
Numeric Functions 35
DateTime Functions 36
String Functions 36
Miscellaneous Functions 37
Metadata File 37
Chart 37
Category 39
Series 39
Values 39
Semantic Zooming 40
What-If Analysis 40
Variables 42
DrillDownGroup 44
Themes 45
Using a Static Theme 45
Using a Dynamic Theme 45
User-Defined Theme for Chart 45
Theme Definition 46
Page 46

Web Controls 46
DatePicker 46
Slider 47
DigitPicker 47
Chart 48
Slider 48
Range Selector 49
Title Bar 49
Toolbar 50
Breadcrumbs 51
Table 52
Graph 53
A – Outline 54
B – defaultStrokePalette 54
C – defaultInsetShadowPalette 55
D – defaultFillPalette 56
Using MAKit in SAP Mobile Platform Hybrid Web Container 57
Creating Database Tables Using SAP Mobile WorkSpace 57
Creating a Sample Connection Profile 57
Creating Sample Tables 58
Creating a New Mobile Application 59
Creating New MBOs 59
Creating a New MBO for the RevenueByY Table 59
Deploying the MBO 59
Creating a New Hybrid App 60
Adding an MBO to the Hybrid App 60
Adding an Action to Retrieve Data from the MBO 60
Deploying the Hybrid App 61
Defining an Application Connection 62
Adding a Simple MAKit Chart to the Hybrid App 66
Adding MAKit-Related Files to the Project 66
Adding the Metadata Files for the Chart 68
Adding a New HTML Screen Page 68
Creating a MAKit Chart 68
Getting Data from SAP Mobile Server 69
Deploying the Hybrid App 71
Creating a Semantic Zooming Chart 71
Creating Sample Data Tables 71
Creating MBOs for the Two Tables 71
Adding Metadata for the Semantic Zooming Chart 72

Adding New HTML Screen Pages 72
Creating the MAKit Semantic Chart 73
Getting Data from the SAP Mobile Server 74
Deploying the Hybrid App 81
Creating a Drilldown Chart 82
Creating Sample Data Tables 82
Creating MBOs for a Drilldown Chart 82
Adding Metadata for the Drilldown Chart 84
Adding New HTML Screen Pages 86
Creating the MAKit Drilldrown Chart 87
Getting Data from the SAP Mobile Server 88
Deploying the Hybrid App 101
System Requirements for the SAP Mobile Platform Web Container 101
Reference 102
SQL Statements for RevenueByY Table 102
SQL Statements for RevenueByQ Table 103
SQL Statements for RevenueByM Table 105
SQL Statements for EmpIncome Table 113

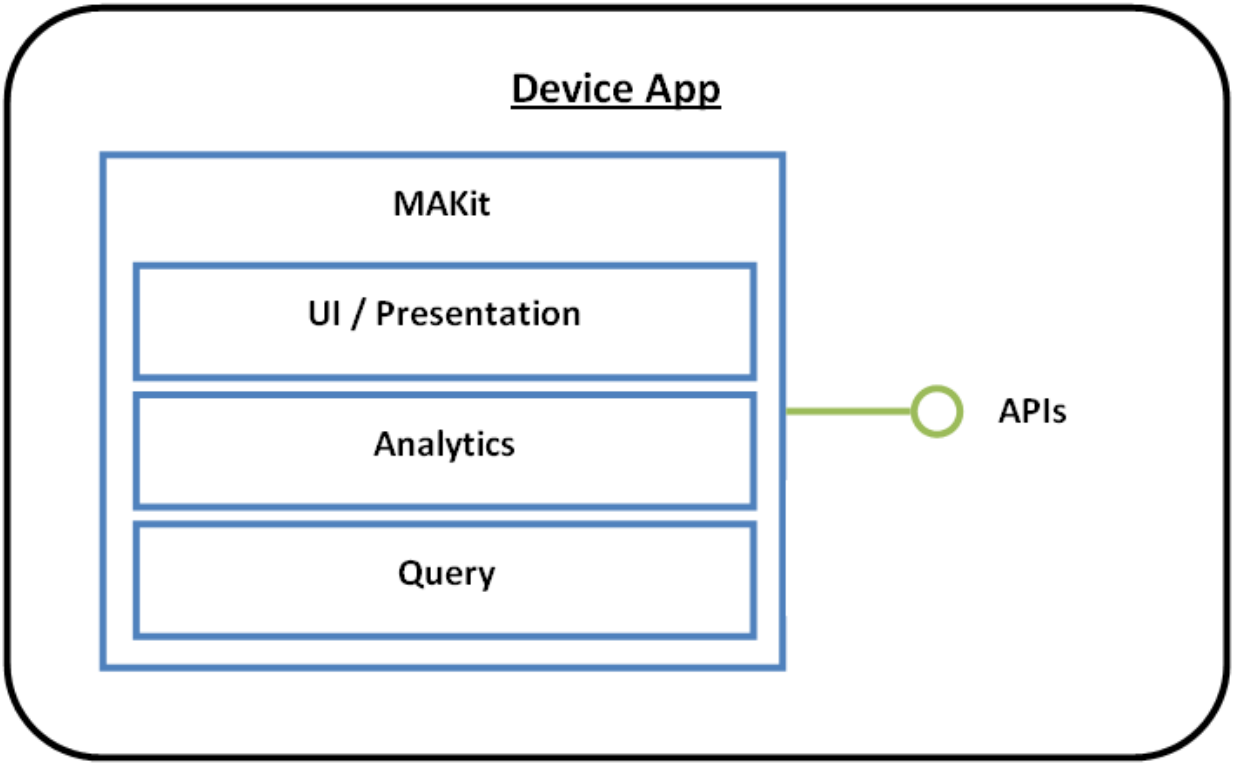
Introduction

The *MAKit Developer Guide: HTML* is to help developers quickly understand Mobile Analytics Kit (MAKit) concepts, and to expedite application development. MAKit, which lets users graphically view data and modify chart values to examine "what-if" scenarios, is distributed as JavaScript and css files.

MAKit provides a set of easy-to-use APIs and metadata that allows application developers to build analytical applications that visualize the data retrieved in the SAP® Mobile Platform Hybrid Web Container.

Architecture

The MAKit `UI/Presentation` module includes the UI with its views, `Analytics` processes on-device analytics, and `Query` provides a standard way to feed data into MAKit. The APIs offer flexibility and customization options.



Creating a Chart Using MAKit

Creating a Static Web Project in Eclipse

1. Select **File** > **New** > **Static Web Project**.
2. Specify a project name. Click **Finish**.

Importing MAKit JavaScript and CSS Files to the Project

1. Right-click the **WebContent** folder, select **New > Folder**, and add a new folder called `css`. Import the MAKit `css` files to the `css` folder.
2. Right-click the **WebContent** folder, select **New > Folder**, and add a new folder called `scripts`. Import the MAKit JavaScript files to the `scripts` folder.
3. Right-click the **WebContent** folder, select **New > Folder**, and add a new folder called `images`. Import the MAKit image files to the `scripts` folder.

Adding a Metadata File

1. Right-click the **WebContent** folder, select **New > Folder**, and add a new folder called `metadata`.
2. Right-click the **metadata** folder, select **New > File**, and add a new file called `mychart.xml`. Copy this chart metadata to the file:

```
<Chart Name="chart_1" ChartType="column" Query="revenue" Title="Revenue">

        <Category Column="year"/>

        <Series Column="product"/>

        <Values>

                <Value Expression="revenue" Format="currency"/>

        </Values>

</Chart>
```

Adding an HTML File

1. Right-click the **WebContent** folder, select **New > HTML file** and create an HTML file.
2. Include this meta element to specify initial scale and device width:

```
<meta name="viewport" content="initial-scale=1; maximum-scale=1; user-scalable=0; width=device-width;"/>
```

3. Include this MAKit css file:

```
<link rel="stylesheet" href="css/chart.css" type="text/css"/>
```

4. Include this MAKit JavaScript file:

```
<script src="scripts/sap.makit-2.3.min.js"></script>
```

Adding a Chart

1. Add an HTML DIV element to be used as a container for the chart. Specify the width and height of the DIV element in the code (as shown below), or during page initialization. The default width of a DIV element is 100%.

```
<div id="chart_1" style="height:500px;"></div>
```

2. Create a full-view chart by adding the code below. The chart must be created with an ID of HTML DIV element. The second argument is optional and specifies whether to create a full-view chart. By default, a chart includes only the Graph view, whereas a full-view chart includes Toolbar, Graph, Table, Range Selector, What-If, and Analysis views. By default, the What-if and Dynamic Analytic views are invisible, and can be customized on an initialized event. Use the **titlechanged** event to display a title on the navigation bar in a full-view chart.

```
<script type="text/javascript">

window.onload = function() {

        var chart = new $MA.Chart("chart_1", true);

        chart.bind("initialized", onInitialized);

        chart.bind("titlechanged", onTitleChanged);

        chart.load("metadata/mychart.xml");
```

```
}


function onInitialized(id) {

        // Write your code to show or hide chart components

}


function onTitleChanged(id, title) {

        // Write your code to show title on navigation bar

}

</script>
```

Feeding Data via the Query Function

1. The application must implement the **executeMAQuery** function to feed data to MAKit. This allows the application to retrieve data from any datasource and to preprocess data before sending it to MAKit. The source of data is transparent to MAKit. This sample uses the "revenue" query:

```
function executeMAQuery(chartObj, name, args)

{

        var dataTable;

        if (name == "revenue")

                dataTable = getRevenueData();

        chartObj.setDataTable(dataTable);

};
```

2. Create a data table in each query function, and within the data table, create all columns required by the chart. For this sample, the "year" column is required for category, the "product" column is required for series, and the "revenue" column is required for values.

```
function getRevenueData()

{

        var dataTable = new $MA.DataTable();

        dataTable.addColumn("year", "number");

        dataTable.addColumn("product", "string");

        dataTable.addColumn("revenue", "number");

        dataTable.addRow ([2009, "Product 1", 100]);

        dataTable.addRow([2009, "Product 2", 150]);

        dataTable.addRow ([2010, "Product 1", 120]);
```

```
        dataTable.addRow([2010, "Product 2", 80]);

        return dataTable;

}
```

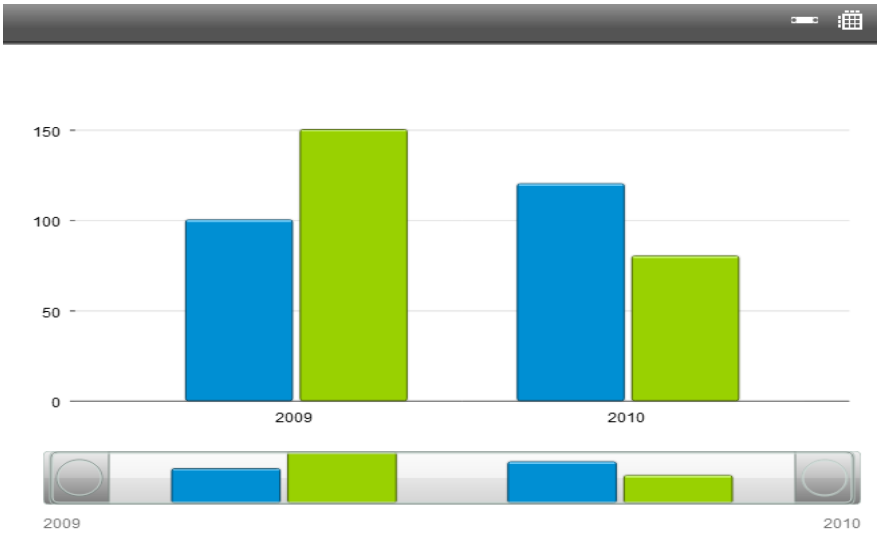
Viewing the Chart

This is the complete code for this basic chart:

```
<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="initial-scale=1; maximum-scale=1; user-scalable=0; width=device-
width;"/>

<title>Insert title here</title>

<link rel="stylesheet" href="css/chart.css" type="text/css"/>

<script src="scripts/sap.makit-2.3.min.js"></script>

</head>

<body>

<script type="text/javascript">

window.onload = function() {

        var chart = new $MA.Chart("chart_1", true);

        chart.bind("initialized", onInitialized);

        chart.bind("titlechanged", onTitleChanged);

        chart.load("metadata/mychart.xml");

}

function onInitialized(id) {

        // Write your code to show or hide chart components

}

function onTitleChanged(id, title) {

        // Write your code to show title on navigation bar

}

function executeMAQuery(chartObj, name, args)

{

        var dataTable;

        if (name == "revenue")
```

```
                dataTable = getRevenueData();

        chartObj.setDataTable(dataTable);

};

function getRevenueData()

{

        var dataTable = new $MA.DataTable();


        dataTable.addColumn("year", "number");

        dataTable.addColumn("product", "string");

        dataTable.addColumn("revenue", "number");

        dataTable.addRow ([2009, "Product 1", 100]);

        dataTable.addRow([2009, "Product 2", 150]);

        dataTable.addRow ([2010, "Product 1", 120]);

        dataTable.addRow([2010, "Product 2", 80]);

        return dataTable;

}


</script>

<div id="chart_1" style="height:500px;"></div>

</body>

</html>
```

Right-click your project name, and select **Run As > Run on Server.** Click **Finish.**



Sorting the Chart Category

Change the sort order by specifying the **category.sortorder** property in metadata, or calling the **setProperty** method. Supported sort orders include:

- Ascending (default) – sorts the category column and zooming columns in ascending order.
- Descending – sorts the category column and zooming columns in descending order.
- None – no sorting. Use this option only if the data is already sorted by category.

Refreshing the Chart

Add this code to refresh the chart when the tablet orientation is changed. The **OrientationChange** event is not yet supported by all browsers—you may be able to use the **resize** event instead, although it may be triggered several times when the orientation is changed. Use the $MA static constructor to find the chart with ID. Use the **orientation** event to change the chart layout and refresh the chart.

```
window.addEventListener(navigator.userAgent.indexOf("iPad") > 0 ? "orientationchange" : "resize",
function() {

        onOrientationChanged();

}, false);



function onOrientationChanged() {

        $MA("chart_1").refresh();

}
```

Creating a Semantic Zooming Chart

Semantic zooming is useful for aggregating data. A commonly used scenario is to aggregate data by year, quarter, and month. Another scenario is to aggregate data by region, country, state, and city. You can also use the chart table to view aggregate data.

Adding a Metadata File

This example defines a semantic zooming chart with three zooming levels: level 1 displays data by year; level 2 displays data by quarter, and level 3 displays data by month:

```
<Chart Name="chart_1" ChartType="column" Title="Revenue">

        <Category/>

        <Series Column="product"/>

        <Values>

                <Value Expression="revenue" Format="currency"/>

        </Values>

        <ZoomingLevels>

                <ZoomingLevel Query="productRevenueByYear" CategoryColumn="year" Title="Year"/>

                <ZoomingLevel Query="productRevenueByQuarter" CategoryColumn="quarter"
Title="Quarter"/>

               <ZoomingLevel Query="productRevenueByMonth" CategoryColumn="month" Title="Month"/>

        </ZoomingLevels>

</Chart>
```

Feeding Data

The data table returned by queries of the three zooming levels must reflect the zooming relationship. In this example, the level 1 query returns a table with these columns: “year”, “product”, ”revenue”. The level 2 query returns a table with these columns: “year”, “quarter”, “product”, “revenue”. The level 3 query returns a table with these columns: “year”, “quarter”, “month”, “product”, “revenue”.

This sample uses the “productRevenueByYear”, “productRevenueByQuarter”, and “productRevenueByMonth” queries:

```
function executeMAQuery(chartObj, name, args)

{

        var dataTable;

        if (name == "productRevenueByYear")

                dataTable = getProductRevenueByYearData();

        if (name == "productRevenueByQuarter")

                dataTable = getProductRevenueByQuarterData();

        if (name == "productRevenueByMonth")

                dataTable = getProductRevenueByMonthData();

        chartObj.setDataTable(dataTable);

};
```

This sample uses SQLite® test data. The year, product, and sum (revenue) are retrieved for the level 1 query. The data is retrieved asynchronously, and the chart should be refreshed when data retrieval is completed.

```
function getRevenueByYearData()

{
```

```
        var dataTable = new $MA.DataTable();

        dataTable.addColumn("year", "number");

        dataTable.addColumn("product", "string");

        dataTable.addColumn("revenue", "number");



        if (window.openDatabase)

        {

                var db = window.openDatabase("mademo", "1.0", "HTML5 MAKit Demo App", 1024 *
1024);

                if (!db)

                        alert("Failed to create DB");



                db.transaction(

                    function(tx) {

                        tx.executeSql("SELECT year, product, SUM(revenue) FROM ProductRevenue
GROUP BY year, product", [],

                                function(tx, result)

                                {

                                                var row;

                                                for(var i = 0; i < result.rows.length; i++) {

                                                row = result.rows.item(i);

                                                dataTable.addRow(row);

                                        }

                                        dataTable.refresh();

                                },

                                function(tx, e)

                                {

                                        alert(e.message);

                                });

                    }

                );

        }
```

```
        return dataTable;

}
```

The level 2 query retrieves the year, quarter, product, and sum (revenue):

```
var sql = "SELECT year, quarter, product, SUM(revenue) FROM ProductRevenue GROUP BY year, quarter,
product;";
```

The level 3 query retrieves the year, quarter, month, product, and sum (revenue);

```
var sql = "SELECT year, quarter, month, product, SUM(revenue) FROM ProductRevenue GROUP BY year,
month, product;";
```

Viewing the Zooming Chart

By default, the top-level zooming chart is shown. Click zooming buttons in the toolbar to view data by quarter and month.

Creating a Drilldown Group

Use a drilldown group to filter data and show it in different chart types. The filter action is triggered by a double-click or a tap on a chart. Drilldown also supports chart tables and semantic zooming charts.

Adding a Metadata File

This example defines a drilldown group with three charts: level 1 displays data by year; level 2 displays data by quarter; and level 3 displays data by month:

```
<DrillDownGroup>

        <Chart Name="chart_1" ChartType="column" Query="productRevenueByYear" Title="Revenue By
Year">

                <Category Column="year"/>

                <Series Column="product"/>

                <Values>

                        <Value Expression="revenue" Format="currency"/>

                </Values>

        </Chart>

        <Chart Name="chart_2" ChartType="column" Query="productRevenueByQuarter" Title="Revenue By
Quarter">

                <Category Column="quarter"/>

                <Series Column="product"/>

                <Values>

                        <Value Expression="revenue" Format="currency"/>

                </Values>

        </Chart>

        <Chart Name="chart_3" ChartType="column" Query="productRevenueByMonth" Title="Revenue By
Month">

                <Category Column="month"/>

                <Series Column="product"/>

                <Values>

                        <Value Expression="revenue" Format="currency"/>

                </Values>

        </Chart>

</DrillDownGroup>
```

Feeding Data

The data table returned by queries of the three drilldown levels must reflect the drilldown relationship. In this example, the level 1 query returns a table with these columns: “year”, ”revenue”. The level 2 query returns a table with these columns: “quarter”, “revenue”; additionally, the data is filtered by selected year. The level 3 query returns a table with these columns: “month”, “revenue”; additionally, the data is filtered by selected year and selected quarter.

This sample uses SQLite test data. The level 1 query retrieves the year and sum (revenue)—the same as for the semantic zooming chart.

The level 2 query retrieves the quarter and sum (revenue), which are filtered by selected year. The drilldown information is stored in args and required when retrieving data. There is one arg in level 2, two args in level 3, and so on. Each arg consists of information like itemValue and seriesName. itemValue is an array, so that drilldown can also support a semantic zooming chart. The filter by series action is triggered if only one series is selected.

```
function getRevenueByQuarterData(args)

{

        var dataTable = new $MA.DataTable();

        dataTable.addColumn("year", "number");

        dataTable.addColumn("quarter", "number");

        dataTable.addColumn("revenue", "number");



        if (window.openDatabase)

        {

                var db = window.openDatabase("mademo", "1.0", "HTML5 MAKit Demo App", 1024 *
1024);

                if (!db)

                        alert("Failed to create DB");



                db.transaction(

                    function(tx) {

                        var sql = "SELECT year, quarter, SUM(revenue) as quarterrevenue FROM
ProductRevenue";

                        if (args)

                        {

                                sql += " WHERE year = ";

                                sql += args[0].itemValues[0];

                        }

                        sql += " GROUP BY year, quarter;";

                        tx.executeSql(sql, [],

                                    function(tx, result) {

                                         var row;

                                         for(var i = 0; i < result.rows.length; i++) {

                                                 row = result.rows.item(i);

                                                 dataTable.addRow(row);
```

```
                                        }

                                        dataTable.refresh();

                                },

                                function(tx, e) {

                                        alert(e.message);

                                }

                                );

                        }

                );

        }

        return dataTable;

}
```

The level 3 query retrieves the month and sum (revenue), which are filtered by selected year and quarter:

```
Var sql = “SELECT year, quarter, month, product, SUM(revenue) AS monthrevenue, month FROM
ProductRevenue”;

if (args)

{

        sql += “ WHERE year = “;

        sql += args[0].itemValues[0];

        sql += “ AND quarter = “;

        sql += args[1].itemValues[0];

}

sql += “ GROUP BY year, month, product;”;
```

Viewing the Drilldown Chart

By default, the top-level drilldown chart appears. A breadcrumb trail indicates the selected values. Click any item in the breadcrumb trail to drill back.

Creating a What-If Chart

What-if analysis allows the user to dynamically change input data and see those changes reflected immediately in the chart. The chart value should be computed based on an expression instead of a column. Semantic zooming charts do not support what-if analysis. What-if controls require jQuery JavaScript and css files.

Adding a Metadata File

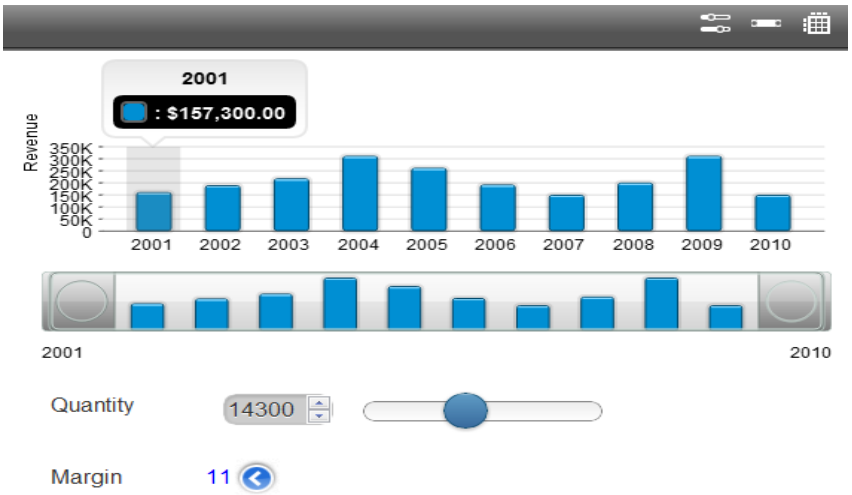
This example defines a chart that uses what-if analysis:

```
<Chart Name="chart_1" ChartType="column" Query="year_qty_margin" Title="Revenue By Year">

        <Category Column="Year"/>

        <Values>

                <Value Expression="Quantity * Margin" DisplayName="Revenue" Format="currency"/>

        </Values>

        <WhatIfColumns>

                <Column Name="Quantity" Description="Quantity"  VariableType="integer" Value="0"
Min="10000" Max="20000" ></Column>

                <Column Name="Margin" Description="Margin" VariableType="float" Value="0.0"
Min="0.0" Max="50.0" Step="0.5" Decimals="1" ></Column>

        </WhatIfColumns>
```

```
</Chart?
```

Viewing the What-If Chart

1. Click the **What-If** icon.
2. Select an item and change the value of the what-if column. The chart values are recomputed and refreshed based on the new value of the what-if column.



Showing the What-If View by JavaScript API

To show a what-if view using JavaScript, use:

```
$MA(chart_id).showWhatIfView(true)
```

Creating a Dynamic Analytics Chart

Dynamic analytics (DA) allows the user to dynamically change value of global variable and immediately see that new value reflected in the chart.

For dynamic analytics, compute the chart value based on an expression instead of a column. Use jQuery JavaScript and CSS files for DA controls, and load the DA variables before loading a DA chart. You can use both global variables and what-if columns in the same chart.

Adding a Global DA Variables Metadata File

This example defines a DA variable metadata file:

```
<Variables>

        <Variable Name="$testbool" DisplayName="Test Bool" VariableType="boolean" Value="0" />

        <Variable Name="$basevalue" DisplayName="Test Integer" VariableType="integer"
Value="50000" Min="0" Max="100000" />

        <Variable Name="$testinteger" DisplayName="Test description" VariableType="integer"
Value="0" Min="0" Max="100" />

        <Variable Name="$testfloat" DisplayName="Test description" VariableType="float"
Value="20000.0" Min="0.0" Max="50000" Step="0.5" Decimals="2" />
```

```
        <Variable Name="$testdate" DisplayName="Test date" VariableType="date" Value="2010-03-22"
/>

        <Variable Name="$testenum" DisplayName="Investment priority on" VariableType="enum"
Selection="1">

                <Choices>

                        <Choice Display="n/a" Value="n/a" />

                        <Choice Display="Web Ads" Value="Web Ads" />

                        <Choice Display="TV Ads" Value="TV Ads" />

                        <Choice Display="Newspaper Ads" Value="Newspaper Ads" />

                        <Choice Display="Road Shows" Value="Road Shows" />

                        <Choice Display="Conventions" Value="Conventions" />

                </Choices>

        </Variable>

</Variables>
```

Adding a Metadata File

This example defines a chart that uses dynamic analytics:

```
<Chart Name="chart_1" ChartType="column" Query="year_qty_margin" Title="Revenue By Year">

        <Category Column="Year"/>

        <Values>

                <Value Expression="if($testbool, Quantity * Margin, if($testenum=='aa',
$basevalue, if($testdate=='2010-03-22',$basevalue,Quantity * Margin + $basevalue + $testfloat)))"
DisplayName="Revenue" Format="currency"/>

        </Values>

        <WhatIfColumns>

                <Column Name="Quantity" Description="Quantity"  VariableType="integer" Value="0"
Min="10000" Max="20000" ></Column>

                <Column Name="Margin" Description="Margin" VariableType="float" Value="0.0"
Min="0.0" Max="50.0" Step="0.5" Decimals="1" ></Column>

        </WhatIfColumns>

</chart>
```

Loading Global DA Variables Metadata File and Binding DAChanged Event

1. Load the DA variables metadata file before loading the MA initialization script.

```
window.onload = function () {

        $MA.DA.load("../metadata/da_variables.xml");
```

```
        $MA.Chart.getStyles();

        arrange(1, 1);

        var chart = new $MA.Chart("chart_1", true);

        chart.bind("initialized", onInitialized);

        chart.load("../metadata/da_chart_bar.xml");

}
```

2. Bind the **DAChanged** event and refresh the chart:

```
$(window).bind("dachanged", onDAChanged);

function onDAChanged(e, daname, value)

{

        $MA("chart_1").refresh();

}
```

Viewing the DA Chart

1. Click the **DA** icon.
2. Change the value of the DA variables. The chart values are recomputed and refreshed based on the new variable value.

Showing the Dynamic Analytics View by JavaScript API

To show the Dynamic Analytics view using JavaScript, use:

```
$MA(id).showDynamicAnalysisView(true);
```

Advanced Features

Creating a Dynamic Chart

1. Create a chart at runtime; the syntax is the contents of XML metadata or corresponding JSON object.

```
var chart = new $MA.Chart("chart_1", true);

var syntax = {

        "Name" : id,

        "ChartType" : chartType,

        "Query" : "revenueByYear",

        "Title" : "Revenue",

        "Category" : {

                "Column" : "year"

        },

        "Values" : {
```

```
                "Value" : [ {

                        "Expression" : "revenue",

                        "Format" : "currency"

                } ]

        }

};

chart.create(syntax);
```

2. Change the chart property:

```
chart.setProperty("charttype", chartType);
```

3. Change the chart data:

```
chart.setDataTable(dataTable);
```

Creating a Localized Chart

1. Create a metadata file in a different language (for example, “de”) and place it in a subfolder of the `metadata` folder. The folder name is based on the locale code. The default language and currency symbol are based on system settings.
2. Change the MAKit locale:

```
$MA.setLocale("de");
```

3. Change the MAKit currency symbol:

```
$MA.setCurrencySymbol("€");
```

4. Include the jQuery locale files.
5. When you load the metadata file, the file in the `de` subfolder is loaded.

```
chart.Load("metadata/mychart.xml");
```

Sharing a Chart

1. Bind to the **chartshared** event:

```
chart.bind("chartshared", onChartShared);
```

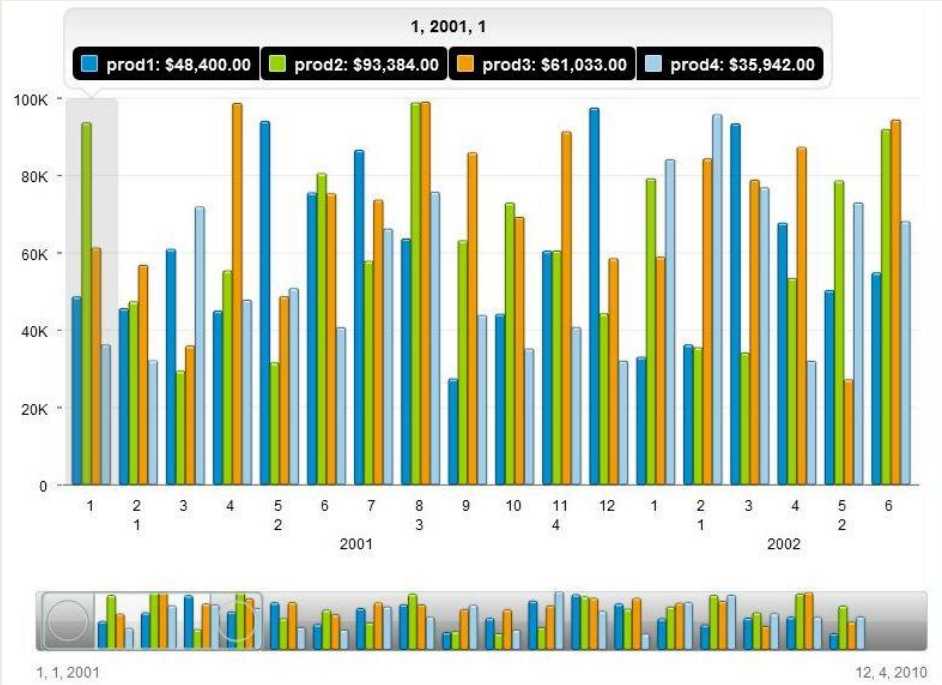
```
function onChartShared(id) {

        $MA.Graph.Share.show($MA(id), "../images/");

}
```

2. Click the **Shared** icon to display the chart sharing screen.
3. Add your own JavaScript wrapper function to save the chart to an image and send it via an e-mail message. The JavaScript wrapper function is usually written in native code and not part of MAKit.

Creating a Multiple Categories Chart

1. This example defines a three-level categories chart:

```
<Chart Name="chart_1" ChartType="column" Query="productRevenueByMonth" Title="Multiple Categories">

    <Categories Display="all">

        <Category Column="year"/>

        <Category Column="quarter"/>

        <Category Column="month"/>

    </Categories>

    <Series Column="product"/>

    <Values>

        <Value Expression="revenue" Format="currency"/>

    </Values>

</Chart>
```

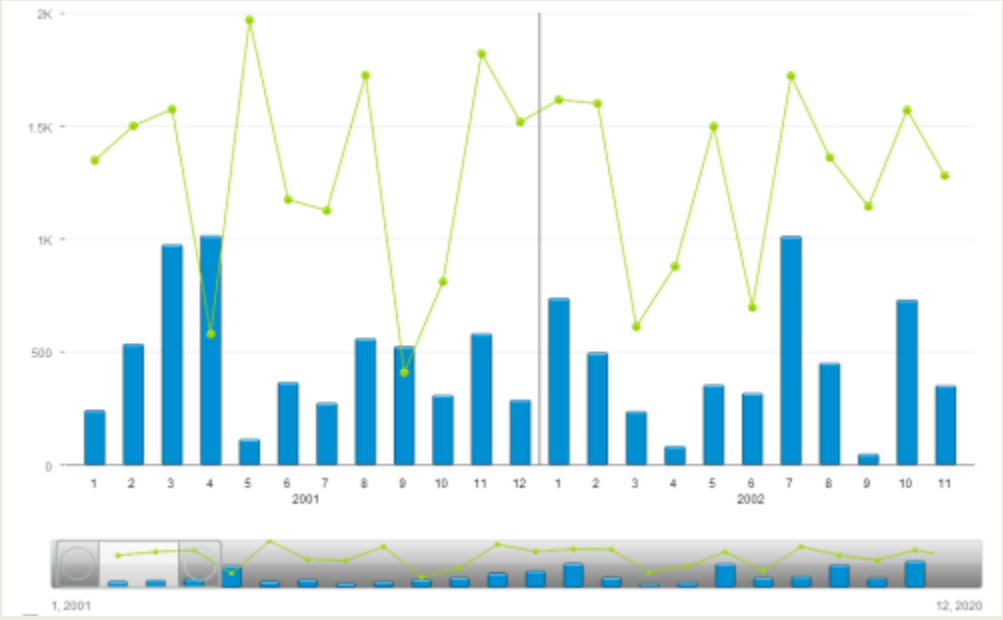


Creating a Combination Chart

1. This example defines two sets of data sharing same categories as two types of graph in a single chart:

```
<OverlayGroup Name="group_1" Animation="none">

<Categories Display="all">

        <Category Column="year"/>

        <Category Column="month"/>

</Categories>

<Layer Name="chart_1" ChartType="column" Query="query1">
```

```
        <Values>
                <Value Expression="revenue" Format="currency"/>
        </Values>
</Layer>
<Layer Name="chart_2" ChartType="line" Query="query2">
        <Values>
                <Value Expression="amount" Format="currency"/>
        </Values>
</Layer>
</OverlayGroup>
```



2. All existing Chart properties are support, with one new property "SecondaryAxis" introduced in "Values" element. The secondary value axis would be draw at the right of the graph if the property is set to "true". For each "Chart" element, "Name" property needs to be present and unique in the combination chart to identify each chart.
3. This feature is currently supported for column & line combination only. More combinations will be supported in next release.

Using the "Lite" Version

A lite version of MAKit is provided for users who do not require dynamic analysis and what-if features. The size of this version (`sap.makit.lite-2.3.min.js`) is 280KB, whereas the size of the full version (`sap.makit-2.3.min.js`) is 377KB.

Best Practices

Performance

- Improve performance by sorting data by category before feeding it to the chart.
- Performance may be affected by overusing expressions.

Usability

- Limit the number of series to 16; otherwise, you must change the default CSS file to support more colors.
- Limit the number of categories for bar charts and horizontal bar charts to 1000, for scrolling usability.
- Limit the number of categories for bubble charts to 100; otherwise, the entire chart is filled with bubbles.

Functions and Events

$MA

$MA is the namespace of MAKit.

Functions

Function Name	Description
$MA(id)	$MA static constructor. Gets the MA object.
setCurrencySymbol(currencySymbol)	Sets the currency symbol. The default currency symbol is based on system settings. This is a static function.
setImageFolder(folder)	Sets the images folder. The default folder is “./images/”. The images folder holds image files for chart-sharing icons and the range selector. This is a static function.
setLocale(locale)	Sets the locale. The default locale is based on system settings. This is a static function.
isBrowserCompatible	Check if MAKIT is compatible with the current browser.

$MA.Chart

A chart is a composite control that includes Graph, Table, Range Selector, What-if, and Dynamic Analytic views. A chart is created based on metadata that is defined in XML format and stored in an XML file or string.

An HTML DIV element is needed as a container for a chart. The chart must be created with an ID of HTML DIV element.

This is a sample of a simple chart:

```
window.onload = function () {

    var chart = new $MA.Chart("chart_1");

    chart.bind("maximize", onMaximize);

    chart.load("mychart.xml");

}



function onMaximize(id) {
```

```
        // Write your code to show Full View Chart

    }
```

This is a sample of a full-view chart:

```
    window.onload = function() {

        var chart = new $MA.Chart("chart_1", true);

        chart.bind("initialized", onInitialized);

        chart.bind("titlechanged", onTitleChanged);

        chart.load("mychart.xml");

    }


    function onInitialized(id) {

        // Write your code to show or hide chart components

    }


    function onTitleChanged(id, title) {

        // Write your code to show title on navigation bar

    }
```

Properties

You can change these properties using the chart setProperty function:

Property Name	Description
Category.Color	Sets the color of category axis.
Category.ShowGrid	Shows or hides grid line for category axis. value – [true, false].
Category.ShowLabel	Shows or hides axis label for category axis. value – [true, false].
Category.SortOrder	Sets the category sort order. value – [ascending, descending, none].
Category.ShowPrimaryLine	Shows or hides primary line for category axis. value – [true, false].
Category.Thickness	Sets the thickness of category axis.
Legend	Sets the chart legend. value – [“none”, “left”, “right”, “top”, “bottom”]
PieStyle	Sets the pie style. value – ["pie", "donut"].
Values.Color	Sets the color of value axis.
Values.Min	Sets the minimum value of value axis.

Values.Max	Sets the maximum value of value axis.
Values.ShowGrid	Shows or hides grid line for value axis. value – [true, false].
Values.ShowLabel	Shows or hides axis label for value axis. value – [true, false].
Values.ShowPrimaryLine	Shows or hides primary line for value axis. value – [true, false].
Values.Thickness	Sets the thickness of value axis.

Functions

Function Name	Description
Chart(id, [isFullView])	Chart constructor. id – ID of HTML DIV element. isFullView – [true, false].
applyCSS()	Applies style sheet to the chart.
bind(eventName, function)	Binds a function to a predefined event.
create(syntax)	Creates a chart object.
getCanvas()	Gets the chart canvas object, user can add custom drawing to the chart.
getCategoryObject()	Gets category array of objects (multiple values) based on the index specified. The return array contains the multivalue category objects. This method is supported for bar and line charts.
getDrawArea()	Gets the dimensions of graph draw area. The return object contains left, top, width, height of chart draw area.
getDrawingContentType()	Gets the drawing content type, for example, "graph" or "rangeselector".
getNumberOfCategories()	Gets number of categories.
getNumberOfVisibleCategories()	Gets number of visible categories. This method is supported for bar and line charts.
getNumberOfZoomingLevels()	Gets number of zooming levels.
getProperty(propertyName)	Gets value of the chart property. The property name can also be in dot notation format (for example: name, category.column and values[0].expression).
getSelectedCategory()	Gets the name of selected category.
getSelectedCategoryIndex()	Gets the index of selected category in visible categories.
getSelectedCategoryGroup()	Gets the selected zooming categories.
getSelectedSeries()	Gets the name of selected series.
getStyles()	Applies styles from css file. This is a static function.
getValueLoc()	Gets the scaled coordinate of value.
getZoomingLevel()	Gets current zooming level of chart.
isDynamicAnalysisViewVisible()	Determines whether the Dynamic Analytics view is visible.
isFullView()	Determines whether the chart is in full-view mode.
isRangeSelectorViewVisible()	Determines whether the Chart Range Selector view is visible.

isTableViewVisivle()	Determines whether the Chart Table view is visible.
isToolBarVisible()	Determines whether the chart toolbar is visible.
isValueBubbleVisible()	Determines whether the value bubble is visible.
isWhatIfViewVisible()	Determines whether the What-if view is visible.
load(fileName)	Loads a metadata file and creates a chart object.
preventDefaultRendering(value)	Prevents default rendering of MAkit. value – [true, false] (default false)
refresh()	Refreshes the chart.
retrieve(query)	Retrieves data through query function.
setCategorySortOrder(value)	(Obsolete) Replaced by category.sortorder property.
setDataPointsColor(value)	Sets color for some data points. value – JSON array of index and color (for example, [[1, “red”], [3, “green”]])
setDataTable(dataTable)	Sets data table to the chart.
setHorizontalBarHeight(value)	Sets the height of bar (for horizontal bar charts only).
setLegend(value)	(Obsolete) Replaced by the legend property. value – [“none”, “left”, “right”, “top”, “bottom”].
setLegendOffset(x, y)	Adjusts the legend position.
setMonoColor(value)	Renders all bubbles in a single color. value – [true, false].
setNumberOfVisibleCategories(value)	Sets number of visible categories. This method is supported for bar and line charts.
setPalette(palette)	Sets the color palette of chart.
setPieChartCategoryTextMaxLength(value)	Sets the text limit for pie chart category. This is a static function.
setProperty(propertyName, value)	Sets value to the chart property.
setSelectedCategoryIndex(value)	Sets the selected category index for initial display.
showTableValue(value)	Shows or hide the value table of horizontal bar charts. value – [true, false].
setValueBubbleStyle(styleObj)	Sets the value bubble style. styleObj.style – ["top", "float", “floattop”]. styleObj.showCategoryText - [true, false]. styleObj.showCategoryDisplayName - [true, false]. styleObj.showValueDisplayName - [true, false]. styleObj.showLegendLabel – [true, false]. styleObj.showNullValue – [true, false] (default true). styleObj.position – [“top”, “size”] (for pie charts only).
setZoomingLevel(level)	Sets chart to zooming level specified by user. level – a zero-based number.
showRangeSelectorView(value)	Shows or hides the Chart Range Selector view. value – [true, false].
showTableView(value)	Shows or hides the Chart Table view. value – [true, false].
showToolBar(value)	Shows or hides the chart toolbar. value – [true, false].

showValueBubble(value)	Shows or hides the value bubble. value - [true, false].
showWhatIfView(value)	Shows or hides the What-if view. value – [true, false].
showDynamicAnalysisView(value)	Shows or hides the Dynamic Analytics view. value – [true, false].
switchToFullView(value)	Sets the chart to simple view or full-view mode. value - [true, false].
unbind(eventName)	Unbinds the function to event.

Events

Event Name	Description
chartshared(id)	This event is triggered when the shared icon is clicked.
draw(id)	This event is triggered when chart is being drawn.
doubletap(id)	This event is triggered when a category is double-tapped. This event is not supported when the chart is used in DrillDownGroup.
initialized(id)	This event is triggered when the chart is initialized. The event should be added to chart before loading.
longpress(id)	This event is triggered when a chart is pressed and held for at least 2 seconds.
retrievestart(id)	This event is triggered before retrieving data.
retrieveend(id)	This event is triggered after retrieving data.
tap(id)	This event is triggered when a category is tapped.
titlechanged(id, title)	This event is triggered when the chart title is changed (for a full-view chart only).
maximize(id)	This event is triggered when the maximize icon of chart title bar is clicked (for a simple chart only).

$MA.DataTable

For MAKit HTML5, data is handled by users and assigned to a chart through the **executeMAQuery** function. All queries required by the chart should be handled by this function.

```
function executeMAQuery(chartObj, name, args)

{

    var dataTable;



    if (name == "query1")

            dataTable = getQuery1Data();

    else if (name == "query2")

            dataTable = getQuery2Data();

    …
```

```
        chartObj.setDataTable(dataTable);

    };
```

Each query function should create a data table, and within the table, all columns required by the chart.

```
    function getQuery1Data()

    {

        var dataTable = new $MA.DataTable();


        // Create columns required by chart

        dataTable.addColumn("column1", "number");

        dataTable.addColumn("column2", "string");

        dataTable.addColumn("column3", "datetime");


        // Append your data

        dataTable.addRow([123, “ABC”, “2011-02-03”]);


        return dataTable;

    }
```

The **setData** function of **FileDataTable** gets data from a file. Supported file types are “csv,” “xml,” and “json.”

```
function getFileData()

{

    var dataTable = new $MA.FileDataTable();


        dataTable.addColumn("column1", "number");

        dataTable.addColumn("column2", "string");

        dataTable.addColumn("column3", "datetime");


        // Append your data from file

        dataTable.setData("csv", "mydata.csv");

                return dataTable;

}
```

Functions

Function Name	Description
DataTable()	Data table constructor.
addColumn(name, type)	Creates a table column. Supported types are “number”, “string”, and “datetime”.
addRow(dataRow)	Appends data row to data table.
addRows(dataRows)	Appends data rows to data table.
getNumberOfColumns()	Gets number of columns.
getNumberOfRows()	Gets number of rows.
Refresh	Refreshes the parent control.
setData(type, value)	Sets the data to data table. type – [csv, xml, json].
setData(fileType, fileName)	Appends data from file to data table (for FileDataTable only). fileType – [“csv”, “xml”, and “json”].
setProperty(name, value)	Sets the table property.

$MA.DrillDownGroup

DrillDownGroup consists of charts that are created based on metadata that is defined in XML format and stored in an XML file or string.

An HTML DIV element is needed as a container for a DrillDownGroup. The DrillDownGroup must be created with an ID of HTML DIV element.

This is a sample of a DrillDownGroup:

```
window.onload = function () {

    var drillDownGroup = new $MA.DrillDownGroup("dd_1");

    drillDownGroup.bind("titlechanged", onTitleChanged);

    drillDownGroup.load("mydrilldowngroup.xml");

}



function onTitleChanged(id, title) {

    // Write your code to show title on navigation bar

}
```

Functions

Function Name	Description
DrillDownGroup(id)	DrillDownGroup constructor. id – ID of HTML DIV element.
bind(eventName, function)	Binds a function to event.
drillDown(itemNumber, [seriesName])	Drills to next chart based on the specified item number and series name.
drillBack()	Drills to prior chart.

drillUp(level)	Drills to previous chart based on the specified level.
getCurrentChart()	Gets the current chart.
getCurrentLevel()	Gets the current level.
refresh()	Refreshes the control.
showToolBar(value)	Shows or hides the toolbar. value – [true, false].
unbind(eventName)	Unbinds the function to event.

Events

Event Name	Description
drilldown(args)	This event is triggered when a category is double-tapped.
initialized(id)	This event is triggered when the DrillDownGroup is initialized. The event should be added to DrillDownGroup before loading.
titlechanged(title)	This event is triggered when the DrillDownGroup title is changed.

Tested Operating Systems and Browsers for MAKit

MAKit has been tested on these platforms:

Platform	Supported Versions
Windows desktop	Google Chrome version 15 or higher
Android phones and tablets	Android 3.1, 4.0.3, 4.1 with Chrome
iPhone and iPad	iOS 5.x and 6.0 with Safari

Expressions

Expressions compute chart data. A chart has three axes (x, y, and z), which are called category, series, and values, respectively. For values, you can specify an expression that lets you select how and what data should appear. For category and series, you can specify only a column name.

Syntax

This is the syntax for an expression:

```
expr = operand (operator operand)?

operand = column_name | function_name(arg [,arg[,arg…]]) | constant

operator = ( + | / | - | * ) | logical_operator

logical_operator: ( < | > | <= | >= | != | == )

arg = column_name | constant | expr

constant = string | numerical_value
```

- Always enclose string constants in single quotes; using double quotes in the same capacity generates an error.

- Column names are case-sensitive; however, function names are case-insensitive.
- Certain operators, such as > and <, must use HTML syntax (for example, `< >)` when entered manually in the metafile.

Examples of Expressions

```
ValueExpr = Col1+Col2*Col3+2.2   ( ColX == e.g. column name from a database table )
ValueExpr = 'Foo' + Col1         ( will convert whatever is in Col1 to string and concatenate )
ValueExpr = sqrt(Col2)           ( Square root of each value on each row )
```

When combining operands of different types, no implicit conversion is performed, except in some very special cases.

This example is invalid because the "a" and "hello" are of different datatypes, which could potentially create a column of mixed types:

```
    ValueExpr = if(a==1,a,'hello')
```

This is the correct way to write the expression:

```
    ValueExpr = if(a==1,string(a),'hello')
```

Implicit conversions are performed only when columns or constants are concatenated.

There is a special function for specifying dates as constants:

```
    ValueExpr = if( SomeColOfDateType > date(2010,9,1), a, b )
```

However, if you specify a date as a string according to the mask YYYY-MM-DD, it is converted to a date object (for example, "2009-09-01"). SAP recommends that you use the **date()** method as described here.

You can nest if-statements; however, too many nesting levels may be difficult to read:

```
    ValueExpr = if( a>b, if( a>c, a, c), b )

    ValueExpr = if (a >b, if (a>c, a, if(a>d, e, f), g ), h ) // you get the picture
```

Expression Types

Each chart has this syntax:

```
<Category Column="" ../>
<Series Column="" … />
<Values><Value Expression="" …/></Values>
```

For example, an application has a table you want to include in a column chart. You must map the values from the table to the chart. To do this, use XML as described above.

The values you want to show in a chart are shown below—three columns of different height.

Project_Group	Members
Alpha	20
Beta	30
Gamma	55

To show these values in the chart, you must specify the category, in this case, "Project_Group." The category becomes the x-axis. The series is set to empty ("") and the value in the y-axis is "Members."

The XML snippet looks like this:

```
<Category Column=”Project_Group” …/>
  <Series Column-“” …/>
  <Values>
  <Value Expression=”Members” … />
</Values>
```

There is no value expression per se—the column name is used as a value expression. To specify a real expression, for example, to add 10% more members, use Expression=“Members*1.10”.

Note: There are a number of runtime functions you can use in expressions. These functions are described later in this document.

Internally, expressions work with columns, so using “Members*1.10” means that every row in the table gets 10% more members. If you have two columns—Column1 and Column2—and you write “Column1 + Column2”, the two columns are added together row by row.

This example takes each row from the Age column and adds 10% to each value, creating a new value array that has the same number of rows:

```
    ValueExpr= Age*1.1
```

The following example calculates the sales revenue of a company.

```
    ValueExpr=Amount*(UnitPrice–UnitCost)
```

Under normal circumstances, developers do not interact directly with the parser and lexer. Instead, they specify the value expressions in a middle layer.

Runtime Functions

You can use runtime functions in expressions; however, make sure you use the correct type of function on the correct datatype. For example, do not use **Abs** (Name) if Name is a column containing strings.

Note: Column names are case-sensitive, but function names are not.

Numeric Functions

```
//
// argument is a constant/column name
//
```

Function Name	Description
Abs(column)	Abs(-1) -> 1, Abs(col) does |col| on all values.
Ceiling(column)	Ceiling(1.01)->2.0
Cos(column)	Cos(1)->0.9998477
Exp(column)	Log(Exp(2)) == 2
Fact(column)	5!->120, be careful with overflow.
Floor(column)	10.1 -> 10.0
Int(column)	12.2 -> 12, use round for rounding values.
Log(column)	Log(Exp(1))=1
LogTen(column)	LogTen(10*10)==2
Mod(column, dividend)	10%5 == 0, 3%2 == 1

Neg(column)	Neg(-1)->1,Neg(1)->-1 Note: Neg is different from Abs. Use Neg when a negative value is needed, since unary minus is not supported. For example, 2*-1 is not allowed; instead write 2*Neg(1).
Round(column,decimals)	Rounds off a value to decimals. The second argument "decimals" can only be a constant and cannot be a column name.
Sign(column)	Sign(1.0)->1,Sign(-121)->-1
Sin(column)	Sin(1)->0.01745241
Sqrt(column)	Sqrt(3)->1.7320508…
Tan(column)	Tan(1)->0.01745506
Truncate(column)	12.32123->12.

DateTime Functions

```
//
// date time functions only work on columns (vectors) of type date, in general it is
// impossible to use them on constant values since there is no date type recognized
// in the parser.
// for example, not allowed: Day( "2010-05-10" ); allowed: Day( MyDateCol )
// (you can, however, use the date function for date constants, see below)
```

Function Name	Description
Date(y,m,d[,hh,m,s] | column)	This versatile function can create a constant by using constant values for year, month, and so on, or converting a datetime column to a date datatype. For example, date(2010,9,1) creates a constant date that can be used in expressions and date(mydatecol) removes hh:mm:ss from the datetime in the column. This may be when comparing datetimes when the time of day does not matter.
Day(column)	Returns the day number 1 – 31/30/28.
DayNumber(column)	1=Sunday, 2=Monday,…
Hour(column)	Returns hour of day.
Minute(column)	Returns minute of hour.
Month(column)	Returns month of date.
RelativeDate(column, days)	Adds days to date, days== constant value, cannot be a column name.
RelativeTime(column, seconds)	Adds seconds to date timestamp== constant value, cannot be a column name.
Second(column)	Returns seconds; column is assumed to contain date objects.
Year(column)	Returns year of date; column is assumed to contain date objects.

String Functions

```
//
// functions operate on columns if not stated otherwise
//
```

Function Name	Description
Left(column, numberOfChars)	left(‘abcdef’,3)->’abc’, left(Column1,1)
LeftTrim(column)	‘ abc ‘ -> ‘abc ‘
Len(column)	Returns number of characters
Lower(column)	‘MYSTRING’ -> ’mystring’
Match(column, string)	Returns 1 for match, 0 for no match
Mid(column, start [, length])	mid(‘abcdef’,3,2)->’cd’
Pos(column, string [, startPos])	pos(‘abcdef’,’c’)->3
Replace(column, startPos, numberOfChars, string)	“1234” -> “1XX4”. startPos =1,…
Right(column, numberOfChars)	right(‘abcdef’,3)->’def’
RightTrim(column)	‘ abc ‘ -> ‘ abc‘
String(column)	Converts the column to strings
Trim(column)	‘ hello ‘ -> ‘hello’
Upper(column)	‘mystring’ -> ‘MYSTRING’
WordCap(column)	‘bob alice eve’ -> ‘Bob Alice Eve’

Miscellaneous Functions

Function Name	Description
if(condition,expr1,expr2)	If ‘condition’ is nonzero, expr1 is evaluated; if zero, expr2 is evaluated.
and(expr1,expr2[,…])	This can, for example, be used in an if statement in this way: if(and(expr1,expr2),res1,res2).
or(expr1,expr2[,…])	As above.
not(expr1==expr2)	unary not(expr1==expr2) is the same as expr1!=expr2.

Metadata File

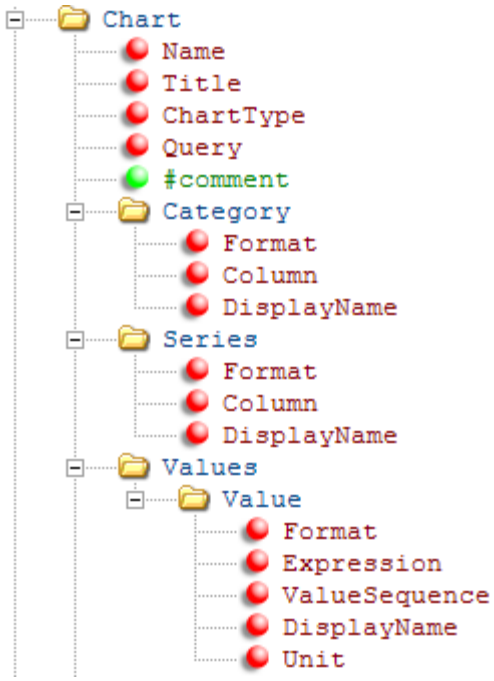
The metadata file is a configuration file for MAKit that contains information about chart, variables, and the interdependencies between charts. The metadata file also connects charts to queries.

The format of the metadata file must be a well-defined XML file.

Note*:* All node names and attribute names have a capitalized first letter.

Chart

The **Chart** node is the root node for the metadata. It contains the description of a chart.



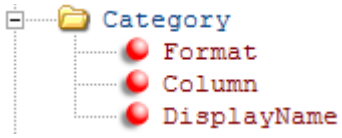
The **Chart** node contains a **Name**, an optional **Title** for the chart, and a **ChartType** , for example, BAR.

Supported chart types include: bar, column, line, pie, and bubble.

Query is a named query, for example, “RevenueByYear,” that resides in the application and is called by MAKit to retrieve chart data. The application uses the **Query** to retrieve data from the appropriate datasource and feed it back to MAKit.

There are a number of nodes under **Chart:**

Category

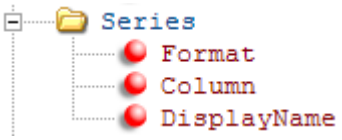


The **Category** column contains the category; typically, the x-axis in an xy-chart. **Column** is the actual unique name of the column to use for the **category**. The **Format** attribute lets you modify the data format; for example, **Format**="WeekDay" converts the date to a weekday name in the chart. **DisplayName** can be any text to describe this category. Both **Format** and **DisplayName** are optional.

Supported Format Strings

Name	Type Req	Description
Month	DateTime	For example, "January"
ShortMonth	DateTime	For example, "Jan"
WeekDay	DateTime	For example, "Monday"
ShortWeekDay	DateTime	For example, "Mon"
ShortDate	DateTime	Depends on locale
LongDate	DateTime	Depends on locale
Currency	Number	Depends on locale, for example, $10.20
Percent	Number	Displays as percent, for example, 10 -> 10%
Rounded{n} (n = 0, 1, 2, .. 9 }	Number	Rounds number to n decimals

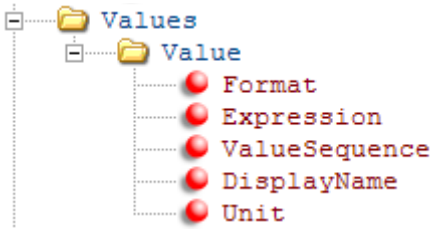
Series



Series specifies the z-axis. However, since the charts are two-dimensional, series are drawn next to each other. An example is a table that shows the number of employees year for year (category) where gender is the series, and the number of employees is the value.

Column is the actual unique name of the column to use for **Series**. The **Format** attribute (optional) lets you modify the data format; for example, Format="WeekDay" converts the date to a weekday name in the chart. See "Supported Format Strings" on page 39 for supported format types. **DisplayName** (optional) can be any text to describe this series.

Values



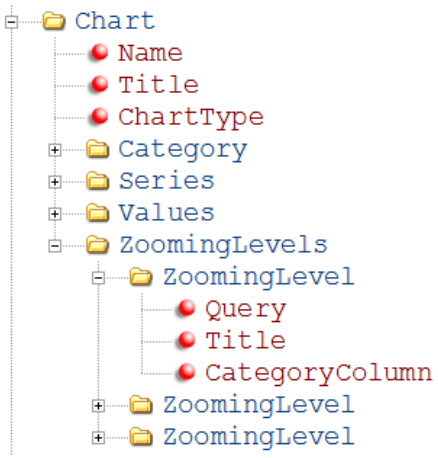
The **Values** node contains one or more **Value** nodes. The **Value** expression shows the values in the chart. Values can optionally use an expression instead of a column name, for example, **Expression**=”Amount*1.1” **DisplayName**=”Added 10% to price”.

The **Format** attribute lets you modify the data format; for example, Format=”Currency” converts the amount to currency in the chart. See “Supported Format Strings” on page 39 for supported format types. **DisplayName** can be any text to describe this series.

Typically, a chart uses only one value expression; however, certain chart types may use multiple expressions. The **Values** collection groups such variables together to allow for support of additional chart types in future releases.

Semantic Zooming

Semantic zooming allows the chart to display data in several zooming levels, for example, by year, quarter, and month. Chart metadata must include a definition for each zooming level.



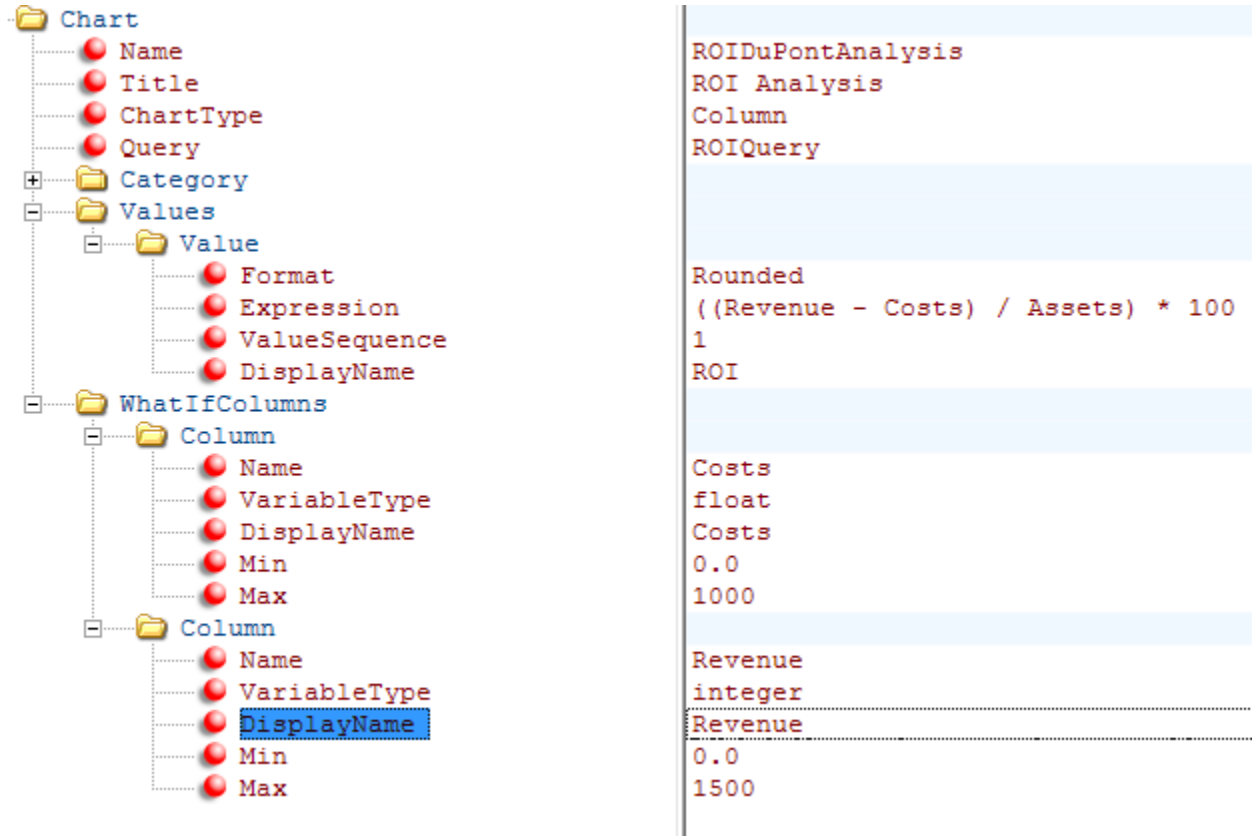
Each zooming level definition includes:

- **Query** – the query name for retrieving data of this level.
- **Title** – optional title for this level.
- **CategoryColumn** – the category column for this level.

Unlike nonzooming charts, semantic zooming charts have different query and category columns in each level. In the metafile for semantic zooming, remove the **Query** attribute for the **Chart** node, and the **Column** attribute for the **Category** node. Add one **Query** attribute and **CategoryColumn** attribute for the **ZoomingLevel** node.

What-If Analysis

What-if analysis allows the user to dynamically change input data and see instant updates in the chart. To enable this feature, define what-if columns in the chart.



The metadata of this chart is:

```
<Chart Name="ROIDuPontAnalysis" Title= "ROI Analysis" ChartType="Column" Query="ROIQuery" >

    <Category Format="" Column="Year" DisplayName="Year" />

    <Values>

        <Value Format="Rounded" Expression="((Revenue - Costs) / Assets) * 100" ValueSequence="1"
DisplayName="ROI" />

    </Values>

    <WhatIfColumns>

        <Column Name="Costs" VariableType="float" DisplayName="Costs" Min="0.0"  Max = "1000"/>

        <Column Name="Revenue" VariableType="integer" DisplayName="Revenue" Min="0.0" Max = "1500"/>

    </WhatIfColumns>

</Chart>
```

The Values expression is "((Revenue - Costs) / Assets) * 100", which means that the calculation involves three columns from the input table: Revenue, Costs, and Assets. The user can select any of these columns for what-if analysis.

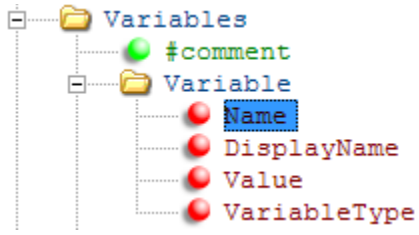
For each what-if column, provide these attributes:

- **Name** – the column name.

- **VariableType** – the datatype of that column. Different datatypes are presented as different controls in the UI. MAKit supports float and integer types.
- **DisplayName** – a text description of the variable. This text appears to the end user in the UI.
- **Min** – the minimum value of the data range.
- **Max** – the maximum value of the data range.

Variables

In MAKit for HTML5, you can define global variables in a separate file. The format of the variable file must be a well-defined XML file. When you load this file into MAKit, you can then use the global variables in the value expression of the chart.



Variable contains a name, an optional description, a variable type and, depending on the type, an optional value.

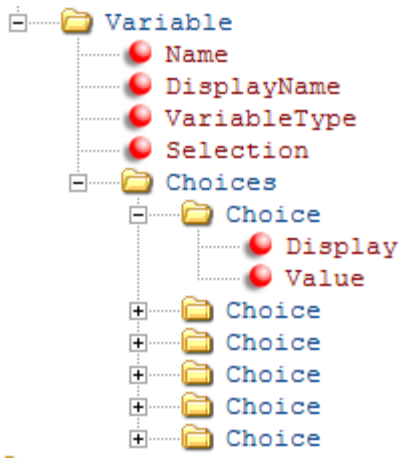
This is an example of a variable definition:

```
<Variables>
        <Variable Name="$testbool" DisplayName="Test Bool" VariableType="boolean" Value="0" />
        <Variable Name="$basevalue" DisplayName="Test Integer" VariableType="integer" Value="50000" Min="0"
Max="100000" />
        <Variable Name="$testinteger" DisplayName="Test description" VariableType="integer" Value="0" Min="0"
Max="100" />
        <Variable Name="$testfloat" DisplayName="Test description" VariableType="float" Value="20000.0" Min="0.0"
Max="50000" Step="0.5" Decimals="2" />
        <Variable Name="$testdate" DisplayName="Test date" VariableType="date" Value="2010-03-22" />
         <Variable Name="$testenum" DisplayName="Investment priority on" VariableType="enum" Selection="1">
                <Choices>
                        <Choice Display="n/a" Value="n/a" />
                        <Choice Display="Web Ads" Value="Web Ads" />
                        <Choice Display="TV Ads" Value="TV Ads" />
                        <Choice Display="'Newspaper Ads'" Value="'Newspaper Ads'" />
                        <Choice Display="'Road Shows'" Value="'Road Shows'" />
                        <Choice Display="'Conventions'" Value="'Conventions'" />
                </Choices>
        </Variable>
</Variables>
```

Each variable uses one or more of these attributes:

- **Name** – the identity of a variable. The variable name must start with $, be defined, and be unique in a chart.
- **VariableType** – the value type of the variable (required). Supported variable types are: `integer`, `float`, `boolean`, `date`, `enum`.
- **Min** – the minimal value of the variable; applicable to `integer` and `date` variables.
- **Max** – the maximum value of the variable; applicable to `float`, `integer`, and `date` variables.
- **Value** – the default value of the variable. Every variable except `enum` must have a default value. `enum` has an attribute called **Selection**, which is a list of values that can be selected from.
- Boolean variable values – these values indicate “true”: true, True, TRUE, yes, Yes, YES, 1. These values indicate “false”: false, False, FALSE, no, No, NO, 0.
- Date variable values must be specified with a date statement using the form “date(yyyy, mm, dd)”; for example, “date(2000,12,30)”.

- **Description** – a text string that describes the variable (required). This string appears along with the control.



The **Display** attribute contains a text string that is shown to the user, and **Value** is the value used in the value expression.

```
<Variable Name="$shown_region" Type="enum" Selection="0" DisplayName="Show data in region">
    <Choices>
            <Choice Display="North America" Value="North America"/>
            <Choice Display="Asia Pacific" Value="APC"/>
            <Choice Display="Europe" Value="Europe"/>
            <Choice Display="Africa" Value="Africa/">
            <Choice Display="South America" Value="South America"/>
    </Choices>
</Variable>
```

An enum variable allows the user to pick one string value from a collection of strings, which are stored in the **Choices** section. Each choice has a **Value** attribute and a **Display** attribute. If the **Display** attribute is defined, it is used; otherwise, the **Value** is used directly.

There can be an arbitrary number of variables. Each variable type has its own set of attributes:

Variables	
#comment	Contains all variables available shared by all charts
Variable	
Name	$testbool
DisplayName	Test description
Value	1
VariableType	boolean
Variable	
Name	$testinteger
DisplayName	Test description
Value	0
Min	0
Max	100
VariableType	integer
Variable	
Name	$testfloat
DisplayName	Test description
Value	0.0
Min	0.0
Max	100.0
Step	0.5
VariableType	float
Decimals	2
Variable	
Name	$testdate
DisplayName	Test description
Value	2010-03-22
Min	1962-01-23
Max	2020-01-01
VariableType	date
Variable	
Name	$testenum
DisplayName	Investment priority on
VariableType	enum
Selection	0
Choices	
Choice	
Display	n/a
Value	n/a

Here is an example of variable usage:

```
<Chart Name="chart_1" ChartType="column" Query="year_qty_margin" Title="Revenue By Year">
        <Category Column="Year"/>
        <Values>
                <Value Expression="if(Probability < $basevalue, Probability, 0)" DisplayName="Revenue"
Format="currency"/>
        </Values>
</Chart>
```

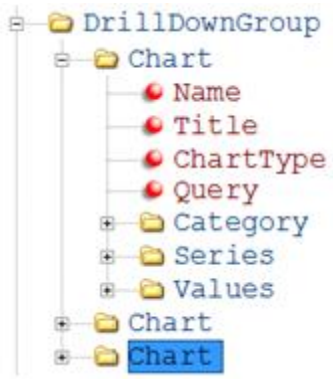
The **Chart** node shows the amount of sales opportunities in each stage. The chart uses a variable (`$basevalue`), which allows the user to show opportunities that have a probability larger than a threshold only. The chart allows the user to adjust this threshold, by means of a slider control.

DrillDownGroup

Use **DrillDownGroup** to create a link between charts, for example, when something like a category is picked in the first chart, and it affects what to show in the next chart. The selected category is passed as a parameter to the query of the next chart.

The **Name** of the drilldown group is used for the layout, which is then applied on all charts in the group. This rule applies to all collections (groups).

A drilldown group contains multiple chart nodes. Each chart node defines the chart of a drilldown level.



Themes

MAKit provides three themes: default, green, and blue. Users can customize additional themes.

The MAKit default theme includes these `css/image` files:

- WebContent/css/chart.css – chart styles.
- `WebContent/css/digitpicker.css` – digitpicker Web control styles.
- WebContent/images/ – all images used by chart.css and digitpicker.css.
- `WebContent/css/jquery/*.css` – all jQuery-related styles.

The blue and green themes are located in `WebContent/demo_debug/css/themes`. These themes override the default chart, JavaScript control, and page styles.

Using a Static Theme

You can add static link references in the page header to specify a theme; for example, to use the blue theme:

1. Add the css file reference in the page:

```
<link rel="stylesheet" type="text/css" class="ui-theme" href="css/blue.css">
```

2. Load the chart style on the page's onload event:

```
$MA.Chart.getStyles();
```

Using a Dynamic Theme

You can specify a theme in JavaScript. For example, users can use a theme in a button click.

Use the green theme by calling the function **updateCSS**, defined in `layout.js`:

```
updateCSS('green',function(success,link)
{
        if (success)
        {
           $MA.Chart.getStyles();
           $MA("chart_1").applyCSS();//apply the theme to chart
        }
});
```

User-Defined Theme for Chart

You can override the chart theme by defining a new chart palette.

1. Define a new palette for the chart:

```
var blueDeepOrangePalette = [
   0x2895c9,
   0x64b2dd,
   0xa8cfe5,
   0xc9e4f3,
   0xdcce9f,
   0xe1af48,
   0xe99f20,
0xdb7e2a];
```

2. Apply the new palette to the chart:

```
$MA.Chart.setPalette(blueDeepOrangePalette);
 $MA("chart_1").refresh();
```

Theme Definition

This section introduces Web controls and theme definitions for graphs.

Page

```
body {
            margin: 0;
            font-family: Helvetica;
}
```

Web Controls

DatePicker

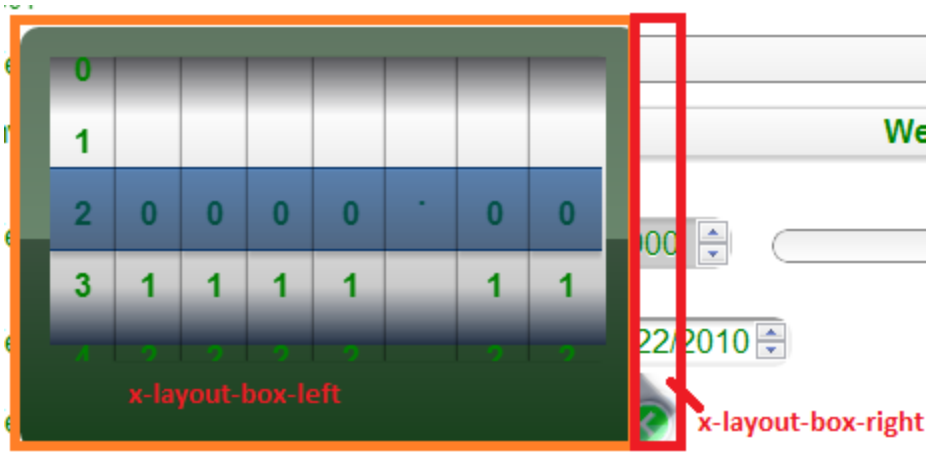
```
.ui-datepicker {   //background
background: #d3eb77;
}
.ui-datepicker td a{  //date color
color:green;
```

```
}

.ui-datepicker-title { //title color
 color:green;
}

.ui-datepicker-date-input {//the input text color
        color:green;
}
```

Slider

```
a.ui-btn-up-b{ //color for button up
 background-image:-webkit-linear-gradient(top,#659C7E,#39A54D);
 background-color:green;
}
a.ui-btn-hover-b {// color for button hover
        background:                    #58C85E !important;
        background-image: -webkit-linear-gradient(top, #7AC389, #58C85E) !important;
}
a.ui-btn-down-b {//color for button down
        background:                    #6BC26D;
        background-image: -webkit-linear-gradient(top, #4D985D, #6BC26D);
 }
.ui-body-c select{//color for button selected
        color:green;
}
```

DigitPicker



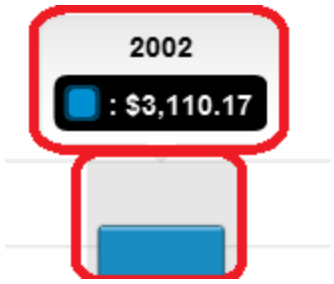

```
#MADigitPickerControl div.x-layout-box-left {
background: -webkit-gradient(linear, left top, left bottom, from(rgb(96,118,98)), to(rgb(24,65,28)), color-stop(0.5, rgb(105,134,108)), color-stop(0.5, rgb(58,81,62)));
}

label.digitpicker-value{
color:green;
}
a.digitpicker-button{
        background:transparent url("digitpicker-button-green.png") no-repeat center;
}
```

Chart

```
.chart {
        background:#FFFFFF;
}
```

Slider

```
.chart_value_bubble {// The background and foreground for bubble text

        background: #000000;
        color: #FFFFFF;
}
```

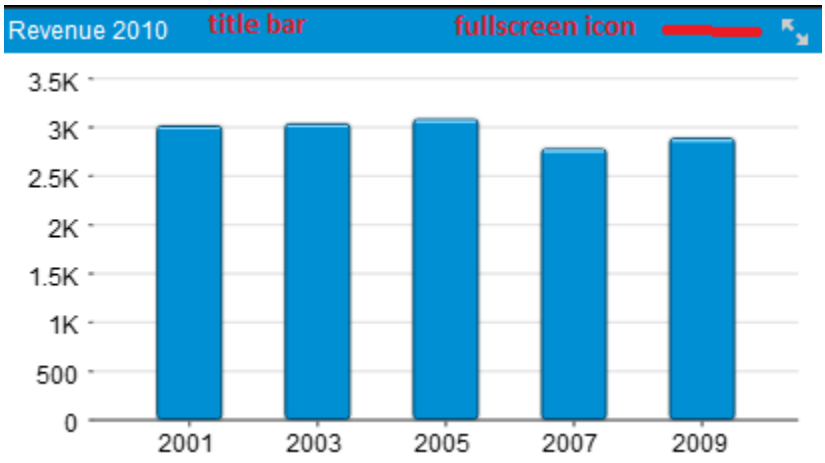



```
.chart_value_bubble_outline
{
        background: rgba(128, 128, 128, 0.2);
        color:#808080;
}
```

Range Selector

```
.chart_range_selector_text {// the text for range selector
        font-size: 12px;
        font-family:Helvetica;
        color:#707070;
}
```

Title Bar




```
.chart_titlebar {
        background: #008FD3
```

```
}

.chart_titlebar > .fullscreen_icon {
        margin-top: 5px;
        background: url(images/fullscreen_icon.png) no-repeat;
        cursor:pointer;
}
```

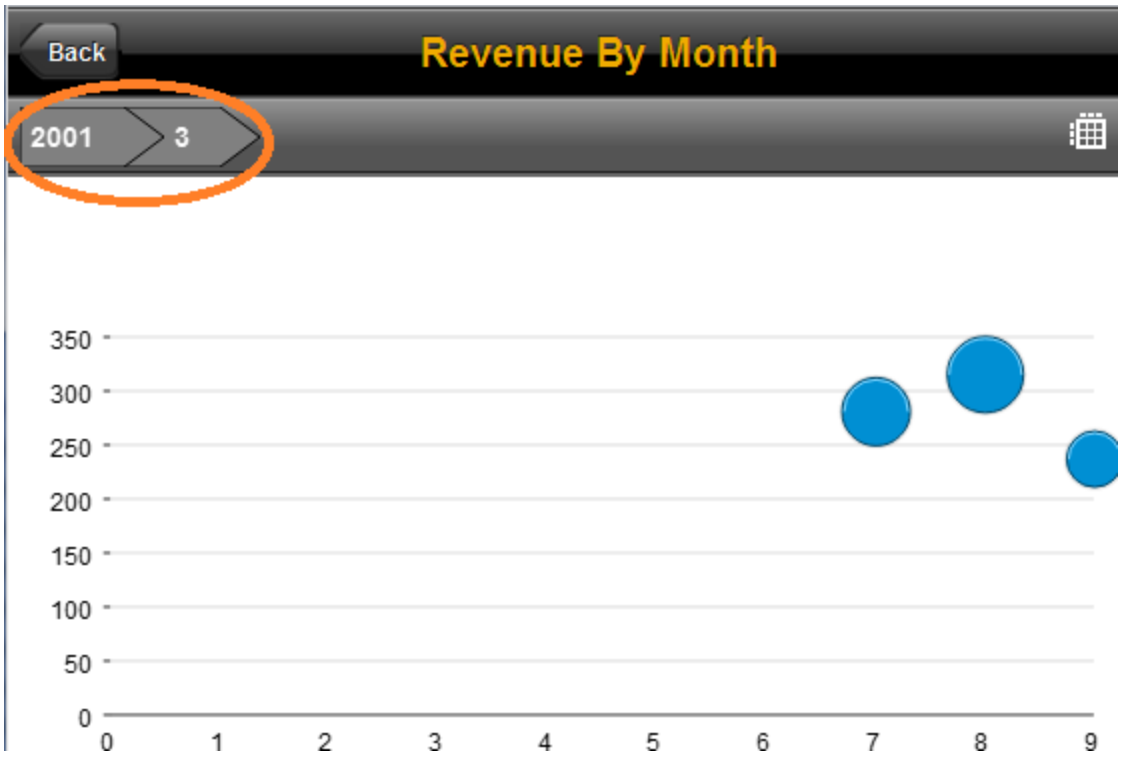
Toolbar

```
.chart_toolbar {
        box-sizing: border-box;
        -moz-box-sizing: border-box;
        border-bottom: 1px solid #666666;
        border-top: 1px solid #666666;
        background: url(images/chartToolbar.png) #666666 repeat-x;
}

.chart_toolbar > .chart_button {  //button on toolbar
        border: 3px;
        background: #222222;
        -moz-border-radius:6px;
        -webkit-border-radius:6px;
        border-radius:6px;
        font-family: inherit;
        font-size: 12px;
        font-weight: bold;
        color: #fff;
}

.chart_toolbar  > .chart_button:hover {
  background: #333333;
  color: #ccc;
}

.chart_toolbar  > .chart_button:active {
  background: #666666;
```

```
}

.chart_toolbar > .table_button { //table button
        border: 0px;
        background: url(images/table_icon.png) no-repeat;
}

.chart_toolbar > .rangeselector_button {//rangeselector button
        border: 0px;
        background: url(images/rangeselector_icon.png) no-repeat;
}

.chart_toolbar > .share_button {//share button
        border: 0px;
        background: url(images/e-mail_icon.png) no-repeat;
}

.chart_toolbar > .whatif_button {//whatif button
        border: 0px;
        background: url(images/whatif_icon.png) no-repeat;
}

.chart_toolbar > .da_button {//da button
        border: 0px;
        background: url(images/da_icon.png) no-repeat;
}

.chart_toolbar h1 {//title on toolbar
        overflow: hidden;
        margin:auto;
        font-size: 20px;
        font-weight: bold;
        text-shadow: rgba(0, 0, 0, 0.4) 0px -1px 0;
        text-overflow: ellipsis;
        white-space: nowrap;
        color: #F0AB00;
        display:inline;
}
```

Breadcrumbs

Breadcrumbs identify the selections that have been made in a drilldown chart.




```
.chart_breadcrumbs {
  background: #808080;
  color: #FFFFFF;
}
```

Table

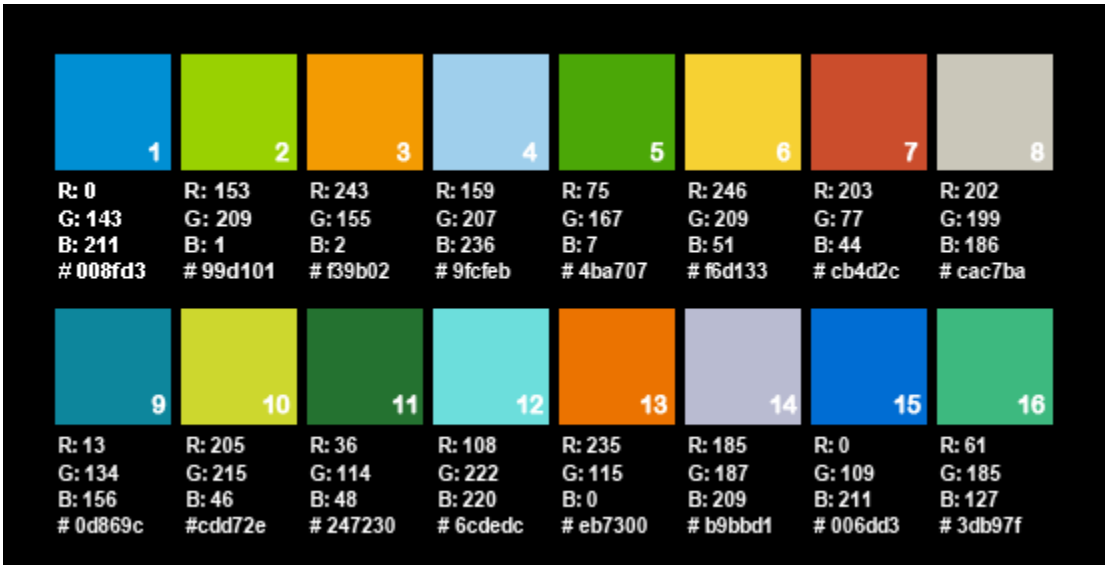
year	revenue
2001	$3,000.61
2002	$3,110.17

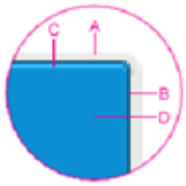
```
.chart_table {//general style
        background:#FFFFFF;
        font-size: 14px;
        font-family:Helvetica;
        color:#000000;
}

.chart_table_dimension_title {//dimension title
        background:#DDDDDD;
```

```
}

.chart_table_row_header {//row header
        background:#DDDDDD;
}

.chart_table_column_headers {//column header
        background:#F7F7F7;
}

.chart_table_contents {// contents
        background:#F7F7F7;
}
```

Graph

The default theme uses the SAP® mobile palette, which comprises 16 colors. Using themes other than the default overrides this palette. This section, however, uses the default theme to show how to define a graph theme.





A – Outline: Outline 3px opacity 0.1; border radius: 5px

B – defaultStrokePalette: Border 1px; border radius: 2px

C – defaultInsetShadowPalette: Inset shadow 0 1px 0

D – defaultFillPalette: Fill color

A – Outline

```
It’s a const value: rgba(0, 0, 0, 0.1).
```

B – defaultStrokePalette

```
.defaultStrokePalette{
}
.defaultStrokePalette  .color1{
        color: #00486A;
}
.defaultStrokePalette  .color2{
        color: #4C6800;
}
.defaultStrokePalette  .color3{
        color: #794D01;
}
.defaultStrokePalette  .color4{
        color: #506876;
}
.defaultStrokePalette  .color5{
        color: #255303;
}
.defaultStrokePalette  .color6{
        color: #7BCC19;
}
.defaultStrokePalette  .color7{
        color: #662716;
}
.defaultStrokePalette  .color8{
        color: #65645D;
}
.defaultStrokePalette  .color9{
        color: #06434E;
}
.defaultStrokePalette  .color10{
        color: #666B17;
}
.defaultStrokePalette  .color11{
        color: #123918;
}
.defaultStrokePalette  .color12{
        color: #366F6E;
}
.defaultStrokePalette  .color13{
```

```
        color: #763A00;
}
.defaultStrokePalette  .color14{
        color: #5C5D68;
}
.defaultStrokePalette  .color15{
        color: #00376A;
}
.defaultStrokePalette  .color16{
        color: #1F5D40;
}
```

C – defaultInsetShadowPalette

```
.defaultInsetShadowPalette{
}
.defaultInsetShadowPalette  .color1{
        color: #80D6FF;
}
.defaultInsetShadowPalette  .color2{
        color: #D3FF80;
}
.defaultInsetShadowPalette  .color3{
        color: #FFD180;
}
.defaultInsetShadowPalette  .color4{
        color: #D6F0FF;
}
.defaultInsetShadowPalette  .color5{
        color: #B9FF85;
}
.defaultInsetShadowPalette  .color6{
        color: #FFEB99;
}
.defaultInsetShadowPalette  .color7{
        color: #FFB09B;
}
.defaultInsetShadowPalette  .color8{
        color: #FFFEF5;
}
.defaultInsetShadowPalette  .color9{
        color: #89EDFF;
}
.defaultInsetShadowPalette  .color10{
```

```
        color: #F9FF9B;
}
.defaultInsetShadowPalette  .color11{
        color: #A8FFB5;
}
.defaultInsetShadowPalette  .color12{
        color: #BEFFFE;
}
.defaultInsetShadowPalette  .color13{
        color: #FFBE80;
}
.defaultInsetShadowPalette  .color14{
        color: #F0F2FF;
}
.defaultInsetShadowPalette  .color15{
        color: #80C1FF;
}
.defaultInsetShadowPalette  .color16{
        color: #AAFFD7;
}
```

D – defaultFillPalette

```
.defaultFillPalette{
}
.defaultFillPalette  .color1{
        color: #008fd3;
}
.defaultFillPalette  .color2{
        color: #99d101;
}
.defaultFillPalette  .color3{
        color: #f39b02;
}
.defaultFillPalette  .color4{
        color: #9fcfeb;
}
.defaultFillPalette  .color5{
        color: #4ba707;
}
.defaultFillPalette  .color6{
        color: #f6d133;
}
.defaultFillPalette  .color7{
```

```
        color: #cb4c2d;
}
.defaultFillPalette  .color8{
        color: #cac7ba;
}
.defaultFillPalette  .color9{
        color: #0d869c;
}
.defaultFillPalette  .color10{
        color: #cdd72e;
}
.defaultFillPalette  .color11{
        color: #247230;
}
.defaultFillPalette  .color12{
        color: #6CDEDC;
}
.defaultFillPalette  .color13{
        color: #EB7300;
}
.defaultFillPalette  .color14{
        color: #B9BBD1;
}
.defaultFillPalette  .color15{
        color: #006DD3;
}
.defaultFillPalette  .color16{
        color: #3DB97F;
}
```

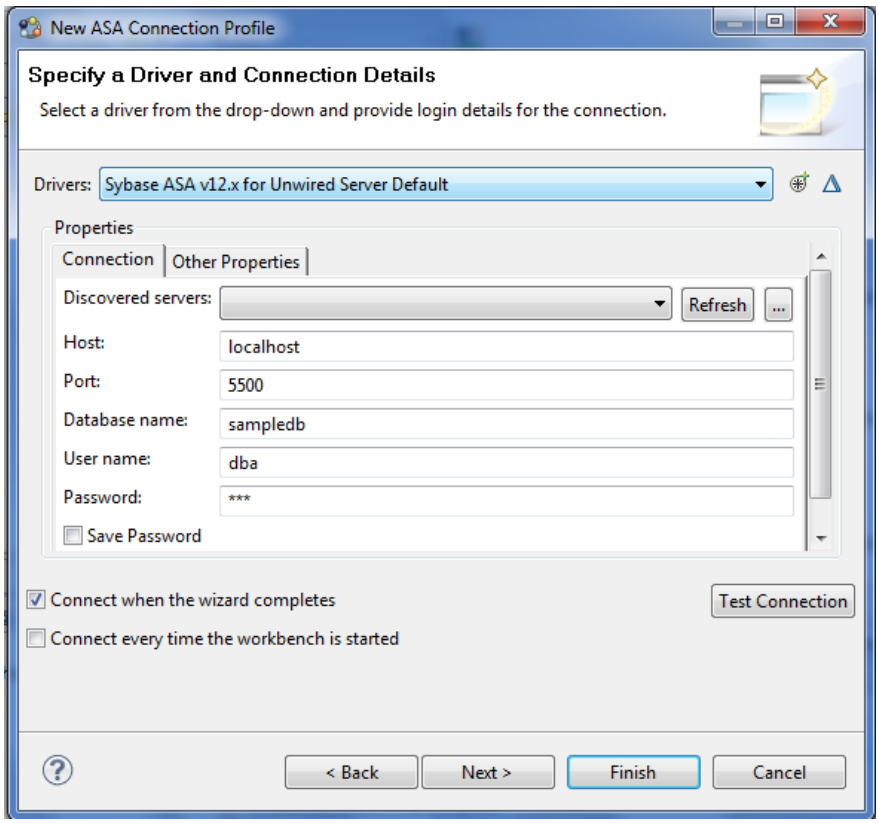
Using MAKit in SAP Mobile Platform Hybrid Web Container

This section discusses how to create charts using MAKit HTML 5 within a SAP Mobile Platform Hybrid Web Container. The example creates an SAP Mobile Platform Hybrid App, which shows several charts using data from an SAP Mobile Server. First, we will create several tables in the default database ("sampledb") in which to store the data. Next, we will create mobile business objects (MBOs) to map to the data table from the database. Finally, we will create different types of charts to show the MBOs.

Creating Database Tables Using SAP Mobile WorkSpace

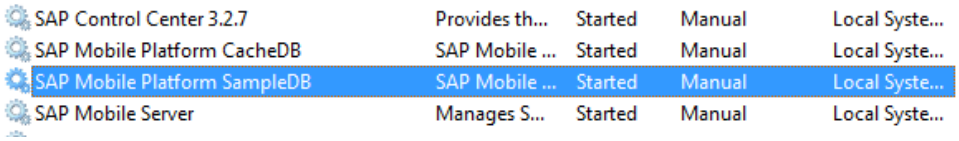
Creating a Sample Connection Profile

1. In Enterprise Explorer, right-click the **Database Connection** folder, and select **New** to open the wizard page.
2. Select **Sybase ASA**, enter `sampledb` as the name, then click **Next**.
3. Select **Sybase ASA v12.0 for Mobile Server Default** as the driver and keep all the default settings.



4. Click **Test Connection** to verify the connection works.

   If the test connection fails, verify that the sampledb service is started: In Windows, select **Start > Control Panel > Administrative Tools > Services**.



5. Click **Finish** to complete.

Creating Sample Tables

Create a table named "RevenueByY" that has this definition:

Primary Key	Column Name	Datatype	Nullable	Default
☐	product	nvarchar(20)	☐	
☐	year	double	☐	
☐	revenue	double	☐	

1. Add sample data to the table:

No.	product [NVARCHAR(20)]	year [DOUBLE]	revenue [DOUBLE]
1	laptop	2005.0	2000.0
2	desktop	2005.0	1500.0
3	tablet	2005.0	1200.0
4	desktop	2006.0	1700.0
5	laptop	2006.0	2300.0
6	tablet	2006.0	1500.0
7	laptop	2007.0	2500.0
8	desktop	2007.0	1800.0
9	tablet	2007.0	1500.0
10	laptop	2008.0	2800.0
11	desktop	2008.0	2000.0
12	tablet	2008.0	1600.0
13	laptop	2009.0	2900.0
14	desktop	2009.0	2100.0
15	tablet	2009.0	2000.0
16	laptop	2010.0	2700.0
17	desktop	2010.0	1900.0
18	tablet	2010.0	2500.0

This shows revenue for 6 years (2005 to 2010). For each year, there are three products (laptop, desktop, tablet).

2. To add all data for table RevenueByY, execute the "SQL Statements for RevenueByY Table," on page 102.
3. To create the two additional tables needed for semantic zooming chart, execute the "SQL Statements for RevenueByQ Table" on page 103 and the "SQL Statements for RevenueByM Table," on page 105.
4. To create the additional table needed for the drilldown chart, execute the "SQL Statements for EmpIncome Table" on page 113.

Creating a New Mobile Application

1. Select **File** > **New** > **Mobile Application Project**.
2. Enter `demo1` as the name.
3. Click **Finish**.

Creating New MBOs

Creating a New MBO for the RevenueByY Table

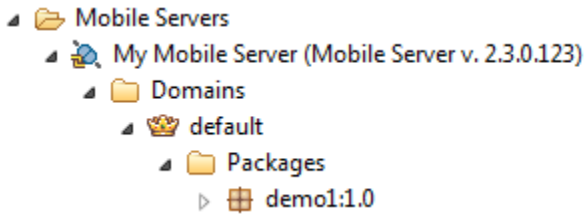
1. From WorkSpace Navigator, right-click the **demo1** folder, then select **Open in Diagram Editor**.
2. Drag the `RevenueByY` table to the Mobile Application Diagram painter.
3. Enter `RevenueByY` as the MBO name.
4. Unselect "create."
5. Click **OK.**

Deploying the MBO

1. From the WorkSpace Navigator, right-click **Mobile Business Objects**, then select **Deploy Project**.
2. Select **Update** as the deployment mode, keep the default name (`demo1`) as the package name, then click **Next**.
3. Review the MBO list and click **Next** to continue.
4. Click **Next** again.

5. Choose **My Mobile Server** as the target server, then click **Next** to continue.
6. Click **Next** again.
7. In the Server Connection column, choose **sampleDB**.
8. Click **Finish**.

Note: You can verify whether the MBO was deployed successfully by going to the Enterprise Explorer and selecting **My Mobile Server > Domains > default > Packages**. "demo1:1.0" indicates that the MBO has been successfully deployed.



Creating a New Hybrid App

1. Select **File** > **New** > **Hybrid App Designer**.
2. Choose **demo1 [Version 1.0]** as the parent folder, specify the Hybrid App file name as `MyWFDemo.xbw`, then click **Next**.
3. Select **Can be started, on demand from the client**, and leave the other options unselected.
4. Click **Finish**.

Adding an MBO to the Hybrid App

1. Go to the Flow Design page by clicking **Flow Design** tab.
2. In WorkSpace Navigator, select the **demo1** > **Mobile Business Object** folder to show all MBOs.
3. Drag the **RevenueByY** MBO to the workflow form. Two screens are automatically created—RevenueByY, which contains a listview that shows the MBO data, and RevenueByYDetail.
4. Link the start-up screen to the RevenueByY screen, so there is a menu item ("Open Screen RevenueByY") on the start-up screen to be created. At runtime, when this menu item is clicked, the workflow transits to this page.

Adding an Action to Retrieve Data from the MBO

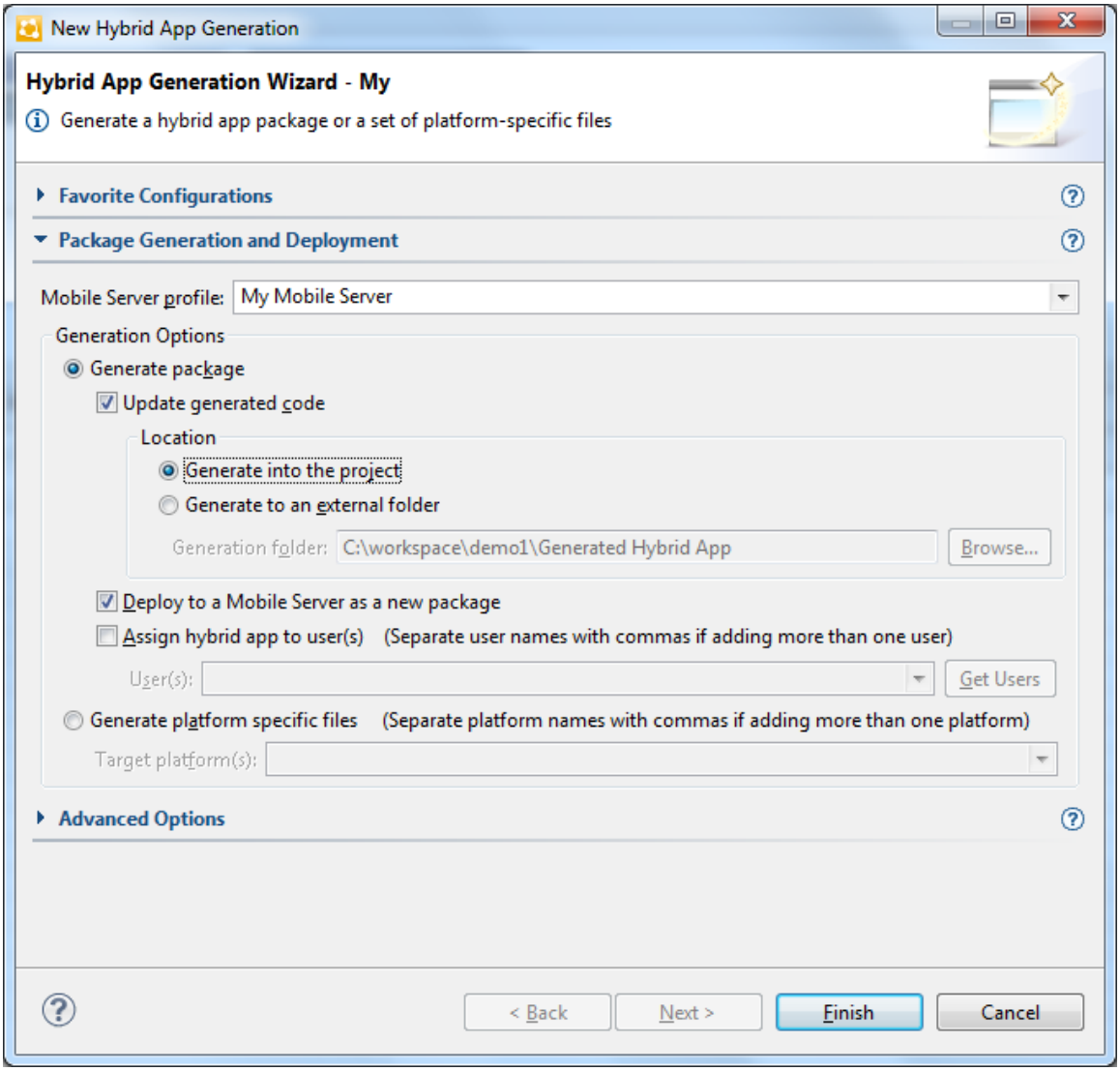
1. In the Flow Design tab, double-click **RevenueByY**.
2. Delete the Submit menu item, then add a new menu item named **Get Data**.
3. On the **Get Data** property page:
   - Change the type from "Save Screen" to "Online Request."
   - Verify that the MBO binding to the RevenueByY MBO exists. If it is not, click **Search** button to bind it. In the dialog, keep the default values, click **Search**, and then choose **RevenueByY**. Click **OK** to finish.
   - Select **Invoke object query** and unselect "Generate old values keys."
   - In Object Query, select **findAll** to retrieve all data rows.

Name: Get Data
Key: Get_Data
☑ Default
Type: Online Request
Details
Mobile business object: demo1/Revenue1 Search... Clear
○ Invoke parent update
○ Invoke operation
Operation:
◉ Invoke object query
Object query: findAll
☐ Generate old value keys
☑ Show credential screen on authentication failure
Submit error message:

4. Save this screen.

Deploying the Hybrid App

1. Right-click the **Screen Design** page, and select **Generate Hybrid App.**
2. Leave the default values as shown:



You can assign this Hybrid App to an existing user on your Mobile Server.

3. Click **Finish**.

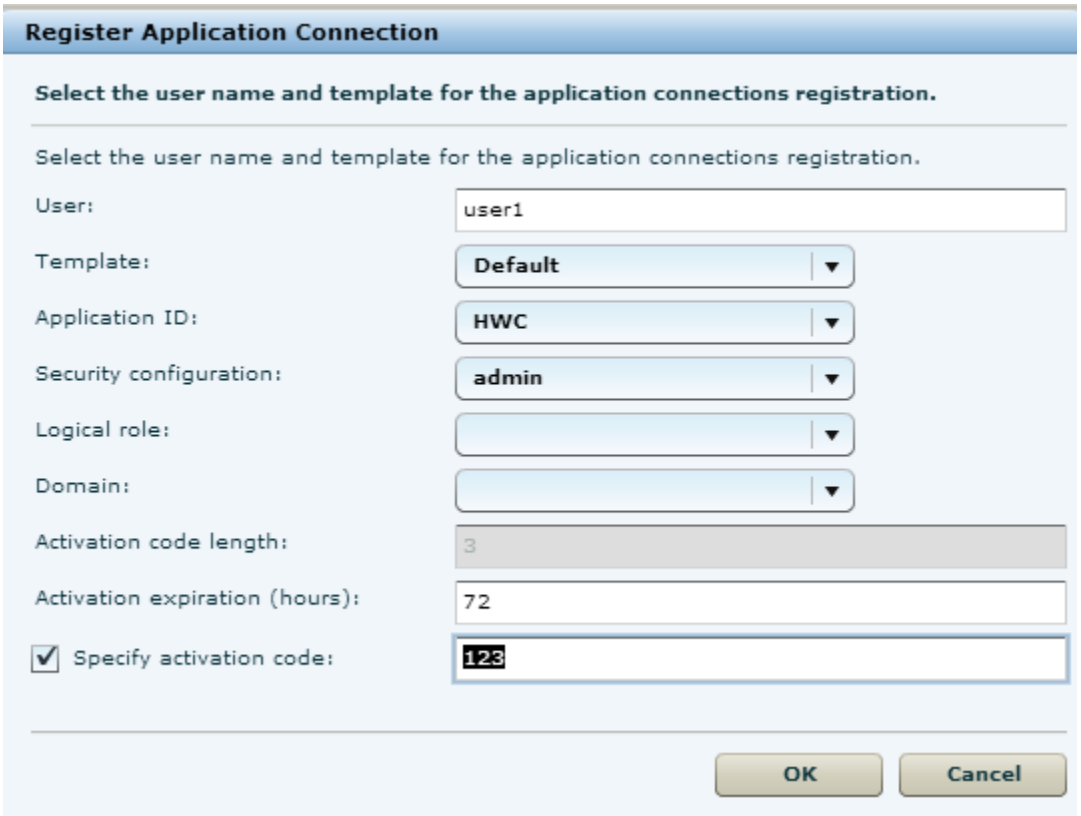
WorkSpace deploys the Hybrid App to the Mobile Server. Verify the deployment by selecting the **Enterprise Explorer** tab > **Mobile Servers** > **My Mobile Server**. In the `Hybrid Apps` folder, you should see MyWFDemo:1—the newly deployed Hybrid App.

Defining an Application Connection

You must assign your Hybrid App to an application connection so that the container can access your Hybrid App from a device.

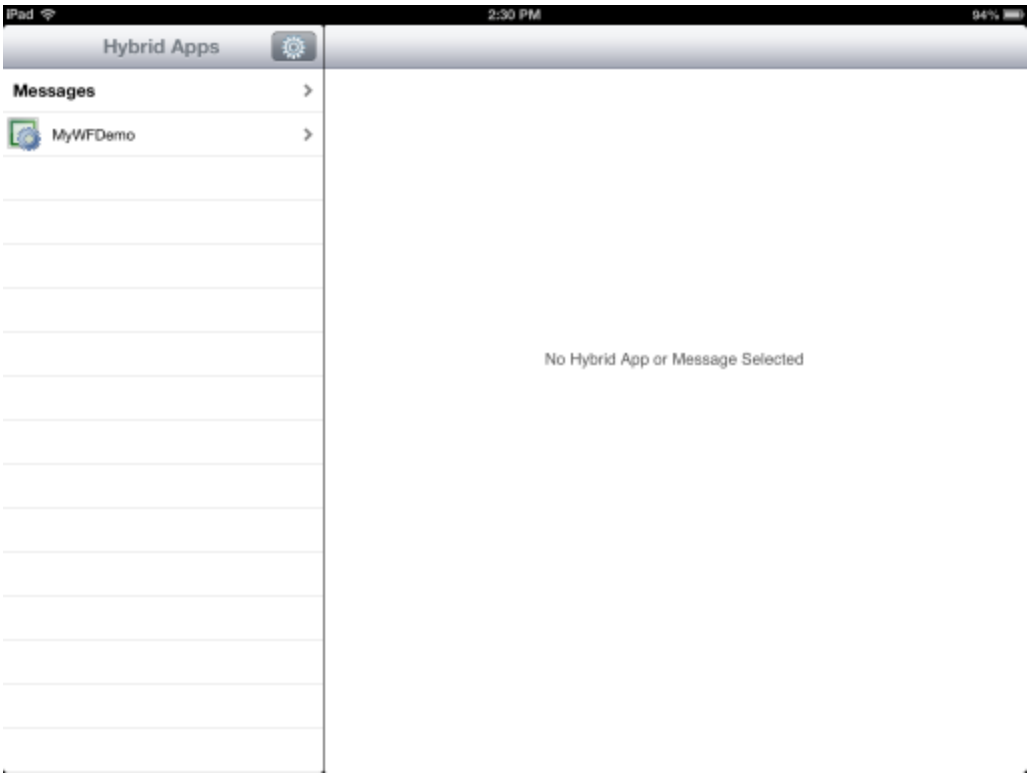
1. Log in to SAP Control Center. The default user name and password is “supAdmin” and “s3pAdmin”.
2. In the left pane, click **Applications.** In the Application Connection tab, click **register**, and enter this information:
   - User Name – `user1` or any user name you want
   - Template – default
   - Application ID: HWC
   - Security configuration: admin
   - Logical role: default
   - Domain : default
   - Activation code length: 3

- Activation expiration (hours): 72
- Select **Specify activation code**, and set it to 123 or any string you want

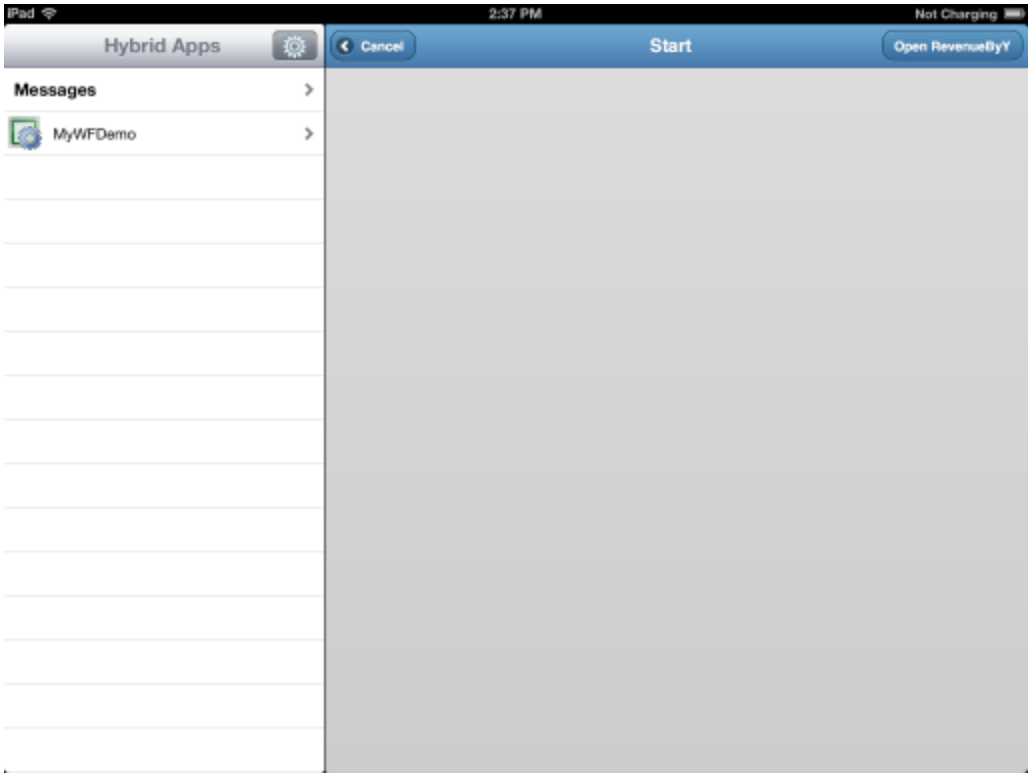


3. In the left pane under **Hybrid Apps**, click the newly deployed Hybrid App. In the Application Connection tab, click **Assign**, search for and select the user you just defined, then click **OK** to assign it to the Hybrid App.

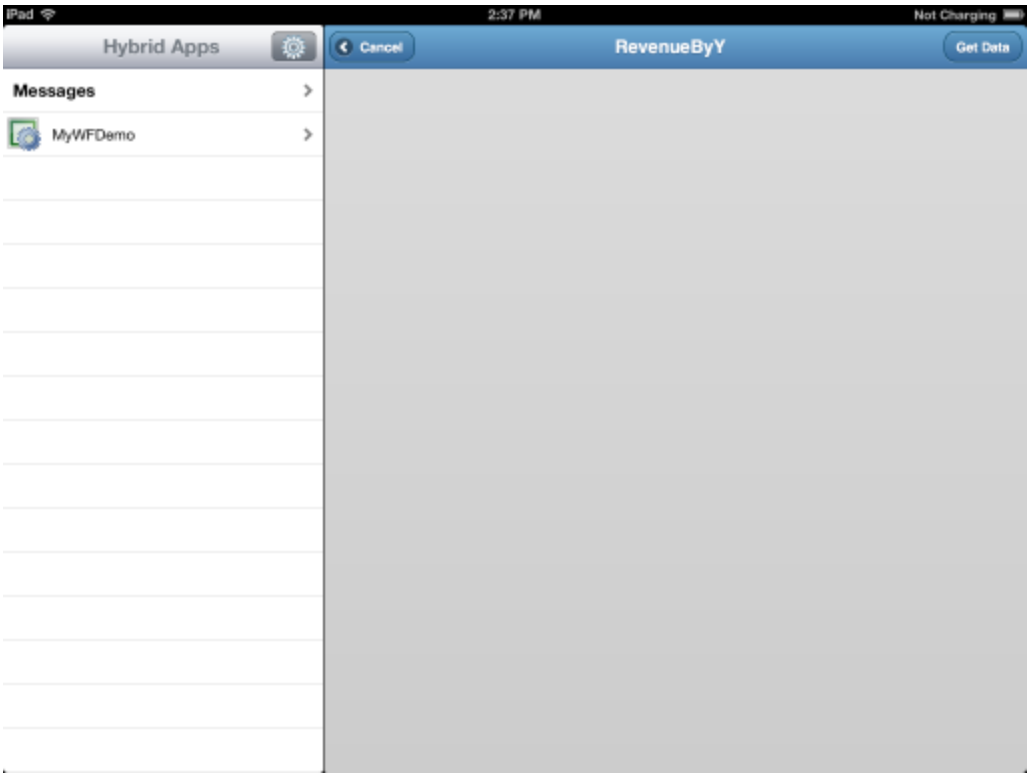
   You should see a Hybrid App named “MyWFDemo” on your iOS/Android device.



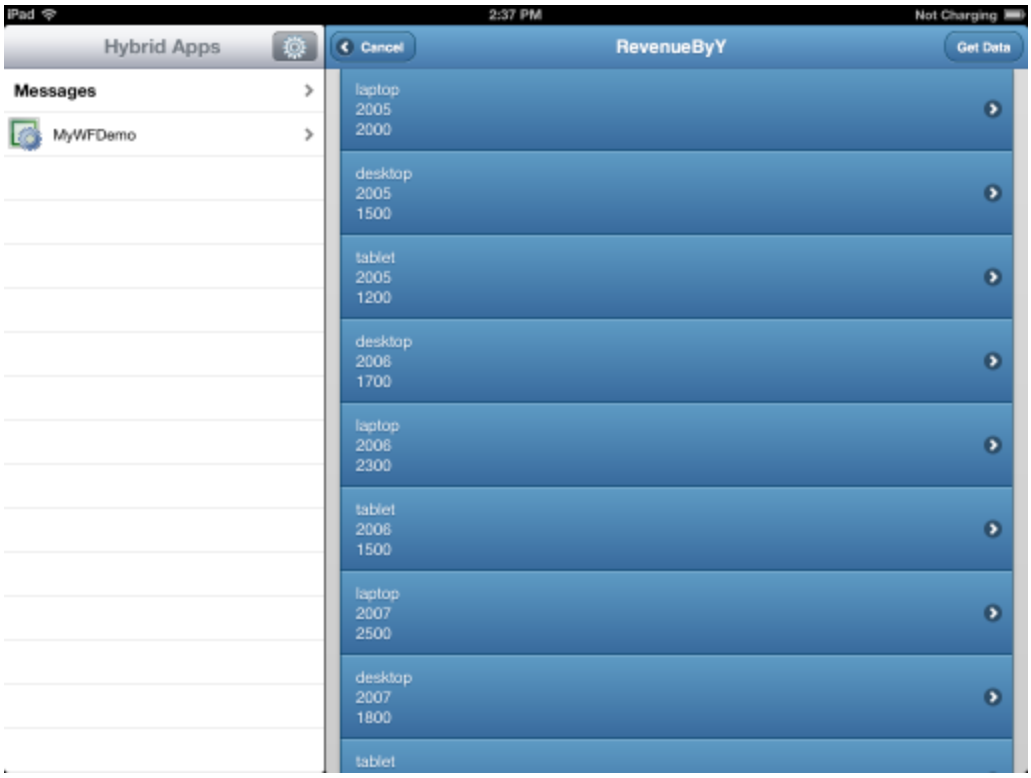
Click the Hybrid App to see a screen with the “Open RevenueByY” menu. Tap the menu:



You see the next screen with a “Get Data” menu item. Tap **Get Data**:



Click **Get Data** to see the data in the listview:



Adding a Simple MAKit Chart to the Hybrid App

Adding MAKit-Related Files to the Project

We will now add some custom JavaScript files to the project. We will designate a target folder with the path `demo1\Generated Hybrid App\MyWFDemo\html` and call it <*Target Root*>.

1. Add the JavaScript file `MyLayout.js` to the `<Target Root>/js` folder. You can assign any name to this file; we will later add functions that define the screen layout and arrange charts. Add this code to the JavaScript file:

```
//used for defining the layout of normal chart on the screen

arrangeCharts = function(cols, rows, contentDivElem, refresh)

{

         var width = window.innerWidth;

        var height = window.innerHeight;

        var contents = contentDivElem;

        if (contents == "undefined")

                contents = "contents";


        var elem = document.getElementById(contents);

        elem.style.width = width + "px";

        elem.style.height = (height  - 40) + "px";


        var len = elem.children.length;

        var childElem;

        for (var i = 0; i < len; i++)

        {

                childElem = elem.children[i];

                childElem.style.width = (95 / cols)  + "%";

                childElem.style.height = (95 / rows) + "%";


                if (refresh != undefined && refresh)

                {

                        if (childElem.chart != undefined)
```

```
                                childElem.chart.refresh();

                }

        }

};


//used for defining the layout of drilldown chart

arrangeDrillDownCharts = function(cols, rows, contentDivElem, refresh)

{

        var offset = 0;

        var toolbars = document.getElementsByClassName("toolbar");

        if (toolbars != undefined && toolbars.length > 0)

                offset = toolbars[0].offsetHeight;


        var width = window.innerWidth;

        var height = window.innerHeight;


        var contents = contentDivElem;

        if (contents == "undefined")

                contents = "contents"


        var elem = document.getElementById(contents);

        elem.style.width = width + "px";

        elem.style.height = (height - offset - 40 - 40) + "px";


        childElem = elem.children[0];

        childElem.style.width = (95/cols) + "%";

        childElem.style.height = (95/rows) + "%";


        if (refresh != undefined && refresh)

        {

                if (childElem.drilldowngroup != undefined)
```

```
                        childElem.drilldowngroup.refresh();

        }

};
```

Note: You can customize this function for your app.

2. Add another JavaScript file, named `MyQuery.js`, to the `<Target Root>/js` folder. Leave this file empty for now; we will later add code to it.
3. Create a subfolder named `data` under the `<Target Root>` folder.

Adding the Metadata Files for the Chart

MAKit uses metadata to define charts, so you must create all the metadata files for your app, and add them to the Hybrid App project under the `<Target Root>/data` folder.

In this sample, we will show a column chart. Create a metadata file using the following code. Name it `bi_chart_bar.xml`, and save it in the `<Target Root>/data` folder.

```
<Chart Name="gr_1" ChartType="column" Query="productRevenueByYear" Title="Revenue By Year">

        <Category Column="year"/>

        <Series Column="product"/>

        <Values>

                <Value Expression="revenue" Format="currency"/>

        </Values>

</Chart>
```

For more information about the metadata definition, see “Metadata File” on page 37.

Adding a New HTML Screen Page

1. On the Flow Design page, select **Screen** from the `Screens` folder, and add it to the workflow. Name the new screen `ColumnChart`.
2. Select the **GoTo** control from the palette, then click+drag from **RevenueByY** to **ColumnChart** to link the RevenueByY screen to the newly created screen. This adds a menu item to the RevenueByY screen.
3. On the ColumnChart design page, select the **HtmlView** control from the toolbox, then click inside the ColumnChart screen.
4. Click the **HtmlView** control, and on the property tab, create a new key (“ChartKey” or any name) for this view by clicking **New Key**. Set the value for this key as:

```
<div id="contents">

<div id="columnChart_1" class="chart"></div>

</div>
```

Creating a MAKit Chart

The generated code for the Hybrid App includes a file named `Custom.js`. You can add your code to this file to customize your application. For our sample, add the code in the **hwc.customAfterShowScreen** function to create the chart:

```
hwc.customAfterShowScreen(screenToShow, screenToHide) {
```

```
    if (screenToShow === "ColumnChart") {

            arrangeCharts(1, 1, "contents");//only one chart on then screen

            showFullBiChart("columnChart_1", "data/bi_chart_bar.xml");

    }

}
```

Note: "ColumnChart" is the key of the screen; "contents" is the ID defined above for a placeholder; "columnChart_1" is the ID of the chart defined above; "data/bi_chart_bar.xml" is the metadata file.

The definition of the **showFullBiChart** function is:

```
function showFullBiChart(htmlElementID, metadataFile) {

    var elem = document.getElementById(htmlElementID);

    var chart = new $MA.Chart(elem.id, true);

    chart.load(metadataFile);

}
```

Getting Data from SAP Mobile Server

As MAKit is a query-based solution, the application developer must implement a query that provides data to a MAKit chart. Hybrid App developers must add the code that implements the query. In MAKit HTML5, a specific global function—**executeMAQuery**— is the only bridge or interface between MAKit and the hosting JavaScript. MAKit calls this function, which returns a "dataTable" to MAKit, anytime it needs data. You can define the function in a separate `.js` file or in the `custom.js` file. For our sample, define it in the `MyQuery.js` file:

```
function executeMAQuery(chartObj, name, args){

    var dataTable;

    if (name === "productRevenueByYear") {

            dataTable = getProductRevenueByYearData();

            chartObj.setDataTable(dataTable);

    }

}
```

Note: " productRevenueByYear " is the query name defined in the metadata for the chart.

The **getProductRevenueByYearData** function gets the data for this chart. To retrieve data from the SAP Mobile Server, **getProductRevenueByYearData** must call some additional functions from the SAP Mobile Platform container. The code for retrieving data should be similar to:

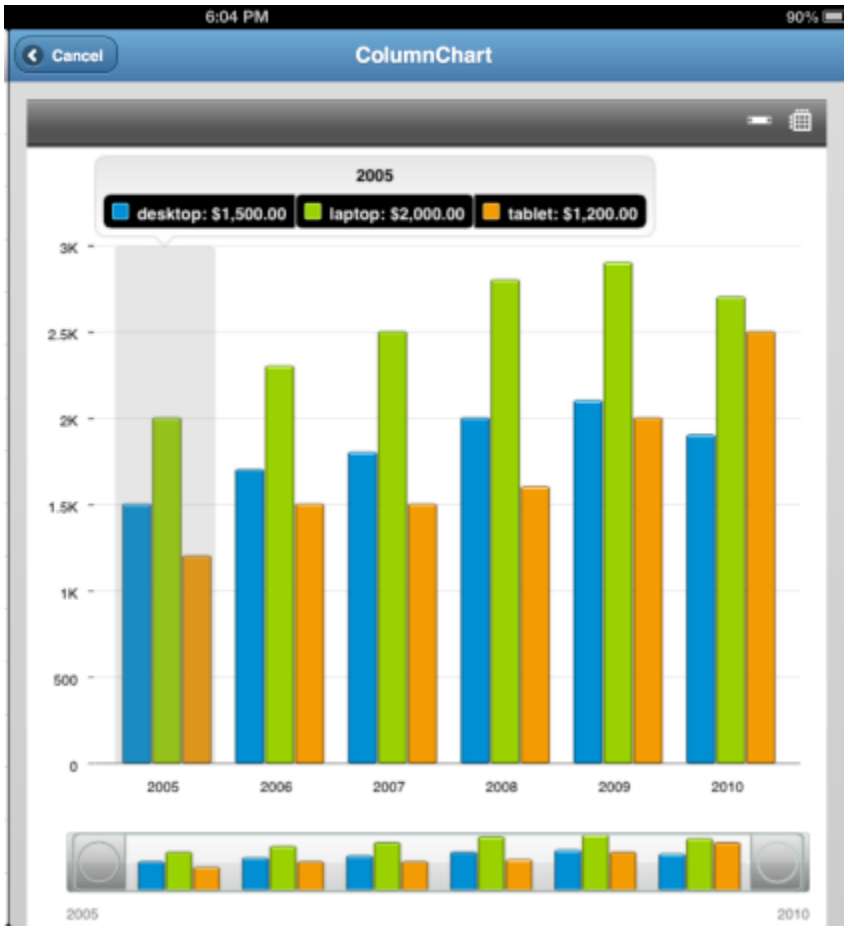
```
function getProductRevenueByYearData()

{

    //Define the table columns for the chart

    var dataTable = new $MA.DataTable();

    dataTable.addColumn("product", "string");
```

```
    dataTable.addColumn("year", "number");

    dataTable.addColumn("revenue", "number");


    //Get data from SUP MBO -- RevenueByY

    var message = getCurrentMessageValueCollection();

    var itemList = message.getData("RevenueByY");

    var items = itemList.getValue();

    var noOfItems = items.length;

    if (noOfItems > 0) {

            var i = 0;

            while (i < noOfItems) {

                    var theItems = items[i];

                    //Get column “year” from the MBO

                    var yearObj=theItems.getData("RevenueByY_year_attribKey");

                    var year = yearObj.getValue();

                    //Get column “product” from the MBO

                    var prodObj=theItems.getData("RevenueByY_product_attribKey");

                    var prod = prodObj.getValue();

                    //Get column “revenue” from the MBO

                    var amountObj=theItems.getData("RevenueByY_revenue_attribKey");

                    var amount = amountObj.getValue();

                     //Set the a row of data to the table

                    dataTable.addRow([prod, year, amount] );

                    i = i + 1;

            }

    }

    else {

            alert("No data is available.");

    }

    return dataTable;

}
```

Deploying the Hybrid App

1. Right-click the **Flow Design** or **Screen Design** page and select **Generate Hybrid App**.
2. Keep the default settings.
3. Click **Finish**.

At this point, you should have a Hybrid App on your SAP Mobile Server that shows a column chart.



Creating a Semantic Zooming Chart

Creating Sample Data Tables

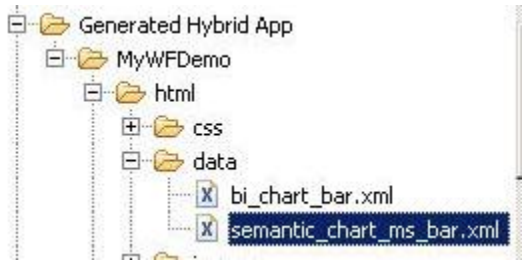
Zooming charts require two additional data tables for the Quarter and Month zooming levels. Check your database to verify the tables for semantic zooming chart exist. If they do not, create them by executing "SQL Statements for RevenueByQ Table" on page 103 and "SQL Statements for RevenueByM Table" on page 105.

Creating MBOs for the Two Tables

See "Creating New MBOs" on page 59. Use "RevenueByQ" as the MBO name for the `RevenueByQ` table. Use "RevenueByM" as the MBO name for the `RevenueByM` table. Then deploy the MBOs to SAP Mobile Server.

Adding Metadata for the Semantic Zooming Chart

Create a new XML file and name it `semantic_chart_ms_bar.xml`. For this sample, add the metadata file to the `data` folder:



The metadata for the semantic zooming chart is defined in `semantic_chart_ms_bar.xml`, and must look like this:

```
<Chart Name="chart_1" ChartType="column" Query="productRevenueByYear" Title="Revenue">

    <Category Column="year"/>

    <Series Column="product"/>

    <Values>

            <Value Expression="revenue" Format="currency"/>

    </Values>

    <ZoomingLevels>

            <ZoomingLevel Query="productRevenueByYear" CategoryColumn="year" Title="Year"/>

            <ZoomingLevel Query="productRevenueByQuarter" CategoryColumn="quarter" Title="Quarter"/>

            <ZoomingLevel Query="productRevenueByMonth" CategoryColumn="month" Title="Month"/>

    </ZoomingLevels>

</Chart>
```

For more information about the metadata definition, see “Metadata File” on page 37.

Adding New HTML Screen Pages

1. On the Flow Design page, select **Screen** from the `Screens` folder, and add it to the workflow. Name the new screen `ZoomingChart`.
2. Select the **GoTo** control from the palette, then click+drag from **RevenueByY** to **ZoomingChart** to link the RevenueByY screen to the newly created screen. This adds a menu item to the RevenueByY screen.
3. Go to the screen (ZoomingChart) design page and add an HtmlView control to the screen by selecting **HtmlView** from the toolbox, then clicking inside the ZoomingChart screen.
4. Click the **HtmlView** control, and on the property tab, create a new key (“zoomingChartKey” or any name) for this view by clicking **New Key**. Set the value for this key as:

   ```
   <div id="zContents">
       <div id="zChart_1" class="chart"></div>
   </div>
   ```

5. Create another two screens for data retrieval. To bind the MBO to the workflow (or its screen), create one screen for each MBO—one for the RevenueByQ MBO, and one for the RevenueByM MBO.

- In WorkSpace Navigator, select **demo1 > Mobile Business Object** to show all MBOs.
- Drag the **RevenueByQ** MBO to the workflow form. Two screens are automatically created—RevenueByQ and RevenueByQDetail. RevenueByQ contains a listview that shows the data from the RevenueByQ MBO.
- Drag the **RevenueByM** MBO to the workflow form. Two screens are automatically created—RevenueByM and RevenueByMDetail. RevenueByM contains a listview that shows the data from the RevenueByM MBO.

6. Add one action menu item for retrieving the data from RevenueByQ.
   - On the Flow Design tab, double-click **ZoomingChart**.
   - Delete the "Submit" menu item if it exists. Then add a new menu item named `GetQData`.
   - On the GetQData property page:
     - Change the type from "Save Screen" to "Online Request."
     - Set the Mobile Business Object to `demo1/RevenueByQ`.
     - Select **Invoke object query** and unselect "Generate old values keys."
     - For Object Query, select **findAll** to retrieve all data rows.
     - Save the screen.
7. Add another action menu item for retrieving the data from RevenueByM.
   - On the Flow Design tab, double-click **ZoomingChart**.
   - Delete the "Submit" menu item if it exists. Then add a new menu item named `GetMData`.
   - On the GetMData property page:
     - Change the type from "Save Screen" to "Online Request."
     - Set the Mobile Business Object to `demo1/RevenueByM`.
     - Select **Invoke object query** and unselect "Generate old values keys."
     - For Object Query, select **findAll** to retrieve all data rows.
     - Save the screen.

Creating the MAKit Semantic Chart

The generated code for the Hybrid App includes a `Custom.js` file. You can add your code to this file to customize your application. For our sample, add the following code to the **hwc.customAfterShowScreen** function to create the chart; leave the rest of this function untouched. The code should look like:

```
if (screenToShow === "ZoomingChart") {

    arrangeCharts(1, 1, "zContents");    //only one chart on then screen

    showFullZoomingChart("zChart_1", "data/semantic_chart_ms_bar.xml");

}
```

Note: "ZoomingChart" is the key of the screen; "zContents" is the ID defined above for a placeholder; "zChart_1" is the ID of the chart defined above; "data/ semantic_chart_ms_bar.xml" is the metadata file.

The definition of the **showFullZoomingChart** function is:

```
function showFullZoomingChart (htmlElementID, metadataFile) {

    var elem = document.getElementById(htmlElementID);

    var chart = new $MA.Chart(elem.id, true);

chart.load(metadataFile);

}
```

Getting Data from the SAP Mobile Server

A semantic zooming chart needs several queries to get the data for different zooming levels. Whenever the chart needs data, it calls the **executeMAQuery** function and passes the query name to it. Hybrid App developers must add code to implement the queries.

In the SAP Mobile Platform Hybrid Web container, data retrieval is asynchronous, which means that when you request data from the SAP Mobile Server, the application does not wait for the return of the data, but continues executing code. When the data is ready, the application calls the callback function **hwc.customAfterDataReceived**.

Add your new query handling code to **executeMAQuery** as follows and keep the existing code in this function untouched:

```
var yDataOK;                            //if the data for Year is ready

var qDataOK = false;                    //if the data for Quarter is ready

var mDataOK = false;                    //if the data for Month is ready

var myWorkflowMessage;          //the incoming message

var ChartObject;                //the chart object

var QueryName;                  // the query name

var Args;                       //possible argument

function executeMAQuery(chartObj, name, args){

    var dataTable;

if (name === "productRevenueByYear") {          //for the simple chart sample

        dataTable = getProductRevenueByYearData();

        chartObj.setDataTable(dataTable);

}

else if (name === "productRevenueByQuarter") {

        if (qDataOK == false) {

                ChartObject = chartObj;

                QueryName = name;

                Args = args;

                retrieveRevenueByQData();       //send request to get data(1)

        }

        else {

                //get the data and pass it to the chart

                var dataTable = getProductRevenueByQuarterData(args); //get data (2)

                chartObj.setDataTable(dataTable);

                chartObj.refresh();

                QueryName = null;
```

```
                Args = null;

        }

}

else if (name === "productRevenueByMonth") {

        if (mDataOK == false) {

                ChartObject = chartObj;

                QueryName = name;

                Args = args;

                retrieveRevenueByMData();        //send request to get data(3)

        }

        else {

                var dataTable = getProductRevenueByMonthData(args);      //get data (4)

                chartObj.setDataTable(dataTable);

                chartObj.refresh();

QueryName = null;

                Args = null;

        }

}

else {



}



}
```

Note: "productRevenueByQuarter" and “productRevenueByMonth” are the query names defined in the metadata for quarter and month levels.

As the query code shows, if the quarter-level data is ready, the **getProductRevenueByQuarterData** function is called to retrieve the data; if the month-level data is ready, the **getProductRevenueByMonthData** function is called to retrieve the data. These functions are defined as:

```
function getProductRevenueByQuarterData(args)

{

    var dataTable = new $MA.DataTable();

    dataTable.addColumn("product", "string");

    dataTable.addColumn("year", "number");
```

```
dataTable.addColumn("quarter", "number");

dataTable.addColumn("revenue", "number");


//Get data from SUP MBO -- RevenueByQ

var message = getCurrentMessageValueCollection();

var itemList = message.getData("RevenueByQ");

var items = itemList.getValue();

var noOfItems = items.length;

if (noOfItems > 0) {

    var i = 0;

     while (i < noOfItems) {

             var theItems = items[i];

            var yearObj=theItems.getData("RevenueByQ_year_attribKey");

            var year = yearObj.getValue();


            var prodObj=theItems.getData("RevenueByQ_product_attribKey");

            var prod = prodObj.getValue();


            var qObj=theItems.getData("RevenueByQ_quarter_attribKey");

            var quarter = qObj.getValue();


            var amountObj=theItems.getData("RevenueByQ_revenue_attribKey");

            var amount = amountObj.getValue();


            dataTable.addRow([prod, year, quarter, amount] );

            i = i + 1;

     }

}

else {

    alert("No data is available.");

}
```

```
    return dataTable;
}

function getProductRevenueByMonthData(args)
{
    var dataTable = new $MA.DataTable();
    dataTable.addColumn("product", "string");
    dataTable.addColumn("year", "number");
    dataTable.addColumn("quarter", "number");
    dataTable.addColumn("month", "number");
    dataTable.addColumn("revenue", "number");

    //Get data from SUP MBO -- RevenueByM
    var message = getCurrentMessageValueCollection();
    var itemList = message.getData("RevenueByM");
    var items = itemList.getValue();
    var noOfItems = items.length;
    if (noOfItems > 0) {
        var i = 0;
         while (i < noOfItems) {
                var theItems = items[i];
                var yearObj=theItems.getData("RevenueByM_year_attribKey");
                var year = yearObj.getValue();

                var prodObj=theItems.getData("RevenueByM_product_attribKey");
                var prod = prodObj.getValue();

                var qObj=theItems.getData("RevenueByM_quarter_attribKey");
                var quarter = qObj.getValue();

                var mObj=theItems.getData("RevenueByM_month_attribKey");
```

```
                var month = mObj.getValue();


                var amountObj=theItems.getData("RevenueByM_revenue_attribKey");

                var amount = amountObj.getValue();


                dataTable.addRow([prod, year, quarter, month, amount] );

                i = i + 1;

         }

    }

    else {

        alert("No data is available.");

    }

    return dataTable;

}
```

This code shows that if the quarter-level or month-level data is not ready, **retrieveRevenueByQData** or **RetrieveRevenueByMData** is called to request the data. These two functions are defined as:

```
//Send request to get Quarter data

function retrieveRevenueByQData() {

var rmiKeys = [];

    var rmiKeyTypes = [];

    var rmiInputOnlyKeys = [];

    var rmiInputOnlyKeyTypes = [];

    var workflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('ZoomingChart', 'GetQData',
rmiKeys, rmiKeyTypes);

    var inputOnlyWorkflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('ZoomingChart',
'GetQData', rmiInputOnlyKeys, rmiInputOnlyKeyTypes);



hwc.doOnlineRequest('ZoomingChart', 'GetQData', 60, 0, '', null, workflowMessageToSend,
inputOnlyWorkflowMessageToSend.serializeToString());

}



//Send request to get Month data

function retrieveRevenueByMData() {
```

```
 var rmiKeys = [];

 var rmiKeyTypes = [];

 var rmiInputOnlyKeys = [];

 var rmiInputOnlyKeyTypes = [];

 var workflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('ZoomingChart', 'GetMData',
rmiKeys, rmiKeyTypes);

var inputOnlyWorkflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('ZoomingChart',
'GetMData', rmiInputOnlyKeys, rmiInputOnlyKeyTypes);

hwc.doOnlineRequest('ZoomingChart', 'GetMData', 60, 0, '', null, workflowMessageToSend,
inputOnlyWorkflowMessageToSend.serializeToString());

}
```

Data communication from SAP Mobile Server is asynchronous. When the data is ready, the system calls the **hwc.customAfterDataReceived** function, which is defined in `Custom.js`. Add code to this function to retrieve the data and pass it to the MAKit chart:

```
hwc.customAfterDataReceived = function(incomingDataMessage) {

        myWorkflowMessage = incomingDataMessage;

        try

        {

                var valueCols = incomingDataMessage.getValues();

                //check data for year

                var yItemList = valueCols.getData("RevenueByY");

                if (yItemList != undefined) {

                        yDataOK = true;

                }

                else {

                        yDataOK = false;

                }

                //check data for quarter

                var qItemList = valueCols.getData("RevenueByQ");

                if (qItemList != undefined) {

                        qDataOK = true;

                }

                else {
```

```
                qDataOK = false;
        }
        //check data for month
        var mItemList = valueCols.getData("RevenueByM");
        if (mItemList != undefined) {
                mDataOK = true;
        }
        else {
                mDataOK = false;
        }

        //send the specific data back to MAKIT Chart
        if (QueryName === "productRevenueByYear") {
                var dataTable = getProductRevenueByYearData();
                ChartObject.setDataTable(dataTable);
                dataTable.refresh();
        }
        else if (QueryName === "productRevenueByQuarter") {
                var dataTable = getProductRevenueByQuarterData(Args)
                ChartObject.setDataTable(dataTable);
                dataTable.refresh();
        }
        else if (QueryName === "productRevenueByMonth") {
                var dataTable = getProductRevenueByMonthData(Args);
                ChartObject.setDataTable(dataTable);
                dataTable.refresh();
        }
        else {

        }
```

```
                QueryName = null;

                Args = null;



        }

        catch (err)

        {

                QueryName = null;

                Args = null;

        }

}
```

Deploying the Hybrid App

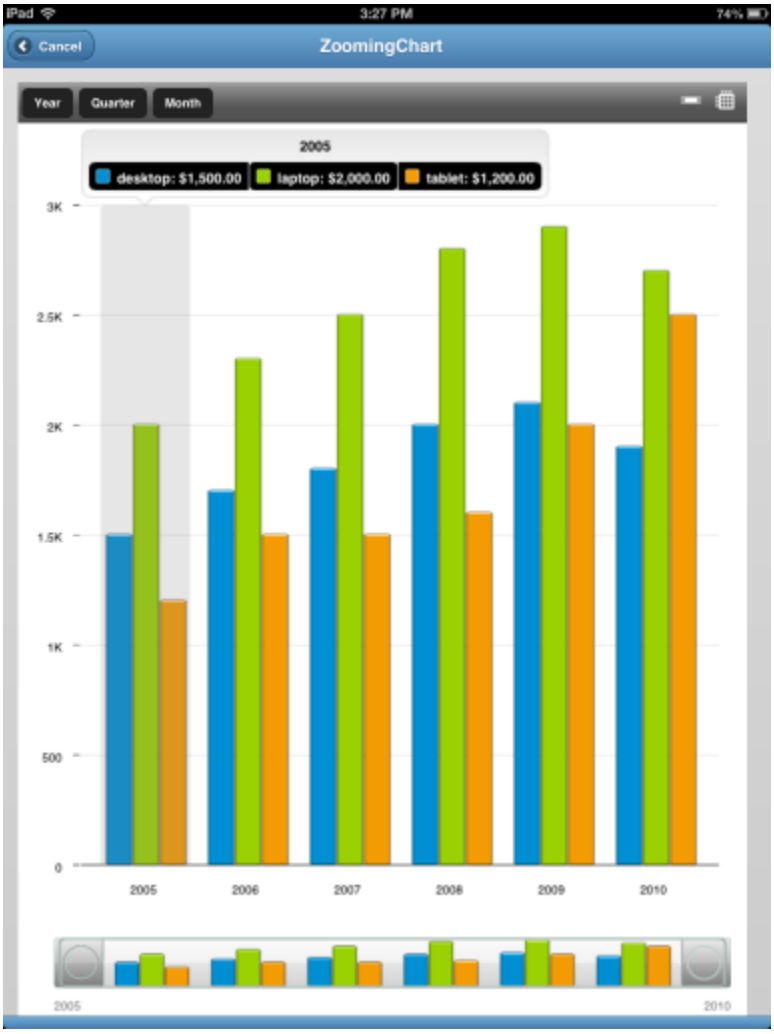
1. Right-click either the **Flow Design** or **Screen Design** page and select **Generate Hybrid App**.
2. Keep the default settings.
3. Click **Finish**.

You should see a Hybrid App on the SAP Mobile Server that shows a semantic zooming chart.

Note: If the app transits to the ZoomingChart screen, you see two menu items—GetQData and GetMData. We created them for MBO binding and data retrieval, but we do not want them to appear. To hide them, add this code to the **hwc.customBeforeShowScreen** function in `Custom.js`:

```
hwc.customBeforeShowScreen(screenToShow, screenToHide) {

    if (screenToShow === "ZoomingChart") {

            hwc.removeMenuItem("ZoomingChart", "GetQData");

            hwc.removeMenuItem("ZoomingChart", "GetMData");

    }

    return true;

}
```

The zooming chart is similar to the one shown below. To see the zooming levels, click **Year**, **Quarter**, or **Month**.



Creating a Drilldown Chart

Creating Sample Data Tables

This drilldown chart requires a new table from the sample database. The table name is `EmpIncome`. Check your database to verify that the table exists. If it does not, create it by executing the "SQL Statements for EmpIncome Table," on page 113.

Creating MBOs for a Drilldown Chart

This sample has three levels to drill down and up: Branch, Department, and Employee.

Creating an MBO for Branch

Create an MBO to show total income and bonuses for every branch. We will create two computed columns to sum the salaries and bonus column for a branch.

1. From WorkSpace Navigator, right-click the **demo1** folder, then select **Open in Diagram Editor.**
2. Right-click Mobile Business Object under `demo1` folder. Then select New > Mobile Business Object.
3. Enter `Branch` as the MBO name.
4. Choose **Select a Data Source**. Enter `JDBC` as the datasource type, and `My Sample Database` as the connection profile, then click **Next.**
5. Select **SQL Query Statement**, then enter:

```
SELECT branCode, SUM(income), incomeType, SUM(size) FROM EmpIncome GROUP BY branCode, incomeType ORDER BY branCode, incomeType
```

6. Click **Finish**.

Creating an MBO for Department

Create an MBO to show all departments for specific branch. This MBO takes the branch code as a parameter.

1. Create the MBO:
   - Right-click **Mobile Business Object** under the `demo1` folder. Then select **New > Mobile Business Object**.
   - Enter `dept` as the MBO name.
   - Choose **Select a Data Source**. Enter `JDBC` as the datasource type, and `My Sample Database` as the connection profile, then click **Next.**
   - Select **SQL Query Statement**, then enter:

     ```
     SELECT branCode, deptCode, SUM(income), incomeType,SUM(size) FROM EmpIncome GROUP BY branCode, deptCode, incomeType ORDER BY branCode, deptCode, incomeType
     ```

   - Click **Finish**.
2. Define object queries. A new query is needed to define the result set.
   - In the Mobile Application Diagram screen, select the **dept** MBO.
   - Below the Mobile Application Diagram screen, under the Properties tab, select the **Attributes** subtab.
   - Select the **Object Queries** tab.
   - Click **Add**, and in the new query wizard:
     - Enter `findByBranchCode` as the query name.
     - Click **Add** to add a new parameter.
     - Enter `brCode_param` as the parameter name.
     - Select **INT** as the datatype.
     - Unselect "Nullable."
     - Select **branCode** as the **Mapped to** value.
   - Click **Generate**. The query statement should look like:

     ```
     SELECT x.* FROM dept.x WHERE x.branCode = :brCode_param
     ```

   - Select **Create an index**.
   - Select **Return Multiple Objects**.
   - Click **Finish**.

   Note: Make note of the query parameter name (**brCode_param**) for later use, when you define the screen and menu item.

Creating an MBO for Employee

The third drilldown level is Employee. Create an MBO to show all employees for specific branch and department. This MBO takes the branch code and department code as parameters.

1. Create the MBO:
   - Right-click **Mobile Business Object** under the `demo1` folder. Then select **New > Mobile Business Object**.
   - Enter `Employee` as the MBO name.
   - Choose **Select a Data Source**. Enter `JDBC` as the datasource type, and `My Sample Database` as the connection profile, then click **Next**.
   - Select **SQL Query Statement**, and enter:

     ```
     SELECT branCode, deptCode, empCode, SUM(income), EmpIncome.incomeType, SUM(size) FROM EmpIncome GROUP BY branCode, deptCode, empCode, incomeType ORDER BY branCode, deptCode, empCode, incomeType
     ```

- Click **Finish**.

2. Define object queries:
   - In the Mobile Application Diagram screen, select the **Employee** MBO.
   - Below the Mobile Application Diagram screen, under the Properties tab, select the **Attributes** subtab.
   - Select the **Object Queries** tab.
   - Click **Add**, and in the new query wizard:
     - Enter `findByBranchDeptCode` as the query name.
     - Click **Add** to add a new parameter.
     - Enter `emp_brCode_param` as the parameter name.
     - Select **INT** as the datatype.
     - Unselect "Nullable."
     - Select **branCode** as the **Mapped to** value.
     - Click **Add** to add another new parameter.
     - Enter `emp_deptCode_param` as the parameter name.
     - Select **INT** as the datatype.
     - Unselect "Nullable."
     - Select **deptCode** as the **Mapped to** value.
   - Click **Generate.** The query statement should look like:

```
SELECT x.* FROM Employee x WHERE x.branCode = :emp_brCode_param AND x.deptCode =
:emp_deptCode_param
```

   - Select **Create an index**.
   - Select **Return Multiple Objects**.
   - Click **Finish**.

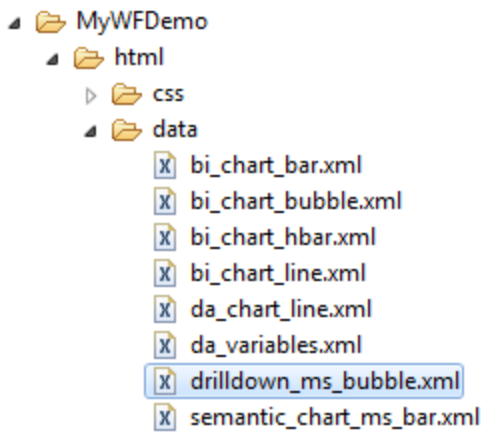
   Note: Make note of the query parameter keys (**emp_brCode_param** and **emp_deptCode_param**) for later use, when you define the screen and menu item.

Deploying the MBOs to the SAP Mobile Platform Server

1. From the WorkSpace Navigator, right-click **Mobile Business Objects**, then select **Deploy Project**.
2. Select either **Update** or **Replace** as the deployment mode, keep the default name (`demo1`) as the package name, then click **Next**.
3. Review the MBO list, and click **Next** to continue.
4. Click **Next** again.
5. Choose **My Mobile Server** as the target server, then click **Next** to continue.
6. Click **Next** again.
7. Under the Server connection column, select **sampledb**.
8. Click **Finish**.

Adding Metadata for the Drilldown Chart

Create a new XML file and name it `drilldown_ms_bubble.xml`. For the sample, add the metadata file to the `data` folder:



The metadata for the drilldown chart is defined in `drilldown_ms_bubble.xml`. The contents of this metadata file must look like:

```
<DrillDownGroup>

    <Chart Name="chart_1" ChartType="bubble" Query="branIncomeDetail" Title="Employee Income Detail By
Branch">

        <Category Column="branCode" DisplayName="Branch Code"/>

        <Series Column="incomeType"/>

    <Values>

        <Value Expression="branIncome" DisplayName="Employee Income Detail"/>

        <Value Expression="bubbleSize"/>

    </Values>

    </Chart>

    <Chart Name="chart_2" ChartType="bubble" Query="deptIncomeDetail" Title="Employee Income Detail By
Department">

        <Category Column="deptCode" DisplayName="Department Code"/>

        <Series Column="incomeType"/>

    <Values>

        <Value Expression="branIncome" DisplayName="Employee Income Detail"/>

        <Value Expression="bubbleSize"/>

    </Values>

    </Chart>

    <Chart Name="chart_3" ChartType="bubble" Query="empIncomeDetail" Title="Individual Employee Income
Detail">

        <Category Column="empCode" DisplayName="Employee Code"/>

        <Series Column="incomeType"/>
```

```
    <Values>

        <Value Expression="branIncome" DisplayName="Employee Income Detail"/>

        <Value Expression="bubbleSize"/>

    </Values>

    </Chart>

</DrillDownGroup>
```

For more information about the metadata definition, see “Metadata File” on page 37.

Adding New HTML Screen Pages

1. On the Flow Design page, select **Screen** from the `Screens` folder, and add it to the workflow. Name the new screen `dwChart`.
2. Select the **GoTo** control from the palette, then click+drag from **RevenueByY** to **dwChart** to link the RevenueByY screen to the newly created screen. This adds a menu item to the RevenueByY screen.
3. Go to the screen (dwChart) design page. Select the **HtmlView** control from the toolbox, then click inside the dwChart screen.
4. Click the **HtmlView** control, and on the property tab, create a new key (“dwChartKey” or any name) for this view by clicking **New Key**. Set the value for this key as:

```
<div id="dwContents">

<div id="dwChart_1"></div>
</div>
```

5. Create three keys for data mapping. The Dept MBO needs one parameter, and the Employee MBO needs two parameters to retrieve data. Therefore, define three keys for the user to input the data in the workflow.
    - Click the dwChart screen outside the HtmlView control.
    - Go to the property tab.
    - In the left pane, click **Keys**, and in the right pane, click **New.**
        - Enter **KeyforQueryDept** as the name.
        - Enter **int** as the type.
        - Leave all other fields at their default values.
    - Click **OK** to save.

    Repeat these steps to create two more keys named **branKeyforQueryEmp** and **deptKeyforQueryEmp**. All other properties are the same as **KeyforQueryDept**.

    Note: Make note of these key names (**KeyforQueryDept, branKeyforQueryEmp, deptKeyforQueryEmp**) for later use, when you define the screen and menu item.
6. Create another three screens for data retrieval. To bind the MBOs to the workflow, you must create a screen for each MBO: Branch, Dept, and Employee.
    - In the WorkSpace Navigator, click **demo1 > Mobile Business Object** to show all MBOs.
    - Drag the **Branch** MBO to the workflow form. Two screens are automatically created—Branch and BranchDetail. The Branch screen contains a listview that shows the data from the Branch MBO.
    - Drag the **dept** MBO to the workflow form. Two screens are automatically created—dept and deptDetail. The **dept** screen contains a listview that shows the data from the **dept** MBO.
    - Drag the **Employee** MBO to the workflow form. Two screens are automatically created—Employee and EmployeeDetail. The Employee screen contains a listview that shows the data from the Employee MBO.
7. Add one action menu item to retrieve data from the Branch MBO.
    - On the Flow Design tab, double-click the **dwChart** screen.

- Delete the "Submit" menu item if it exists. Then add a new menu item named `GetBranch`.
- On the GetBranch property page:
  - Change the type from "Save Screen" to "Online Request."
  - Set the Mobile Business Object to `demo1/Branch`.
  - Select **Invoke object query** and unselect "Generate old values keys."
  - For Object Query, select **findAll** to retrieve all data rows.
- Save the screen.

8. Add another action menu item to retrieve data from the Dept MBO.
   - On the Flow Design tab, double-click the **dwChart**.
   - Add a new menu item named `GetDept`.
   - On the GetDept property page:
     - Change the type from "Save Screen" to "Online Request."
     - Set the Mobile Business Object to `demo1/dept`.
     - Select **Invoke object query** and unselect "Generate old values keys."
     - For Object Query, select **findByBranchCode** to retrieve data.
   - In the left pane, click **Parameter Mappings.** You see there is one parameter (**brCode_param**). Click **Edit** to set the mapping. Set the mapping type as **Key** and select **KeyforQueryDept** as the mapping key.
   - Save the screen.
9. Add one more action menu item to retrieve data from the Employee MBO.
   - On the Flow Design tab, double-click **dwChart**.
   - Add a new menu item named `GetEmployee`.
   - On the GetEmployee property page:
     - Change the type from "Save Screen" to "Online Request."
     - Set the Mobile Business Object to `demo1/Employee`.
     - Select **Invoke object query** and unselect "Generate old values keys."
     - For Object Query, select **findByBranchDeptCode** to retrieve data.
   - In the left pane, click **Parameter Mappings.**
     - Choose **emp_brCode_param** and click **Edit** to set the mapping. Set the mapping type as **Key** and select **branKeyforQueryEmp** as the mapping key.
     - Choose **emp_deptCode_param** and click **Edit** to set the mapping. Set the mapping type as **Key** and select **deptKeyforQueryEmp** as the mapping key.
   - Save the screen.

Creating the MAKit Drilldrown Chart

The generated code for the hybrid app includes a `Custom.js` file. You can add your code to this file to customize your application. For our sample, add the following code to the **hwc.customAfterShowScreen** function to create the drilldown chart. Leave the remainder of this function unchanged.

```
if (screenToShow === "dwChart") {

    showFulldwChart("dwChart_1", "data/drilldown_ms_bubble.xml");

}
```

Note: "dwChart" is the key of the screen; "dwContents" is the ID defined above for a placeholder; "dwChart_1" is the ID of the chart defined above; "data/drilldown_ms_bubble.xml" is the metadata file.

The definition of the **showFulldwChart** function is:

```
function showFulldwChart(htmlElementID, metadataFile) {

    arrangeDrillDownCharts(1, 1, "dwContents");
```

```
    var elem = document.getElementById(htmlElementID);

    var chart = new $MA.DrillDownGroup(elem.id, true);

    chart.load(metadataFile);



}
```

Getting Data from the SAP Mobile Server

The drilldown chart needs several queries to get the data for different drilldown levels. Whenever the chart needs data, it calls the **executeMAQuery** function and passes the query name to it. For this sample, we need three queries, to retrieve data from each of the Branch, Department, and Employee MBOs. Hybrid App developers must add code to implement these queries.

In the SAP Mobile Platform Hybrid Web container, data retrieval is asynchronous, which means that when you request data from the SAP Mobile Server, the application does not wait for the return of the data, but continues executing code. When the data is ready, the application calls the callback function **hwc.customAfterDataReceived**.

Adding Global Variables and Modifying the executeMAQuery Function

Add new query handling code to **executeMAQuery** as shown below. Leave the existing code in this function unchanged:

```
var yDataOK;                            //if the data for Year is ready

var qDataOK = false;                    //if the data for Quarter is ready

var mDataOK = false;                    //if the data for Month is ready

var myWorkflowMessage;                  //the incoming message

var ChartObject;                        //the chart object

var QueryName;                          // the query name

var Args;                               //possible argument



var branDataOK = false;                 //if the data for Branch is ready

var bran4DeptQuery;           //variable to store the branch value for dept retrieval

var bran4EmployeeKey;         //variable to store the branch value for employee retrieval

var dept4EmployeeKey;         //variable to store the dept value for employee retrieval


function executeMAQuery(chartObj, name, args){

    var dataTable;

    if (name === "productRevenueByYear") {               //for the simple chart sample

        dataTable = getProductRevenueByYearData();

        chartObj.setDataTable(dataTable);

    }
```

```
else if (name === "productRevenueByQuarter") {
    if (qDataOK == false) {
            ChartObject = chartObj;
            QueryName = name;
            Args = args;
            retrieveRevenueByQData();        //send request to get data(1)
    }
    else {
            //get the data and pass it to the chart
            var dataTable = getProductRevenueByQuarterData(args); //get data (2)
            chartObj.setDataTable(dataTable);
            chartObj.refresh();
            QueryName = null;
            Args = null;
    }
}
else if (name === "productRevenueByMonth") {
    if (mDataOK == false) {
            ChartObject = chartObj;
            QueryName = name;
            Args = args;
            retrieveRevenueByMData();        //send request to get data(3)
    }
    else {
            var dataTable = getProductRevenueByMonthData(args);      //get data (4)
            chartObj.setDataTable(dataTable);
            chartObj.refresh();
            QueryName = null;
            Args = null;
    }
}
```

```
    else if (name === "branIncomeDetail") {

        if (branDataOK == false) {

                ChartObject = chartObj;

                QueryName = name;

                Args = args;

                retrieveBranchDetailData();      //send request to get data

        }

        else {

                var dataTable = getBranchDetailData(args);       //get data

                chartObj.setDataTable(dataTable);

                chartObj.refresh();

        }

    }

    else if (name === "deptIncomeDetail") {

        ChartObject = chartObj;

        QueryName = name;

        Args = args;

        retrieveDeptDetailData(args);    //send request to get data

    }

    else if (name === "empIncomeDetail") {

        ChartObject = chartObj;

        QueryName = name;

        Args = args;

        retrieveEmployeeDetailData(args);        //send request to get data

    }

    else {



    }

}
```

Note: “branIncomeDetail", “deptIncomeDetail”, and “empIncomeDetail” are the query names defined in the metadata for branch, department, and employee drilldown levels.

Requesting Data from the Server

If branch data is not ready, or is being retrieved for the first time, **retrieveBranchDetail** is called to request the data. The function is defined as:

```
//Send request to get Branch data

function retrieveBranchDetailData() {

    var rmiKeys = [];

    var rmiKeyTypes = [];

    var rmiInputOnlyKeys = [];

    var rmiInputOnlyKeyTypes = [];

    var workflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('dwChart', 'GetBranch',
rmiKeys, rmiKeyTypes);

var inputOnlyWorkflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('dwChart', 'GetBranch',
rmiInputOnlyKeys, rmiInputOnlyKeyTypes);

hwc.doOnlineRequest('dwChart', 'GetBranch', 60, 0, '', null, workflowMessageToSend,
inputOnlyWorkflowMessageToSend.serializeToString());

}

//Send request to get Dept data

function retrieveDeptDetailData(args) {

    var rmiKeys = [];

    var rmiKeyTypes = [];

    var rmiInputOnlyKeys = [];

    var rmiInputOnlyKeyTypes = [];

    bran4DeptQuery = args[0].itemValues[0];      //save the branch code

    var workflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('dwChart', 'GetDept',
rmiKeys, rmiKeyTypes);

var inputOnlyWorkflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('dwChart', 'GetDept',
rmiInputOnlyKeys, rmiInputOnlyKeyTypes);

hwc.doOnlineRequest('dwChart', 'GetDept', 60, 0, '', null, workflowMessageToSend,
inputOnlyWorkflowMessageToSend.serializeToString());

}

//Send request to get Employee data

function retrieveEmployeeDetailData(args) {

    var rmiKeys = [];

    var rmiKeyTypes = [];
```

```
    var rmiInputOnlyKeys = [];

    var rmiInputOnlyKeyTypes = [];

    bran4EmployeeKey = args[0].itemValues[0];    //save the branch code

    dept4EmployeeKey = args[1].itemValues[0];    //save the department code

    var workflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('dwChart', 'GetEmployee',
rmiKeys, rmiKeyTypes);

var inputOnlyWorkflowMessageToSend = hwc.getMessageValueCollectionForOnlineRequest('dwChart',
'GetEmployee', rmiInputOnlyKeys, rmiInputOnlyKeyTypes);

hwc.doOnlineRequest('dwChart', 'GetEmployee', 60, 0, '', null, workflowMessageToSend,
inputOnlyWorkflowMessageToSend.serializeToString());

}
```

Adding Values for the Query Parameters

To retrieve department and employee data, we must provide the values for the query parameters before the online request is actually submitted. The SAP Mobile Platform Hybrid App container provides a callback function, **hwc.customBeforeSubmit**, in `Custom.js` that allows you to customize the message before it is submitted. For the sample, customize the request message to include the query parameters. After customization, this function should look similar to:

```
hwc.customBeforeSubmit (screenKey, actionName, dataMessageToSend) {

    if ((screenKey === "dwChart") && (actionName === "GetDept")) {

        var myNewValue1 = new MessageValue();

        myNewValue1.setKey("KeyforQueryDept");

        myNewValue1.setValue(bran4DeptQuery);    // bran4DeptQuery was set in retrieve function

        myNewValue1.setType("NUMBER");



        var mvc = dataMessageToSend.getValues();

        if( mvc ) {

                var oldKey = mvc.getData("KeyforQueryDept");

                if (oldKey != undefined) {

                        mvc.remove("KeyforQueryDept");

                }

                mvc.add( myNewValue1.getKey(), myNewValue1 );

        }

    }

    if ((screenKey === "dwChart") && (actionName === "GetEmployee")) {

        var myNewValue1 = new MessageValue();
```

```
        myNewValue1.setKey("branKeyforQueryEmp");

        myNewValue1.setValue(bran4EmployeeKey); //bran4EmployeeKey was set in retrieve function

        myNewValue1.setType("NUMBER");


        var myNewValue2 = new MessageValue();

        myNewValue2.setKey("deptKeyforQueryEmp");

        myNewValue2.setValue(dept4EmployeeKey); //bran4EmployeeKey was set in retrieve function

        myNewValue2.setType("NUMBER");


        var mvc = dataMessageToSend.getValues();

        if( mvc ) {

                //add the query value for branCode

                var oldKey = mvc.getData("branKeyforQueryEmp");

                if (oldKey != undefined) {

                        mvc.remove("branKeyforQueryEmp");

                }

                mvc.add( myNewValue1.getKey(), myNewValue1 );


                //add the query value for deptCode

                var oldKey = mvc.getData("deptKeyforQueryEmp");

                if (oldKey != undefined) {

                        mvc.remove("deptKeyforQueryEmp");

                }

                mvc.add( myNewValue2.getKey(), myNewValue2 );

        }

    }

    return true;

}
```

Getting the Return Data

Data communication from the SAP Mobile Server is asynchronous. When the data is ready, the Hybrid Web Container calls the callback function **hwc.customAfterDataReceived** in `Custom.js`. Add code to this function to get branch, department, and employee data, and pass it to the drilldown chart. Add your code for this drilldown chart; but do not remove the code for other charts. After customization, **hwc.customAfterDataReceived** should look similar to:

```
hwc.customAfterDataReceived = function(incomingDataMessage) {

    myWorkflowMessage = incomingDataMessage;

    try

    {

        var valueCols = incomingDataMessage.getValues();

        //check data for year

        var yItemList = valueCols.getData("RevenueByY");

        if (yItemList != undefined) {

                yDataOK = true;

        }

        else {

                yDataOK = false;

        }

        //check data for quarter

        var qItemList = valueCols.getData("RevenueByQ");

        if (qItemList != undefined) {

                qDataOK = true;

        }

        else {

                qDataOK = false;

        }

        //check data for month

        var mItemList = valueCols.getData("RevenueByM");

        if (mItemList != undefined) {

                mDataOK = true;

        }

        else {

                mDataOK = false;

        }

        //check data for branch

        var bItemList = valueCols.getData("Branch");
```

```
        if (bItemList != undefined) {
                branDataOK = true;
        }
        else {
                branDataOK = false;
        }
        //get data for Year
        if (QueryName === "productRevenueByYear") {
                var dataTable = getProductRevenueByYearData();
                ChartObject.setDataTable(dataTable);
                dataTable.refresh();
        }
        else if (QueryName === "productRevenueByQuarter") {
                var dataTable = getProductRevenueByQuarterData(Args);
                ChartObject.setDataTable(dataTable);
                dataTable.refresh();


        }


        else if (QueryName === "productRevenueByMonth") {
                var dataTable = getProductRevenueByMonthData(Args);
                ChartObject.setDataTable(dataTable);
                dataTable.refresh();
        }
        else if (QueryName === "branIncomeDetail") {
                var dataTable = getBranchDetailData(Args);
                ChartObject.setDataTable(dataTable);
                dataTable.refresh();
        }
        else if (QueryName === "deptIncomeDetail") {
                var dataTable = getDeptDetailData(Args);
```

```
                ChartObject.setDataTable(dataTable);

                dataTable.refresh();

        }

        else if (QueryName === "empIncomeDetail") {

                var dataTable = getEmployeeDetailData(Args);

                ChartObject.setDataTable(dataTable);

                dataTable.refresh();

        }

        else {



        }



        QueryName = null;

        Args = null;

    }

    catch (err)

    {

        QueryName = null;

        Args = null;

    }

}
```

The functions to get the data are defined as in `MyQuery.js`:

```
    //get Branch data

function getBranchDetailData(args)

{

    var dataTable = new $MA.DataTable();

    dataTable.addColumn("branCode", "number");

    dataTable.addColumn("incomeType", "string");

    dataTable.addColumn("branIncome", "number");

    dataTable.addColumn("bubbleSize", "number");
```

```
//Get data from SUP MBO -- Branch

var message = getCurrentMessageValueCollection();

var itemList = message.getData("Branch");

var items = itemList.getValue();

var noOfItems = items.length;

if (noOfItems > 0) {

    var i = 0;

    while (i < noOfItems) {

       var theItems = items[i];

       var yearObj=theItems.getData("Branch_branCode_attribKey");

       var branCode = yearObj.getValue();


       var prodObj=theItems.getData("Branch_incomeType_attribKey");

       var incometype = prodObj.getValue();


       var qObj=theItems.getData("Branch_income__attribKey");

       var income = qObj.getValue();


       var mObj=theItems.getData("Branch_size__attribKey");

       var sz = mObj.getValue();


       dataTable.addRow([branCode, incometype, income, sz] );


       i = i + 1;

     }

}

else {

    alert("No data is available.");

}


return dataTable;
```

```
}

//get Dept data

function getDeptDetailData(args)

{

    var dataTable = new $MA.DataTable();

    dataTable.addColumn("branCode", "number");

    dataTable.addColumn("deptCode", "number");

    dataTable.addColumn("incomeType", "string");

    dataTable.addColumn("branIncome", "number");

    dataTable.addColumn("bubbleSize", "number");


    //Get data from SUP MBO -- Dept

    var message = getCurrentMessageValueCollection();

    var itemList = message.getData("dept");

    var items = itemList.getValue();

    var noOfItems = items.length;

    if (noOfItems > 0) {

        var i = 0;

        var data = "";

         while (i < noOfItems) {

           var theItems = items[i];

           var yearObj=theItems.getData("dept_branCode_attribKey");

           var branCode = yearObj.getValue();


   var deptObj=theItems.getData("dept_deptCode_attribKey");

           var deptCode = deptObj.getValue();


           var prodObj=theItems.getData("dept_incomeType_attribKey");

           var incometype = prodObj.getValue();


           var qObj=theItems.getData("dept_income__attribKey");
```

```
            var income = qObj.getValue();


            var mObj=theItems.getData("dept_size__attribKey");
            var sz = mObj.getValue();


            dataTable.addRow([branCode, deptCode, incometype, income, sz] );
            i = i + 1;
          }
    }
    else {
        alert("No data is available.");
    }


    return dataTable;
}
//get Employee data
function getEmployeeDetailData(args)
{
    var dataTable = new $MA.DataTable();
    dataTable.addColumn("branCode", "number");
    dataTable.addColumn("deptCode", "number");
    dataTable.addColumn("empCode", "number");
    dataTable.addColumn("incomeType", "string");
    dataTable.addColumn("branIncome", "number");
    dataTable.addColumn("bubbleSize", "number");


    //Get data from SUP MBO -- Employee
    var message = getCurrentMessageValueCollection();
    var itemList = message.getData("Employee");
    var items = itemList.getValue();
    var noOfItems = items.length;
```

```
if (noOfItems > 0) {

    var i = 0;

    var data = "";


     while (i < noOfItems) {

       var theItems = items[i];

       var yearObj=theItems.getData("Employee_branCode_attribKey");

       var branCode = yearObj.getValue();


       var deptObj=theItems.getData("Employee_deptCode_attribKey");

       var deptCode = deptObj.getValue();


       var empObj=theItems.getData("Employee_empCode_attribKey");

       var empCode = deptObj.getValue();


       var prodObj=theItems.getData("Employee_incomeType_attribKey");

       var incometype = prodObj.getValue();


       var qObj=theItems.getData("Employee_income__attribKey");

       var income = qObj.getValue();


       var mObj=theItems.getData("Employee_size__attribKey");

       var sz = mObj.getValue();


       dataTable.addRow([branCode, deptCode, empCode, incometype, income, sz] );

       i = i + 1;

     }

}

else {

    alert("No data is available.");

}
```

```
    return dataTable;

}
```

Deploying the Hybrid App

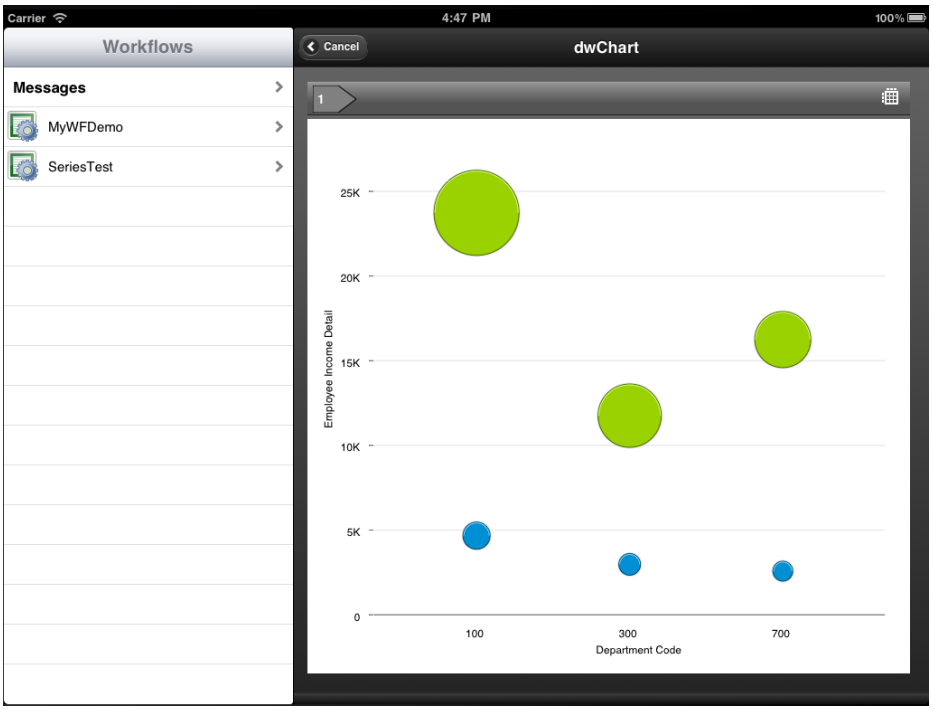
1. Right-click the **Flow Design** or **Screen Design** page and select **Generate Hybrid App**.
2. Keep the default settings.
3. Click **Finish**.

You should be able to show a Hybrid App that shows a drilldown chart.

Note: If the app transits to the dwChart screen, you see three menu items—GetBranch, GetDept, and GetEmployee. To hide them, add this code to the **hwc.customBeforeShowScreen** function in `Custom.js`:

```
if (screenToShow === "dwChart") {

    hwc.removeMenuItem("dwChart", "GetBranch");

    hwc.removeMenuItem("dwChart", "GetDept");

    hwc.removeMenuItem("dwChart", "GetEmployee");

}
```

The drilldown chart looks similar to that shown below. Double-click any bubble in the chart to drill down to next level. To drill up, click the level buttons at the top-left corner.



System Requirements for the SAP Mobile Platform Web Container

	OS

iOS Hybrid Web Container V2.2.2	iOS
Android Hybrid Web Container V2.2.2	Android

Reference

SQL Statements for RevenueByY Table

```
/* create RevenueByY */

CREATE TABLE dba.RevenueByY  (

product NVARCHAR(20) NOT NULL,

year DOUBLE NOT NULL,

revenue DOUBLE NOT NULL,

);


/* insert sample data*/

INSERT INTO dba. RevenueByY  VALUES ("laptop",2005.0,2000.0);

INSERT INTO dba. RevenueByY  VALUES ("desktop",2005.0,1500.0);

INSERT INTO dba. RevenueByY  VALUES ("tablet",2005.0,1200.0);

INSERT INTO dba. RevenueByY  VALUES ("desktop",2006.0,1700.0);

INSERT INTO dba. RevenueByY  VALUES ("laptop",2006.0,2300.0);

INSERT INTO dba. RevenueByY  VALUES ("tablet",2006.0,1500.0);

INSERT INTO dba. RevenueByY  VALUES ("laptop",2007.0,2500.0);

INSERT INTO dba. RevenueByY  VALUES ("desktop",2007.0,1800.0);

INSERT INTO dba. RevenueByY  VALUES ("tablet",2007.0,1500.0);

INSERT INTO dba. RevenueByY  VALUES ("laptop",2008.0,2800.0);

INSERT INTO dba. RevenueByY  VALUES ("desktop",2008.0,2000.0);

INSERT INTO dba. RevenueByY  VALUES ("tablet",2008.0,1600.0);

INSERT INTO dba. RevenueByY  VALUES ("laptop",2009.0,2900.0);

INSERT INTO dba. RevenueByY  VALUES ("desktop",2009.0,2100.0);

INSERT INTO dba. RevenueByY  VALUES ("tablet",2009.0,2000.0);

INSERT INTO dba. RevenueByY  VALUES ("laptop",2010.0,2700.0);

INSERT INTO dba. RevenueByY  VALUES ("desktop",2010.0,1900.0);
```

```
INSERT INTO dba. RevenueByY  VALUES ("tablet",2010.0,2500.0);
```

SQL Statements for RevenueByQ Table

```
/* create table RevenueByQ */

CREATE TABLE dba.RevenueByQ (

product NVARCHAR(20) NOT NULL,

year DOUBLE NOT NULL,

quarter DOUBLE NOT NULL,

revenue DOUBLE NOT NULL,

);


/* insert sample data*/

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2005.0,1.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2005.0,1.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2005.0,1.0,200.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2005.0,2.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2005.0,2.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2005.0,2.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2005.0,3.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2005.0,3.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2005.0,3.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2005.0,4.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2005.0,4.0,700.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2005.0,4.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2006.0,1.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2006.0,1.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2006.0,1.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2006.0,2.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2006.0,2.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2006.0,2.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2006.0,3.0,400.0);
```

```sql
INSERT INTO dba.RevenueByQ  VALUES ("laptop",2006.0,3.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2006.0,3.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2006.0,4.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2006.0,4.0,900.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2006.0,4.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2007.0,1.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2007.0,1.0,100.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2007.0,1.0,200.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2007.0,2.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2007.0,2.0,800.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2007.0,2.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2007.0,3.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2007.0,3.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2007.0,3.0,200.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2007.0,4.0,800.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2007.0,4.0,1000.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2007.0,4.0,700.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2008.0,1.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2008.0,1.0,200.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2008.0,1.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2008.0,2.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2008.0,2.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2008.0,2.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2008.0,3.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2008.0,3.0,800.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2008.0,3.0,300.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2008.0,4.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2008.0,4.0,1200.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2008.0,4.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2009.0,1.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2009.0,1.0,400.0);
```

```
INSERT INTO dba.RevenueByQ  VALUES ("tablet",2009.0,1.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2009.0,2.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2009.0,2.0,800.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2009.0,2.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2009.0,3.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2009.0,3.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2009.0,3.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2009.0,4.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2009.0,4.0,1100.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2009.0,4.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2010.0,1.0,200.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2010.0,1.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2010.0,1.0,400.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2010.0,2.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2010.0,2.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2010.0,2.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2010.0,3.0,500.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2010.0,3.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2010.0,3.0,600.0);

INSERT INTO dba.RevenueByQ  VALUES ("desktop",2010.0,4.0,700.0);

INSERT INTO dba.RevenueByQ  VALUES ("laptop",2010.0,4.0,1100.0);

INSERT INTO dba.RevenueByQ  VALUES ("tablet",2010.0,4.0,1000.0);
```

SQL Statements for RevenueByM Table

```
/* create RevenueByM */

CREATE TABLE dba.RevenueByM (

product NVARCHAR(20) NOT NULL,

year DOUBLE NOT NULL,

quarter DOUBLE NOT NULL,

month DOUBLE NOT NULL,

revenue DOUBLE NOT NULL,

);
```

```
/* insert sample data*/
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,1.0,1.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,1.0,1.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,1.0,1.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,1.0,2.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,1.0,2.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,1.0,2.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,2.0,4.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,2.0,4.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,2.0,4.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,2.0,5.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,2.0,5.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,2.0,5.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,2.0,6.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,2.0,6.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,2.0,6.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,3.0,7.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,3.0,7.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,3.0,7.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,3.0,8.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,3.0,8.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,3.0,8.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,3.0,9.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,3.0,9.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,3.0,9.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,4.0,10.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,4.0,10.0,600.0);
```

```
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,4.0,10.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,4.0,11.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,4.0,11.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,4.0,11.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2005.0,4.0,12.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2005.0,4.0,12.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2005.0,4.0,12.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,1.0,1.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,1.0,1.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,1.0,1.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,1.0,2.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,1.0,2.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,1.0,2.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,2.0,4.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,2.0,4.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,2.0,4.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,2.0,5.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,2.0,5.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,2.0,5.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,2.0,6.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,2.0,6.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,2.0,6.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,3.0,7.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,3.0,7.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,3.0,7.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,3.0,8.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,3.0,8.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,3.0,8.0,300.0);
```

```
INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,3.0,9.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,3.0,9.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,3.0,9.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,4.0,10.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,4.0,10.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,4.0,10.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,4.0,11.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,4.0,11.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,4.0,11.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2006.0,4.0,12.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2006.0,4.0,12.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2006.0,4.0,12.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,1.0,1.0,100.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,1.0,1.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,1.0,1.0,100.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,1.0,2.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,1.0,2.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,1.0,2.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,1.0,3.0,100.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,1.0,3.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,1.0,3.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,2.0,4.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,2.0,4.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,2.0,4.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,2.0,5.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,2.0,5.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,2.0,5.0,500.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,2.0,6.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,2.0,6.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,2.0,6.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,3.0,7.0,200.0);
```

```
INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,3.0,7.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,3.0,7.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,3.0,8.0,500.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,3.0,8.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,3.0,8.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,3.0,9.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,3.0,9.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,3.0,9.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,4.0,10.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,4.0,10.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,4.0,10.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,4.0,11.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,4.0,11.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,4.0,11.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2007.0,4.0,12.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2007.0,4.0,12.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2007.0,4.0,12.0,700.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,1.0,1.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,1.0,1.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,1.0,1.0,50.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,1.0,2.0,150.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,1.0,2.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,1.0,2.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,1.0,3.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,1.0,3.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,2.0,4.0,100.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,2.0,4.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,2.0,4.0,320.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,2.0,5.0,250.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,2.0,5.0,300.0);
```

```sql
INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,2.0,5.0,320.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,2.0,6.0,350.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,2.0,6.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,2.0,6.0,320.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,3.0,7.0,350.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,3.0,7.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,3.0,7.0,480.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,3.0,8.0,450.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,3.0,8.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,3.0,8.0,460.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,3.0,9.0,500.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,3.0,9.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,3.0,9.0,430.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,4.0,10.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,4.0,10.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,4.0,10.0,580.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,4.0,11.0,500.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,4.0,11.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,4.0,11.0,620.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2008.0,4.0,12.0,550.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2008.0,4.0,12.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2008.0,4.0,12.0,660.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,1.0,1.0,230.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,1.0,1.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,1.0,1.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,1.0,2.0,250.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,1.0,2.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,1.0,2.0,210.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,1.0,3.0,270.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,1.0,3.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,1.0,3.0,280.0);
```

```
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,2.0,4.0,350.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,2.0,4.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,2.0,4.0,270.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,2.0,5.0,340.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,2.0,5.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,2.0,5.0,320.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,2.0,6.0,350.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,2.0,6.0,300.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,2.0,6.0,310.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,3.0,7.0,480.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,3.0,7.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,3.0,7.0,470.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,3.0,8.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,3.0,8.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,3.0,8.0,470.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,3.0,9.0,460.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,3.0,9.0,400.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,3.0,9.0,430.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,4.0,10.0,630.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,4.0,10.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,4.0,10.0,640.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,4.0,11.0,620.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,4.0,11.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,4.0,11.0,620.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2009.0,4.0,12.0,610.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2009.0,4.0,12.0,600.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2009.0,4.0,12.0,680.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,1.0,1.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,1.0,1.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,1.0,1.0,200.0);
INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,1.0,2.0,200.0);
```

```
INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,1.0,2.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,1.0,2.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,1.0,3.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,1.0,3.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,1.0,3.0,200.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,2.0,4.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,2.0,4.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,2.0,4.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,2.0,5.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,2.0,5.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,2.0,5.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,2.0,6.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,2.0,6.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,2.0,6.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,3.0,7.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,3.0,7.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,3.0,7.0,300.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,3.0,8.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,3.0,8.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,3.0,8.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,3.0,9.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,3.0,9.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,3.0,9.0,400.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,4.0,10.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,4.0,10.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,4.0,10.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,4.0,11.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,4.0,11.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,4.0,11.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("desktop",2010.0,4.0,12.0,600.0);

INSERT INTO dba.RevenueByM  VALUES ("laptop",2010.0,4.0,12.0,600.0);
```

```
INSERT INTO dba.RevenueByM  VALUES ("tablet",2010.0,4.0,12.0,600.0);
```

SQL Statements for EmpIncome Table

```
/* create EmpIncome */

CREATE TABLE dba.EmpIncome (

incomeType VARCHAR(20) NOT NULL,

branCode INT NOT NULL,

deptCode INT NOT NULL,

empCode INT NOT NULL,

income DOUBLE NOT NULL,

size INT NOT NULL,

)

IN SYSTEM

;


/* Insert sample data */

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,1000,10000.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,1000,1000.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,1001,2000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,1001,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,3000,1000.0,200);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,3000,800.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,4000,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,4000,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,7000,1200.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,7000,100.0,20);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,8000,1500.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,8000,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,9000,700.0,120);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,9000,300.0,10);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,100,10000,1300.0,40);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,100,10000,400.0,20);
```

```sql
INSERT INTO dba.EmpIncome  VALUES ("Salary",1,300,10,2000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,300,10,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,300,3500,1000.0,200);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,300,3500,800.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,300,4000,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,300,4000,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,300,7500,1200.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,300,7500,100.0,20);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,300,8000,1500.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,300,8000,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,700,105,2000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,700,105,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,700,410,1000.0,200);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,700,410,800.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,700,530,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,700,530,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,700,720,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,700,720,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",1,700,1000,1200.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",1,700,1000,100.0,20);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,200,1001,2000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,200,1001,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,200,3000,1000.0,200);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,200,3000,800.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,200,4000,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,200,4000,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,400,7000,1200.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,400,7000,100.0,20);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,400,8000,1500.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,400,8000,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,400,9000,700.0,120);
```

```
INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,400,9000,300.0,10);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,400,10000,1300.0,40);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,400,10000,400.0,20);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,400,10,2000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,400,10,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,800,300,1000.0,200);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,800,300,800.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,800,1000,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,800,1000,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,800,7500,1200.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,800,7500,100.0,20);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,800,8000,1500.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,800,8000,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",2,800,9000,2000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,800,8000,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,300,1001,3000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,300,1001,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,300,3000,1000.0,300);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,300,3000,800.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,300,4000,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,300,4000,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,400,7000,1300.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,400,7000,100.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,400,8000,1500.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,400,8000,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,400,9000,700.0,130);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",2,400,9000,300.0,10);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,400,10000,1300.0,40);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,400,10000,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,400,10,3000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,400,10,800.0,50);
```

```sql
INSERT INTO dba.EmpIncome  VALUES ("Salary",3,800,300,1000.0,300);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,800,300,800.0,100);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,800,1000,6000.0,80);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,800,1000,400.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,800,7500,1300.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,800,7500,100.0,30);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,800,8000,1500.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,800,8000,800.0,50);

INSERT INTO dba.EmpIncome  VALUES ("Salary",3,800,9000,3000.0,150);

INSERT INTO dba.EmpIncome  VALUES ("Bonus",3,800,8000,800.0,50);
```