Mobile Analytics Kit Tutorial: iOS Metadata-Driven Development

---

Sybase Unwired Platform 2.2

Document ID: DC01727-01-0220-01
Last Revised: August 2012
Copyright © 2012 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at http://www.sybase.com/detail?id=1011207. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

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 in several other countries all over the world.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

IBM and Tivoli are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

Introduction 5

Prerequisites 5

Lesson 1: Setting Up the iOS Development Environment for MAKit 5

Lesson 2: Building Basic MAKit Charts Using Metadata 8

- Creating ChartsDatasource Class 8
- Creating the XML Metadata File 11
- Implementing the MAKit Query Module 12
  - Creating the ChartDataTable Class 13
  - Creating the ChartQuery Class 14
- Creating the User Interface for the Application 15
  - Configuring MAKitSampleAppDelegate Files 15
  - Adding the MainViewController 17
  - Making Connections 19
- Deploying the Application 20

Lesson 3: Building Semantic Zooming Charts Using Metadata 23

- Adding Methods to the ChartsDataSource Class 23
- Creating XML Metadata File with Zooming Levels 26
- Adding Scripts in ChartQuery and ViewController Classes 26
  - Modifying ChartQuery Class 26
  - Adding Script to MainViewController Class 27
- Deploying the Application 28

Lesson 4: Building Drilldown Charts Using Metadata 33

- Adding Methods to the ChartsDataSource Class 33
- Creating XML Metadata File Configured with DrillDownGroup 35
- Adding Scripts in ChartQuery and ViewController Classes 36
  - Modifying ChartQuery Class 36
  - Adding Script to MainViewController Class 37
- Deploying the Application 38

Lesson 5: Adding Dynamic Analytics to Charts 43

- Creating Dynamic Analytic XML Metadata Configuration 43

Adding Scripts in ChartQuery and ViewController Classes 45
Modifying ChartQuery Class..... 45
Adding Script to the MainViewController Class 45
Deploying the Application 45
Lesson 6: Adding What-If Analysis to Charts 47
Adding Methods to ChartsDataSource Class..... 48
Creating XML Metadata File Configured with What-If Columns..... 49
Adding Scripts in ChartQuery and ViewController Classes 50
Modifying the ChartQuery Class..... 50
Adding Script to the MainViewController Class 51
Deploying the Application 51
Lesson 7: MAKit Chart E-Mail Sharing..... 56
Sending a Chart Through E-mail..... 57

Introduction

Mobile Analytics Kit (MAKit) allows users to graphically view data in different types of charts. The *MAKit Tutorial: iOS Metadata-Driven Development* is to help developers quickly understand MAKit concepts, and expedite application development. MAKit, which lets users graphically view data and modify chart values to examine "what-if" scenarios, is distributed as a static library on the iPhone/iPad. MAKit can be used with the Sybase® Unwired Platform Hybrid Web Container.

Prerequisites

Before beginning the tutorials:

- Verify that you are running Intel-based Mac 10.6 or higher.
- Install Xcode 4.0 with iOS 4.3 SDK.
- Copy the latest iOS Mobile Analytics Kit (MAKit) build (library and header files) to a local Mac machine.
- On a local Mac machine, copy the iOS MAKit build to `/Users/<username>/MAKit`, where `<username>` is the user currently logged in.
- Download and unzip *MAKitTutorialDatabase.zip* to a local folder on the Mac. This file contains the SQLite® database `MAKitTutorials.sqlite`, which is used in the tutorial.
- Create a `/Users/<username>/MAKitTutorial` folder.

Note: All the path information in the tutorial is based on `/Users/<username>/MAKitTutorial`**.** If you create the sample on a different folder, adjust the path according to your local settings.

Lesson 1: Setting Up the iOS Development Environment for MAKit

This lesson describes:

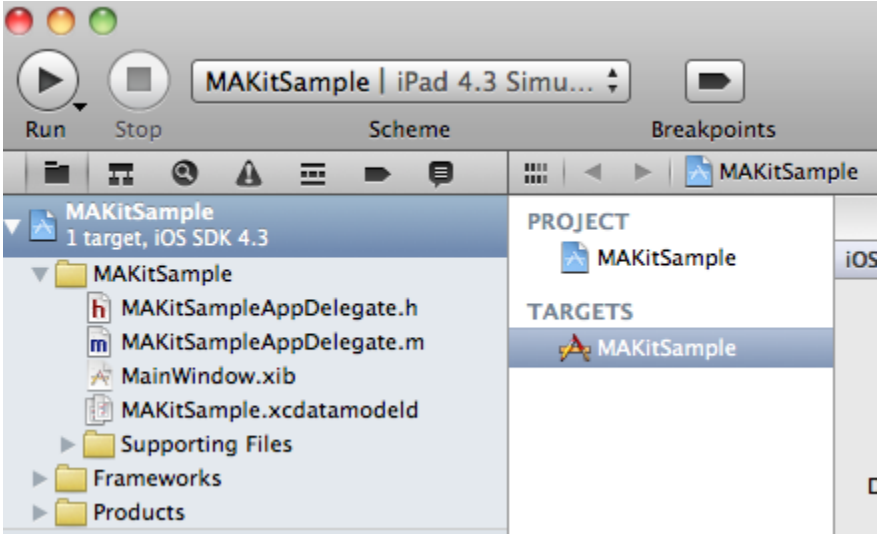
- Creating a new project in Xcode 4.0 IDE
- Adding files to the Xcode project
- Adding MAKit libraries and header files
- Changing the project build settings

Note: This tutorial is developed using Xcode 4.0 and iOS SDK 4.3. If you use a different version of Xcode, some steps may vary. For more information on Xcode, see the Apple Developer Connection at http://developer.apple.com/technologies/tools/xcode.html.

1. Start Xcode and select **File > New > New Project**.
2. Select **iOS Application** and **Window-based Application** as the project template, then click **Next.**
3. Enter `MAKitSample` as the product name, and `iPad` as the device family. Select **Use Core Data**, unselect **Include Unit Tests,** then click **Next**.
4. Save the project in `/Users/<username>/MAKitTutorial`, then click **Create** to open it.

Xcode creates a folder called `MAKitSample`, which contains a project file called `MAKitSample.xcodeproj` and a subfolder called `MAKitSample`, which contains all the Xcode-generated files in the `/Users/<username>/MAKitTutorial` folder.

After finishing this step, the project looks similar to this in Xcode:



5. In the Xcode Project Navigator, right-click **MAKitSample** and select **Add Files to "MAKitSample".**
6. Navigate to the folder containing the SQLite database, select **MAKitTutorials.sqlite**, select **Copy items into destination group's folder (if needed),** and click **Add.**

   This adds the `MAKitTutorials.sqlite` database to the project in the Project Navigator.

7. Right-click the `Frameworks` folder of the MAKitSample project and select **Add Files to "MAKitSample".** Navigate to `/Users/<username>/MAKit/libs/Debug-iphonesimulator` folder, select the `libMAKit.a` library, unselect **Copy items into destination group's folder (if needed),** and click **Add.**

   This adds `libMAKit.a` to the `Frameworks` folder of the project in the Project Navigator.

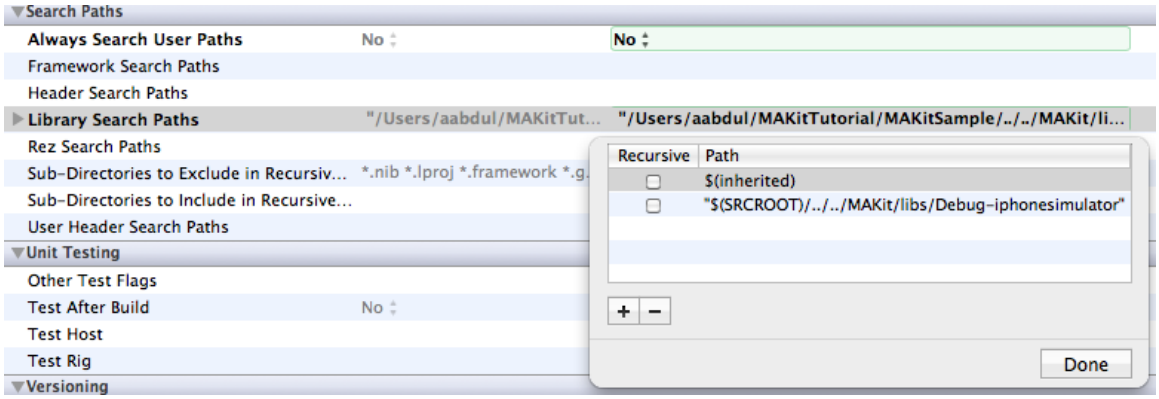
8. Right-click the project root, select **New Group**, and rename it `Resources`.
9. Right-click the `Resources` folder and select **Add Files to "MAKitSample".** Navigate to `/Users/<username>/MAKit`, select the `MAKit.bundle` file, select **Copy items into destination group's folder (if needed)** and click **Add**.

   This adds the `MAKit.bundle` file to the `Resources` folder in the Project Navigator.

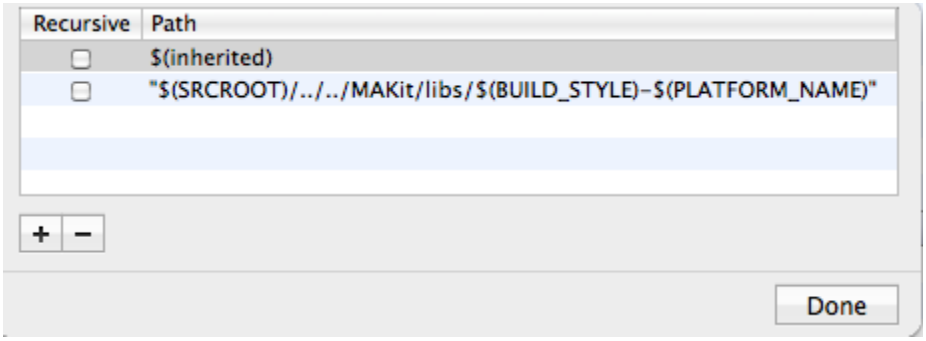
10. Click the project root and in the middle pane, click the **MAKitSample** target.

    a. In the right pane, click the **Build Settings** tab, then scroll down to the **Search Paths** section.
    b. Double-click **Library Search Paths.**

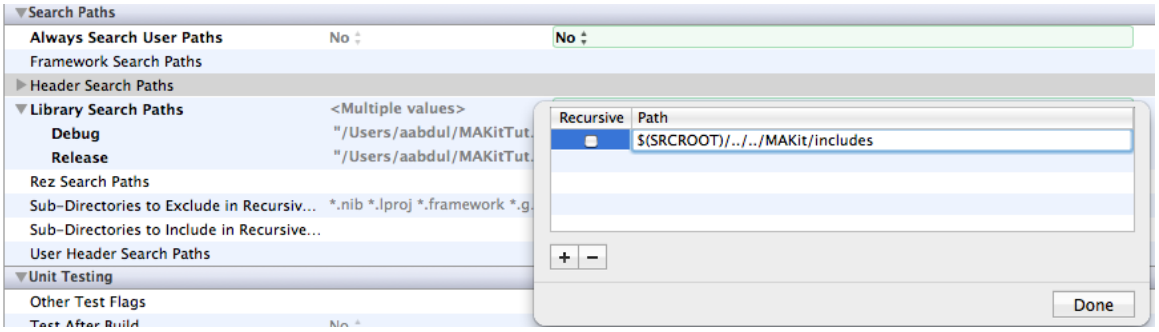
    This populates the Library Search Paths with `$SRCRoot/../../MAKit/libs/Debug-iphonesimulator` for the `libMAKit.a` library.



c. Replace the text `Debug-iphonesimulator` with `$(BUILD_STYLE)-$(PLATFORM_NAME)` macros and click **Done**.



d. Double-click the **Header Search Paths** property entry, click the **+** icon, enter the location of the MAKit header files, such as `$(SRCROOT)/../../MAKit/includes`, and click **Done**.



11. In the middle pane, click the **MAKitSample** target.

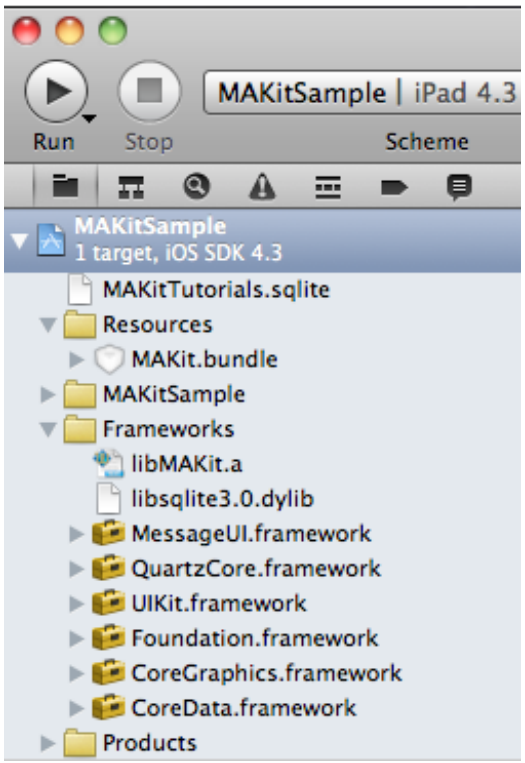
    a. In the right pane, select the **Build Phases** tab, then expand the **Link Binary with Libraries** section.

    b. Click the **+** icon below the list, select the following frameworks, and click **Add** to add them from the iOS 4.3 SDK to the project:

    - MessageUI.framework
    - QuartzCore.framework
    - libsqlite3.0.dylib

12. Drag the frameworks you added in step 11 to the `Frameworks` folder in the Project Navigator.

    After completing these steps, the Project Navigator looks similar to:



13. Select **Product > Clean**, then **Product > Build** to test the initial setup. If you correctly followed all the steps, you see a **`Build Succeeded`** message.

Lesson 2: Building Basic MAKit Charts Using Metadata

This lesson describes:

- Creating the ChartsDatasource class
- Creating an XML metadata file containing charts configuration
- Implementing the MAKit Query module for charts
- Configuring the user interface for the iOS application
- The build and execute procedure

Creating ChartsDatasource Class

Create the ChartsDatasource class to include a method to connect to the tutorial database `MAKitTutorials.sqlite`, and methods to retrieve data for the charts configured in the XML metadata file. The methods implemented in this class are called by the MAKit Query module.

Tip: To create the ChartsDatasource file, copy and paste the code below.

1. Open the `MAKitSample.xcodeproj` in the Xcode IDE.
2. In the MAKitSample project, select **File** > **New** > **New File**.
3. Select **iOS** > **Cocoa Touch** > **Objective-C class** and click **Next.**
4. Accept `NSObject` in the **Subclass of** field and click **Next.**
5. Enter `ChartsDatasource` in the **Save As** field and click **Save.**

This creates `ChartsDatasource.h` and `ChartsDatasource.m` files in the Project Navigator.

6. Click `ChartsDatasource.h` and replace the existing code with:

```
#import <Foundation/Foundation.h>
#import "sqlite3.h"

@interface ChartsDatasource : NSObject {
    sqlite3 *dbInstance;
}

//Database Methods
-(void) OpenDatabase;
-(void) CloseDatabase;

// Basic Charts Datasource Methods
-(NSArray *) NumberOfOrders;
-(NSArray *) SalesPerformance;

@end
```

7. Click **File** > **Save.**

8. Click `ChartsDatasource.m` and replace the existing code with:

```
#import "ChartsDatasource.h"


@implementation ChartsDatasource

//Open Database connection & assign to pointer dbInstance variable
-(void) OpenDatabase {
    @try {
    NSArray *path = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentDir = [path objectAtIndex:0];
    NSString *dbPath = [documentDir stringByAppendingPathComponent:
@"MAKitTutorials.sqlite"];

        if (sqlite3_open_v2([dbPath UTF8String], &dbInstance, SQLITE_OPEN_NOMUTEX,
NULL) == SQLITE_OK) {
            NSLog(@"Database Open Succeeds");
        }
        else {
            sqlite3_close(dbInstance);
        }

    }
    @catch (NSException * e) {
            NSLog(@"Failed to open the database. Reason %@",[e reason] );
    }

}

//Close Database connection
-(void) CloseDatabase {
    sqlite3_close(dbInstance);
}


-(id) init {
    if (self == [super init]) {
    dbInstance = nil;
        [self OpenDatabase];
    }
    return self;
}

-(void) dealloc {

    [self CloseDatabase];
    NSLog(@"Database Closed");
    [super dealloc];
}

//Sales Order Count by Year
-(NSArray *) NumberOfOrders {

   NSMutableArray *returnArray = [[NSMutableArray alloc] init];

   const char * SELECTQry = "SELECT strftime('%Y',OrderDate) as OrderYear, Region,
Count(OrderID) from SalesOrder_Header "
                            " Group By strftime('%Y',OrderDate),Region ";

   sqlite3_stmt *selectStmt = nil;

    if (sqlite3_prepare_v2(dbInstance, SELECTQry, -1, &selectStmt, NULL)==SQLITE_OK) {
        NSString *year = @"";
        NSString *region = @"";

        int ordercount=0;
           while (sqlite3_step(selectStmt)==SQLITE_ROW) {
           year = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 0)];
```

```
            region = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 1)];

            ordercount = sqlite3_column_int(selectStmt, 2);

            NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
            [dict setObject:year forKey:@"Year"];
            [dict setObject:region forKey:@"Region"];
            [dict setObject:[NSNumber numberWithInt:ordercount] forKey:@"OrderCount"];
            [returnArray addObject:dict];
            [dict release];
        }
    }
    else {
            NSLog(@"Failed at Prepare Statement in NumberOfOrders.
Reason %s",sqlite3_errmsg(dbInstance));
    }
    return returnArray;

}

//Sales Performance by Product/Region
-(NSArray *) SalesPerformance {

     NSMutableArray *returnArray = [[NSMutableArray alloc] init];

    const char * SELECTQry = "SELECT Region,ProductType, sum(UnitPrice * Quantity)
FROM SalesOrder_Header oh, SalesOrder_Details od, Product p "
            " WHERE oh.OrderID = od.OrderID AND od.ProductID = p.ProductID "
            " GROUP BY Region,ProductType ";

    sqlite3_stmt *selectStmt = nil;

    if (sqlite3_prepare_v2(dbInstance, SELECTQry, -1, &selectStmt, NULL)==SQLITE_OK) {
        NSString *region  = @"";
        NSString *product = @"";
        float salesamount = 0.0;
    while (sqlite3_step(selectStmt)==SQLITE_ROW) {
            region = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 0)];
            product = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 1)];
                    salesamount = sqlite3_column_double(selectStmt, 2);

            NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
            [dict setObject:region forKey:@"Region"];
            [dict setObject:product forKey:@"ProductType"];
            [dict setObject:[NSNumber numberWithFloat:salesamount]
forKey:@"SalesAmount"];
            [returnArray addObject:dict];

            [dict release];
            }
    }
    else {
            NSLog(@"Failed at Prepare Statement in SalesPerformance.
Reason %s",sqlite3_errmsg(dbInstance));
    }
    return returnArray;
}

@end
```

9. Click **File** > **Save.**

Creating the XML Metadata File

The XML metadata file includes the:

- Charts configuration, which includes the chart name, the query for the chart that pumps the data, and the chart's category, series, and value axis information. The chart configuration resides in the *<dashboard> … </dashboard>* tag.
- Dashboard screen layout, for both landscape and portrait modes, which is configured in the *<DashboardLayout> … </DashboardLayout>* tag.

1. In the MAKitSample project, click **File** > **New** > **New File.**
2. Select **iOS** > **Other** > **Empty** and click **Next.**
3. Enter `SalesMetadata.xml` in the **Save As** field and click **Save.**

   This creates the `SalesMetadata.xml` file in the Project Navigator.
4. Click **`SalesMetadata.xml`** and add this content to the file:

```
<?xml version="1.0" encoding="utf-8"?>
<MobileAnalytic MAKitVersion="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MAKitMetaFile.xsd">
  <!-- Dashboard Charts configuration -->
  <Dashboard>
    <!-- Column Chart configuration for Number of Sales Orders by Year -->
    <Chart Name="SalesOrderCount" Title="Number of Sales Orders" ChartType="Column"
Query="OrderCount">
      <Category Format="" Column="Year" DisplayName="Sales Year" />
      <Series Format="" Column="Region" DisplayName="Region" />
      <Values>
          <Value Format="Rounded0" Expression="OrderCount" ValueSequence="1" DisplayName="# of
Order for year" />
      </Values>
    </Chart>

     <!-- Line Chart configuration for Sales Performance -->
    <Chart Name="SalesPerformance" Title="Sales Performance By Region" ChartType="Line"
Query="SalesPerf">
       <Category Format="" Column="Region" DisplayName="Reason for Sales" />
       <Series Format="" Column="ProductType" DisplayName="Product" />
       <Values>
          <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="Sales
Amount" />
       </Values>
    </Chart>
  </Dashboard>

   <!-- Dashboard Layout Configuration -->
   <DashboardLayout>
     <Landscape>
        <!-- Dashboard display in Landscape Orientation -->
        <!-- Column Chart occupy 60% of the screen height and 100% width -->
        <Layout Name="SalesOrderCount" Top="0" Left="0" Width="100" Height="60" />
        <Layout Name="SalesPerformance" Top="61" Left="0" Width="100" Height="40" />
     </Landscape>
     <Portrait>
        <!-- Dashboard display in Portrait Orientation -->
        <!-- Column Chart occupy 50% of the screen height and 100% width -->
        <Layout Name="SalesOrderCount" Top="0" Left="0" Width="100" Height="50" />
        <Layout Name="SalesPerformance" Top="51" Left="0" Width="100" Height="50" />
      </Portrait>
  </DashboardLayout>
</MobileAnalytic>
```

5. Click **File** > **Save.**

Implementing the MAKit Query Module

The MAKit Query module contains two Objective-C protocols: `MAQueryDelegate` and `MAQueryDataTable`.

`MAQueryDelegate` includes the `executeQuery` method, which executes the name of the query defined in the XML metadata configuration of the chart.

`MAQueryDataTable` holds the result of the query and is passed to the chart to draw the chart.

Creating the ChartDataTable Class

The `ChartDataTable` class must conform to `MAQueryDataTable` protocol and must implement all the protocol methods defined in `MAQueryDataTable`. The ChartDataTable holds the result set of the query, which is passed to the chart to draw the chart.

1. In the MAKitSample project, select **File** > **New** > **New File**.
2. Select **iOS** > **Cocoa Touch** >**Objective-C class** and click **Next.**
3. Accept `NSObject` in the **Subclass of** field and click **Next.**
4. Enter `ChartDataTable` in the **Save As** field and click **Save.**

   This creates the `ChartDataTable.h` and `ChartDataTable.m` files in the Project Navigator.
5. Click `ChartDataTable.h` and replace the existing code with:

```
#import <Foundation/Foundation.h>
#import "MAKit.h"

@interface ChartDataTable : NSObject<MAQueryDataTable> {

}
@property (nonatomic,retain) NSArray *tabularData;

@end
```

6. Click **File** > **Save.**
7. Click `ChartDataTable.m` and replace the existing code with:

```
#import "ChartDataTable.h"

@implementation ChartDataTable

@synthesize tabularData;

#pragma mark -
#pragma mark MAQueryDataTable Delegate methods

-(int)getColumnCount {

    if(self.tabularData != nil && [self.tabularData count] > 0)
    {
        NSDictionary* row = (NSDictionary*)[self.tabularData objectAtIndex:0];
        return [row count];
    }
    else
    {
        return 0;
    }

}


-(int)getRows {

    if (self.tabularData != nil) {
        return [self.tabularData count];
    }
    else {
        return 0;
    }
}


-(id)getData:(NSString*)colName atIndex:(int)rowIndex {
    if(self.tabularData != nil && [self.tabularData count] > rowIndex)
    {
        NSDictionary* row = (NSDictionary*)[self.tabularData objectAtIndex:rowIndex];
        return [row objectForKey:colName];
    }
    else
    {
        return nil;
    }

}

#pragma mark -
-(void) dealloc {

    [self.tabularData release];
    [super dealloc];

}

@end
```

8. Click **File** > **Save.**

Creating the ChartQuery Class

The `ChartQuery` class must conform to `MAQueryDelegate` protocol and must implement the `executeQuery` method to execute the query associated with the chart. (The name of the query is configured in the XML metadata in the Query attribute of the chart tag.)

1. In the MAKitSample project, select **File** > **New** > **New File.**

2. Select **iOS** > **Cocoa Touch** >**Objective-C class** and click **Next.**
3. Accept `NSObject` in the **Subclass of** field and click **Next.**
4. Enter `ChartQuery` in the **Save As** field and click **Save.**

This creates the `ChartQuery.h` and `ChartQuery.m` files in the Project Navigator.

5. Click `ChartQueryClass.h` and replace the existing code with:

```
#import <Foundation/Foundation.h>
#import "MAKit.h"

@interface ChartQuery : NSObject<MAQueryDelegate> {

}

@end
```

6. Click **File** > **Save.**
7. Click `ChartQuery.m` and replace the existing code with:

```
#import "ChartQuery.h"
#import "ChartsDatasource.h"
#import "ChartDataTable.h"

@implementation ChartQuery

#pragma mark -
#pragma mark MAQueryDelegate method

-(id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:
(NSDictionary*)args {

    ChartDataTable *dataTable = [[ChartDataTable alloc] init];

    if ([queryName isEqualToString:@"OrderCount"]) {

        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        dataTable.tabularData = [cds NumberOfOrders];
        [cds release];

    }
    else if ([queryName isEqualToString:@"SalesPerf"]) {

        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        dataTable.tabularData = [cds SalesPerformance];
        [cds release];

    }
    return dataTable;
}

@end
```

8. Click **File** > **Save.**

Creating the User Interface for the Application

Configuring MAKitSampleAppDelegate Files

The `MAKitSampleAppDelegate.h` and `MAKitSampleAppDelegate.m` files are created when you create the Xcode project, but you must add the view controller property to the header file and create the view controller instance in the implementation file to display the initial view.

1. Click `MAKitSampleAppDelegate.h` and replace the existing code with the code below to add the view controller property to the application delegate.

```
#import <UIKit/UIKit.h>

@interface MAKitSampleAppDelegate : NSObject <UIApplicationDelegate> {

}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UINavigationController *navController;

@property (nonatomic, retain, readonly) NSManagedObjectContext *managedObjectContext;
@property (nonatomic, retain, readonly) NSManagedObjectModel *managedObjectModel;
@property (nonatomic, retain, readonly) NSPersistentStoreCoordinator
*persistentStoreCoordinator;

- (void)saveContext;
- (NSURL *)applicationDocumentsDirectory;
- (void) copySampleDbToDevice;

@end
```

2. Click **File** > **Save.**

3. Click `MAKitSampleAppDelegate.m` and replace the portions of existing code as indicated below, to create an instance of view controller property and to copy the database to the device:

```
#import "MAKitSampleAppDelegate.h"

@implementation MAKitSampleAppDelegate


@synthesize window=_window;
@synthesize navController;
@synthesize managedObjectContext=__managedObjectContext;
@synthesize managedObjectModel=__managedObjectModel;
@synthesize persistentStoreCoordinator=__persistentStoreCoordinator;

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *) launchOptions
{
    //Copy database to device
    [self copySampleDbToDevice];

    //Create & Add view controller instance
    [self.window addSubview:navController.view];
    [self.window makeKeyAndVisible];
    return YES;
}
```

```
//Copy MAKitTutorials.sqlite database to device
- (void) copySampleDbToDevice {
    NSFileManager *fm = [NSFileManager defaultManager];
    NSError *copyError;

    NSArray *pathArray = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory ,
NSUserDomainMask, YES);
    NSString *appDocDir = [paths objectAtIndex:0];
    NSString *deviceDBPath = [appDocDir
stringByAppendingPathComponent:@"MAKitTutorials.sqlite"];

    BOOL dbExist = [fm fileExistsAtPath:deviceDBPath];

    if(!dbExist) {

            NSString *sourceDBPath = [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"MAKitTutorials.sqlite"];
            dbExist = [fm copyItemAtPath:sourceDBPath toPath:deviceDBPath
error:&copyError];

            if (!dbExist)
                    NSLog(@"Failed to Copy Sample Database to device/simulator. Error
Msg = '%@'.", [copyError localizedDescription]);
    }

}
```

4. Click **File** > **Save.**

Adding the MainViewController

Create a MainViewController view controller, which functions as the main screen for the MAKitSample application.

1. In the MAKitSample project, select **File** > **New** > **New File.**
2. Select **iOS** > **Cocoa Touch** > **UIViewController subclass** and click **Next.**
3. Select `UITableViewController` in the **Subclass of** field, unselect **With XIB for user interface**, select **Targeted for iPad**, then click **Next.**
4. Enter `MainViewController` in the **Save As** field and click **Save.**

   This creates the `MainViewController.h` and `MainViewController.m` files in the Project Navigator.
5. Click `MainViewController.h` and replace the existing code with:

```
#import <UIKit/UIKit.h>

@interface MainViewController : UITableViewController {

}

@property (nonatomic, retain) NSMutableArray *viewControllers;

@end
```

6. Click **File** > **Save.**
7. Click `MainViewController.m` and replace the existing code with:

```
#import "MainViewController.h"
#import "ChartQuery.h"
#import "MAKit.h"

@implementation MainViewController
@synthesize viewControllers;

- (void)dealloc
{
    [super dealloc];
}
#pragma mark - View lifecycle
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.viewControllers = [[NSMutableArray alloc] init];

    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    [dict setObject:@"MAKit Basic Charts" forKey:@"Title"];
    [dict setObject:@"SalesMetadata" forKey:@"MetaData"];
    [self.viewControllers addObject:dict];
    [dict release];
}

- (void)viewDidUnload
{
    [super viewDidUnload];
    self.viewControllers = nil;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
interfaceOrientation
{
    // Return YES for supported orientations
    return YES;
}
#pragma mark - Table view data source
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return [self.viewControllers count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell...
    NSMutableDictionary *dict = [self.viewControllers objectAtIndex:indexPath.row];
    cell.textLabel.text = [dict objectForKey:@"Title"];
    cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;

    return cell;
}
```

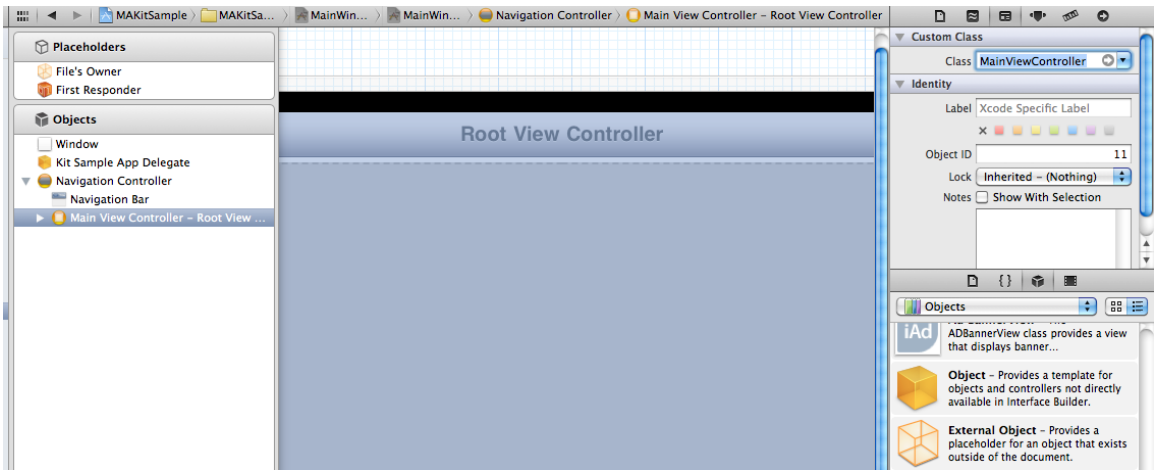
```
#pragma mark - Table view delegate
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath
{
    NSMutableDictionary *dict = [self.viewControllers objectAtIndex:indexPath.row];
    NSString *metaFile = [dict objectForKey:@"MetaData"];

    //Create MAViewController
    ChartQuery *query = [[ChartQuery alloc] init];
    MAViewController *mv = [[MAViewController alloc] init];
    mv.metaDataPath = [[NSBundle mainBundle] pathForResource:metaFile ofType:@"xml"];
    mv.theme = [[[MAKitTheme_WelterWeight alloc] init] autorelease];
    mv.dataSource = query;

    [self.navigationController pushViewController:mv animated:YES];
    [query release];
    [mv release];

}

@end
```

8. Click **File** > **Save.**

Making Connections

Add the Navigation Controller to the `MainWindow.xib` file, and connect the `navController` outlet defined in `MAKitSampleAppDelegate` to the Navigation Controller.

1. Click the `MainWindow.xib` file and, if the Objects Library pane is not already visible, select **View** > **Utilities** > **Object Library.**
2. Drag and drop the Navigation Controller onto the Objects pane below the MAKit Sample App Delegate icon.
3. Control-drag from the **MAKit Sample AppDelegate** icon to the new **Navigation Controller** icon to create a navController outlet.
4. Click the expansion arrow on the **Navigation Controller** icon, select **Navigation Controller** > **View Controller**, and in the identity inspector, select `MainViewController` in the **Class** field.



The ViewController name in the hierarchy changes to `MainViewController`, and the connection from the AppDelegate to the Navigation Controller is created.

5. Click **File** > **Save.**

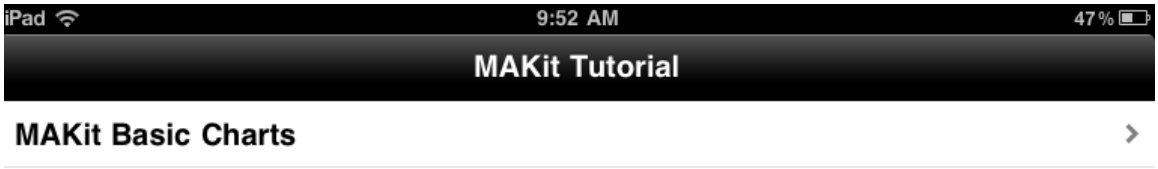
Deploying the Application

Deploy the MAKitSample application to the iPad Simulator for testing.

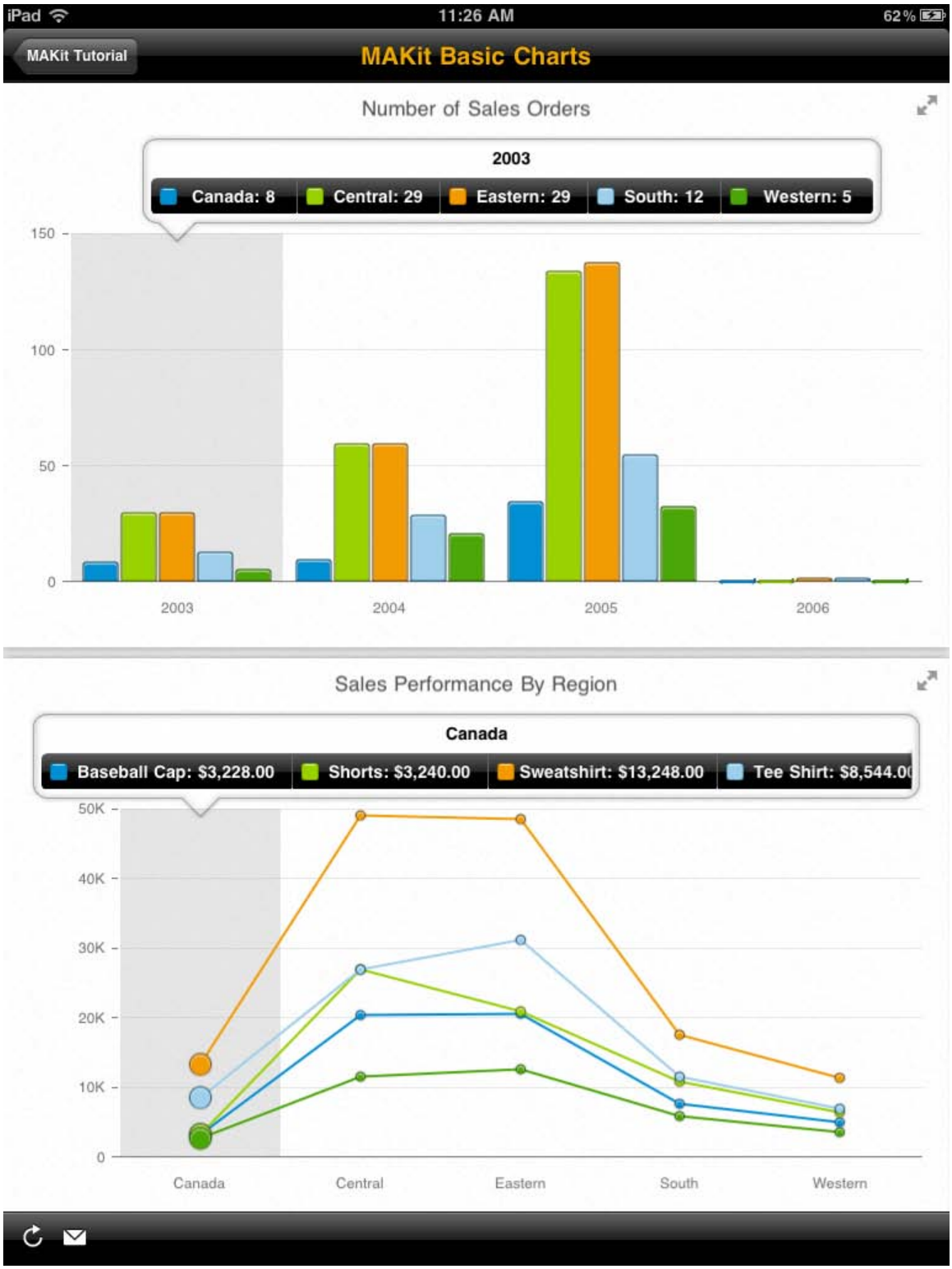
1. In the Xcode MAKitSample project, verify that the **Scheme** is set to **MAKitSample > iPad 4.3 Simulator**, and select **Product** > **Clean.**
2. Select **Product** > **Build.**

   If you correctly followed all the steps in the tutorial, you see a **`Build Succeeded`** message.
3. Select **Product** > **Run.**

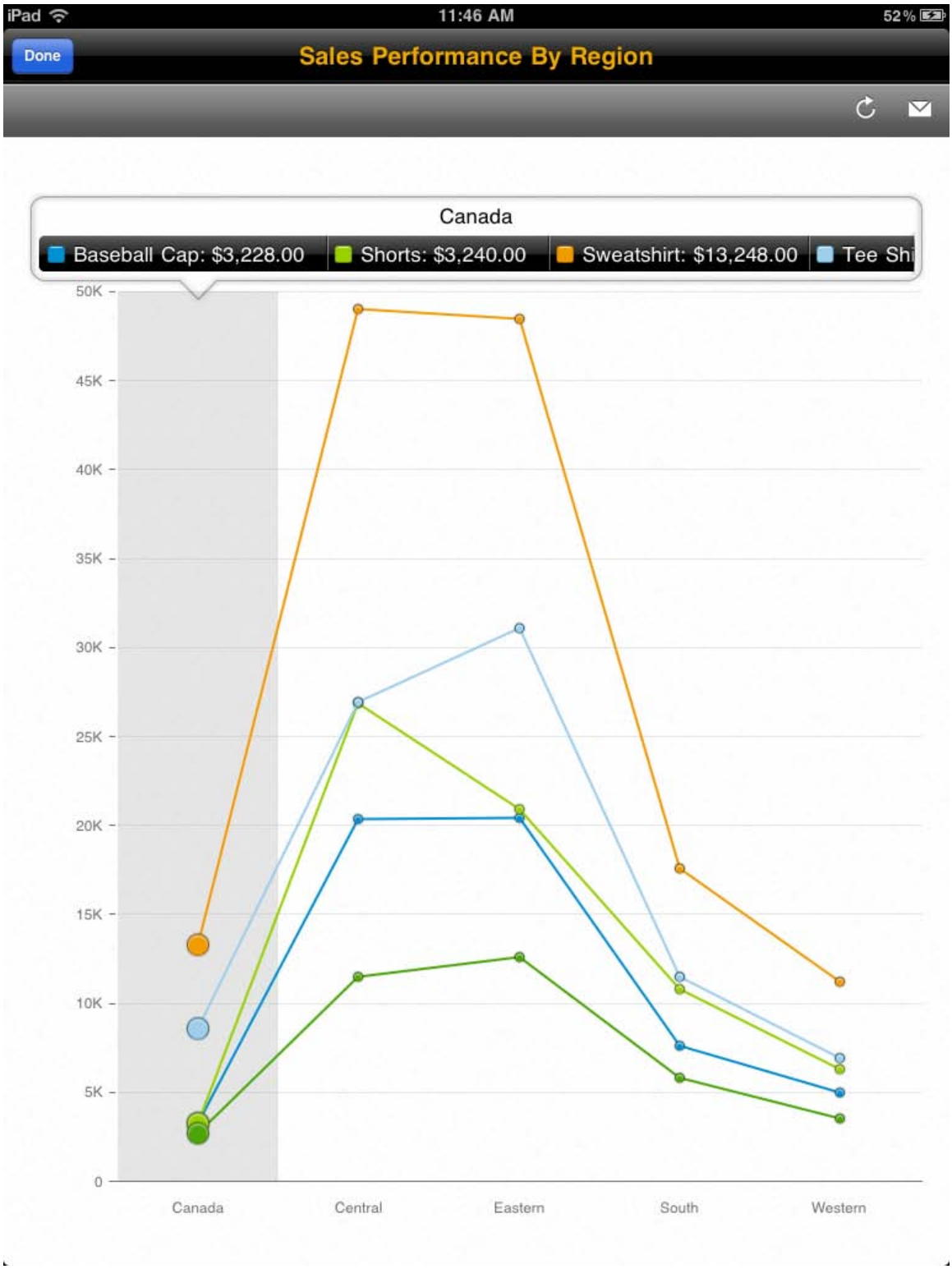
   The project is rebuilt, and the iPad Simulator starts and loads the application. You see the initial screen:



4. Click **MAKit Basic Charts**.



5. Click the **arrow image** to the right to load the chart in full screen.



6. Click **Done** to return to the previous (dashboard) view.

Lesson 3: Building Semantic Zooming Charts Using Metadata

This lesson creates a sales performance zooming chart. At the highest level, the chart depicts sales performance in North America and Europe, by year. The chart includes user interface controls that allow you to zoom into lower levels, for example, to see sales performance by quarter and by month.

This lesson describes:

- Configuring multiple zooming levels for a chart in metadata
- Writing queries for each zooming level in datasource class
- How semantic zooming works at runtime

Adding Methods to the ChartsDataSource Class

To create a zooming chart that has three levels (year, quarter, and month), create three methods—one for each level.

1. In the Xcode IDE, open `MAKitSample.xcodeproj`.
2. In the Project Navigator, click `ChartsDatasource.h` file and add these method definitions:

```
#import <Foundation/Foundation.h>
#import "sqlite3.h"

@interface ChartsDatasource : NSObject {
    sqlite3 *dbInstance;
}
...

//Zooming Charts Datasource methods
-(NSArray *) SalesByYear;
-(NSArray *) SalesByQuarter;
-(NSArray *) SalesByMonth;
```

3. Click **File** > **Save**.
4. Click `ChartDatasource.m` file and add the method implementation code for the three methods:

```
#import "ChartsDatasource.h"

@implementation ChartsDatasource

...

//Zooming Charts Datasource methods
//SalesByYear Method
-(NSArray *) SalesByYear {
    NSMutableArray *returnArray = [[NSMutableArray alloc] init];

    const char * SELECTQry = "SELECT SalesYear, RegionGroup,Sum(SalesAmount) "
                " FROM ResellerSales WHERE SalesYear>2001 AND RegionGroup <>'Pacific'"
                " GROUP BY SalesYear,RegionGroup ";

    sqlite3_stmt *selectStmt = nil;

    if (sqlite3_prepare_v2(dbInstance, SELECTQry, -1, &selectStmt, NULL)==SQLITE_OK) {
        NSString *year  = @"";
        NSString *region = @"";
        float salesamount = 0.0;
       while (sqlite3_step(selectStmt)==SQLITE_ROW) {
            year = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 0)];
            region = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 1)];
           salesamount = sqlite3_column_double(selectStmt, 2);

            NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
            [dict setObject:year forKey:@"SalesYear"];
            [dict setObject:region forKey:@"Region"];
            [dict setObject:[NSNumber numberWithFloat:salesamount]
forKey:@"SalesAmount"];
            [returnArray addObject:dict];
            [dict release];
           }
    }
    else {
           NSLog(@"Failed at Prepare Statement in SalesByYear.
Reason %s",sqlite3_errmsg(dbInstance));
    }
    return returnArray;
}

//SalesByQuarter Method
-(NSArray *) SalesByQuarter {
    NSMutableArray *returnArray = [[NSMutableArray alloc] init];

    const char * SELECTQry = "SELECT SalesYear, SalesQuarter, RegionGroup, "
                             " Sum(SalesAmount) FROM ResellerSales "
                             " WHERE SalesYear>2001 AND RegionGroup <>'Pacific'"
                             " GROUP BY SalesYear,SalesQuarter,RegionGroup ";

    sqlite3_stmt *selectStmt = nil;

    if (sqlite3_prepare_v2(dbInstance, SELECTQry, -1, &selectStmt, NULL)==SQLITE_OK) {
        NSString *year  = @"";
        NSString *qtr = @"";
        NSString *region = @"";
        float salesamount = 0.0;
        while (sqlite3_step(selectStmt)==SQLITE_ROW) {
            year = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 0)];

            qtr = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 1)];

            region = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 2)];

                    salesamount = sqlite3_column_double(selectStmt, 3);
```

```
            NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
            [dict setObject:year forKey:@"SalesYear"];
            [dict setObject:qtr forKey:@"SalesQuarter"];
            [dict setObject:region forKey:@"Region"];
            [dict setObject:[NSNumber numberWithFloat:salesamount]
forKey:@"SalesAmount"];
            [returnArray addObject:dict];

            [dict release];
           }
    }
    else {
           NSLog(@"Failed at Prepare Statement in SalesByQuarter.
Reason %s",sqlite3_errmsg(dbInstance));
    }
    return returnArray;

}

//SalesByMonth Method
-(NSArray *) SalesByMonth{
    NSMutableArray *returnArray = [[NSMutableArray alloc] init];

  const char * SELECTQry = "SELECT SalesYear, SalesQuarter, SalesMonth, RegionGroup, "
                           " Sum(SalesAmount) FROM ResellerSales "
                           " WHERE SalesYear>2001 AND RegionGroup <>'Pacific'"
                           " GROUP BY SalesYear,SalesQuarter,SalesMonth,RegionGroup ";

    sqlite3_stmt *selectStmt = nil;
    if (sqlite3_prepare_v2(dbInstance, SELECTQry, -1, &selectStmt, NULL)==SQLITE_OK) {
        NSString *year  = @"";
        NSString *qtr = @"";
        NSString *mth = @"";
        NSString *region = @"";
        float salesamount = 0.0;
        while (sqlite3_step(selectStmt)==SQLITE_ROW) {
           year = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 0)];
           qtr = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 1)];
           mth = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 2)];
           region = [NSString stringWithUTF8String:(char *)
sqlite3_column_text(selectStmt, 3)];
           salesamount = sqlite3_column_double(selectStmt, 4);

            NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
            [dict setObject:year forKey:@"SalesYear"];
            [dict setObject:qtr forKey:@"SalesQuarter"];
            [dict setObject:mth forKey:@"SalesMonth"];
            [dict setObject:region forKey:@"Region"];
            [dict setObject:[NSNumber numberWithFloat:salesamount]
forKey:@"SalesAmount"];
            [returnArray addObject:dict];

            [dict release];
           }
    }
    else {
           NSLog(@"Failed at Prepare Statement in SalesByMonth.
Reason %s",sqlite3_errmsg(dbInstance));
    }
    return returnArray;
}

@end
```

5. Click **File** > **Save**.

Creating XML Metadata File with Zooming Levels

Next, create a new XML metadata file with which to configure the zooming chart. Zoom behavior for each level is configured in the *<ZoomingLevels> … </ZoomingLevels>* tag in the metadata.

Configure the first zooming level to display the sales performance by year:

1. In the MAKitSample project, click **File** > **New** > **New File.**
2. Select **iOS** > **Other** > **Empty** and click **Next.**
3. Enter `ZoomingChartMetadata.xml` in the **Save As** field and click **Save.**

   This creates the `ZoomingChartMetadata.xml` file in the Project Navigator.
4. Click `ZoomingChartMetadata.xml` and add this content to the file:

```
<?xml version="1.0" encoding="utf-8"?>
<MobileAnalytic MAKitVersion="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MAKitMetaFile.xsd">
    <Dashboard>

      <!-- Zooming Chart configuration for Sales Performance ByYear/Qtr/Month -->
      <Chart Name="ZoomingChart" Title="Sales Performance" ChartType="Column" >

         <Category Format=""  DisplayName="Sales ByYear-Quarter-Month" />
         <Series Format="" Column="Region" DisplayName="Sales Region" />
         <Values>
            <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="" />
         </Values>

          <!-- Zooming Levels Configuration -->
          <ZoomingLevels>
            <ZoomingLevel Query="SalesByYear" CategoryColumn="SalesYear" Title="ByYear" />
            <ZoomingLevel Query="SalesByQuarter" CategoryColumn="SalesQuarter" Title="ByQuarter" />
            <ZoomingLevel Query="SalesByMonth" CategoryColumn="SalesMonth" Title="ByMonth" />
          </ZoomingLevels>

        </Chart>

    </Dashboard>
    <DashboardLayout>
        <Landscape>
            <!-- Dashboard display in Landscape Orientation -->
            <Layout Name="ZoomingChart" Top="0" Left="0" Width="100" Height="80" />
        </Landscape>

        <Portrait>
            <!-- Dashboard display in Portrait Orientation -->
            <Layout Name="ZoomingChart" Top="0" Left="0" Width="100" Height="85" />
        </Portrait>
    </DashboardLayout>
</MobileAnalytic>
```

5. Click **File** > **Save.**

Adding Scripts in ChartQuery and ViewController Classes

Modifying ChartQuery Class

Add code to the `ChartQuery` class to execute the appropriate datasource query method, depending on the zooming level.

1. In the Project Navigator, click the `ChartsQuery.m` file.
2. Append this code to the `executeQuery` method with the existing script in the method:

```
#import "ChartQuery.h"
#import "ChartsDatasource.h"
#import "ChartDataTable.h"

@implementation ChartQuery

#pragma mark -
#pragma mark MAQueryDelegate method

-(id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:
(NSDictionary*)args {

    ChartDataTable *dataTable = [[ChartDataTable alloc] init];


    ...

    else if ([queryName isEqualToString:@"SalesByYear"]) {

        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        dataTable.tabularData = [cds SalesByYear];
        [cds release];

    }
    else if ([queryName isEqualToString:@"SalesByQuarter"]) {

        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        dataTable.tabularData = [cds SalesByQuarter];
        [cds release];

    }
    else if ([queryName isEqualToString:@"SalesByMonth"]) {

        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        dataTable.tabularData = [cds SalesByMonth];
        [cds release];

    }
    return dataTable;
}

@end
```

3. Click **File** > **Save**.

Adding Script to MainViewController Class

1. In the Project Navigator, click the `MainViewController.m` file.
2. Add this code to the `ViewDidLoad` method with the existing script in the method:

```
- (void)viewDidLoad
{
    ...

    //Zooming Chart
    dict = [[NSMutableDictionary alloc] init];
    [dict setObject:@"MAKit Zooming Chart" forKey:@"Title"];
    [dict setObject:@"ZoomingChartMetadata" forKey:@"MetaData"];
    [self.viewControllers addObject:dict];
    [dict release];

}
```

3. Click **File** > **Save**.

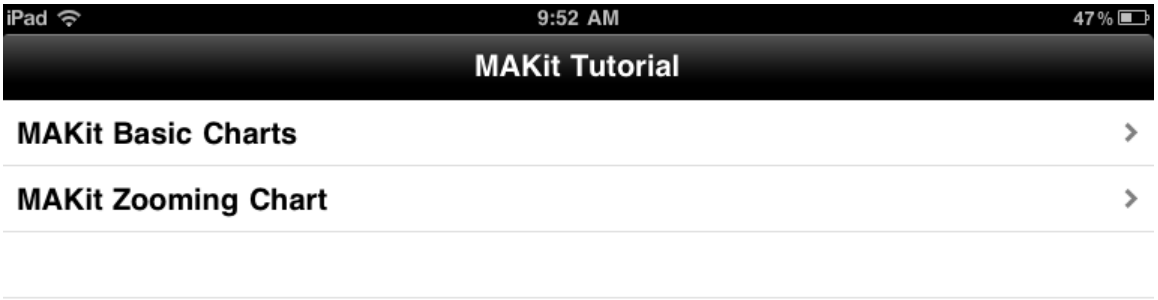
Deploying the Application

Deploy the MAKitSample application to the iPad Simulator for testing.

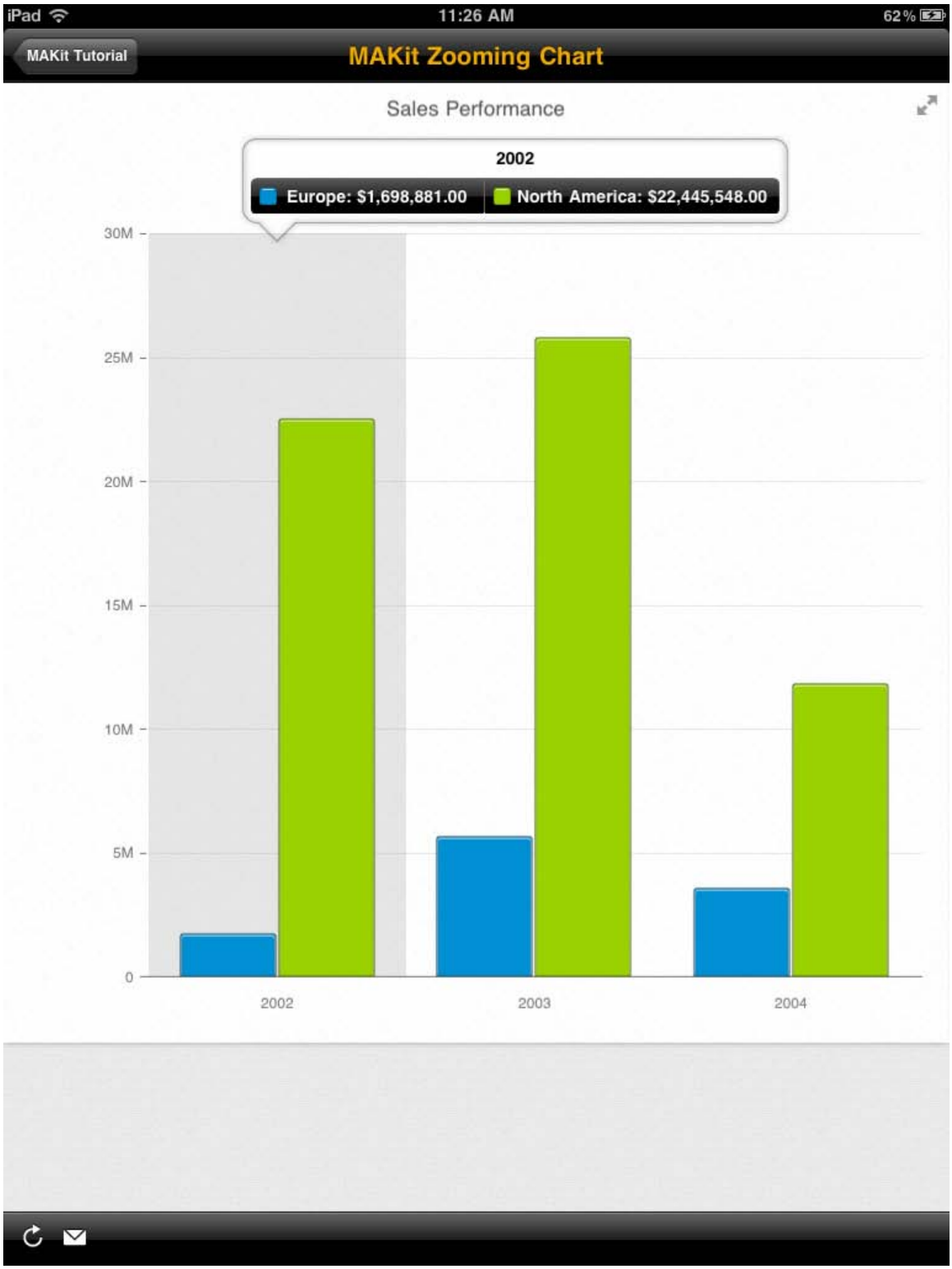
1. In the Xcode MAKitSample project, verify that the **Scheme** is set to **MAKitSample > iPad 4.3 Simulator**, and select **Product** > **Clean.**
2. Select **Product** > **Build.**

   If you correctly followed all the steps in the tutorial, you see a **`Build Succeeded`** message.
3. Select **Product** > **Run**.

   The project is rebuilt and iPad Simulator starts and loads the application. The initial screen now contains additional entry for "Zooming Chart."

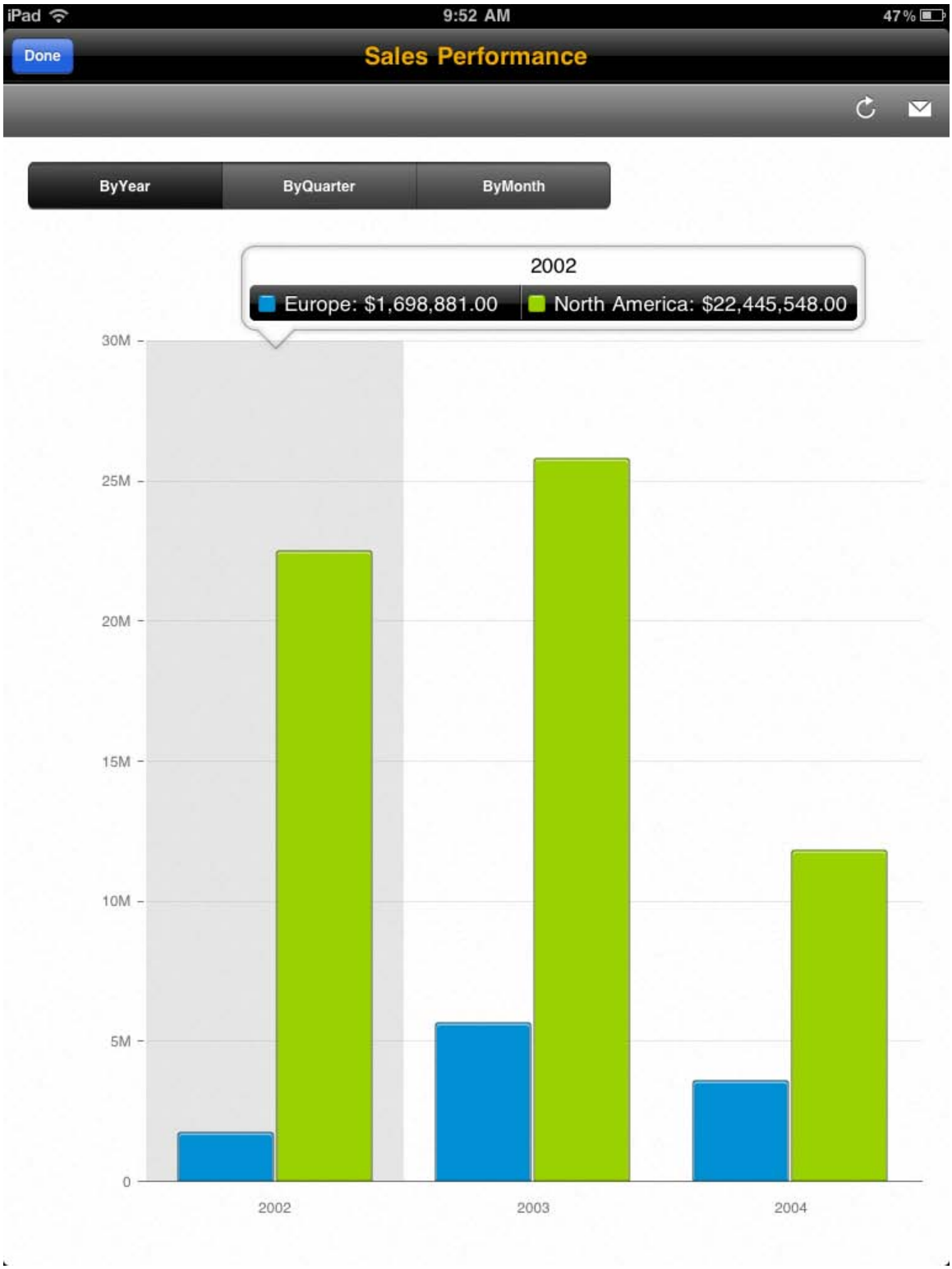


4. Click **MAKit Zooming Chart**.

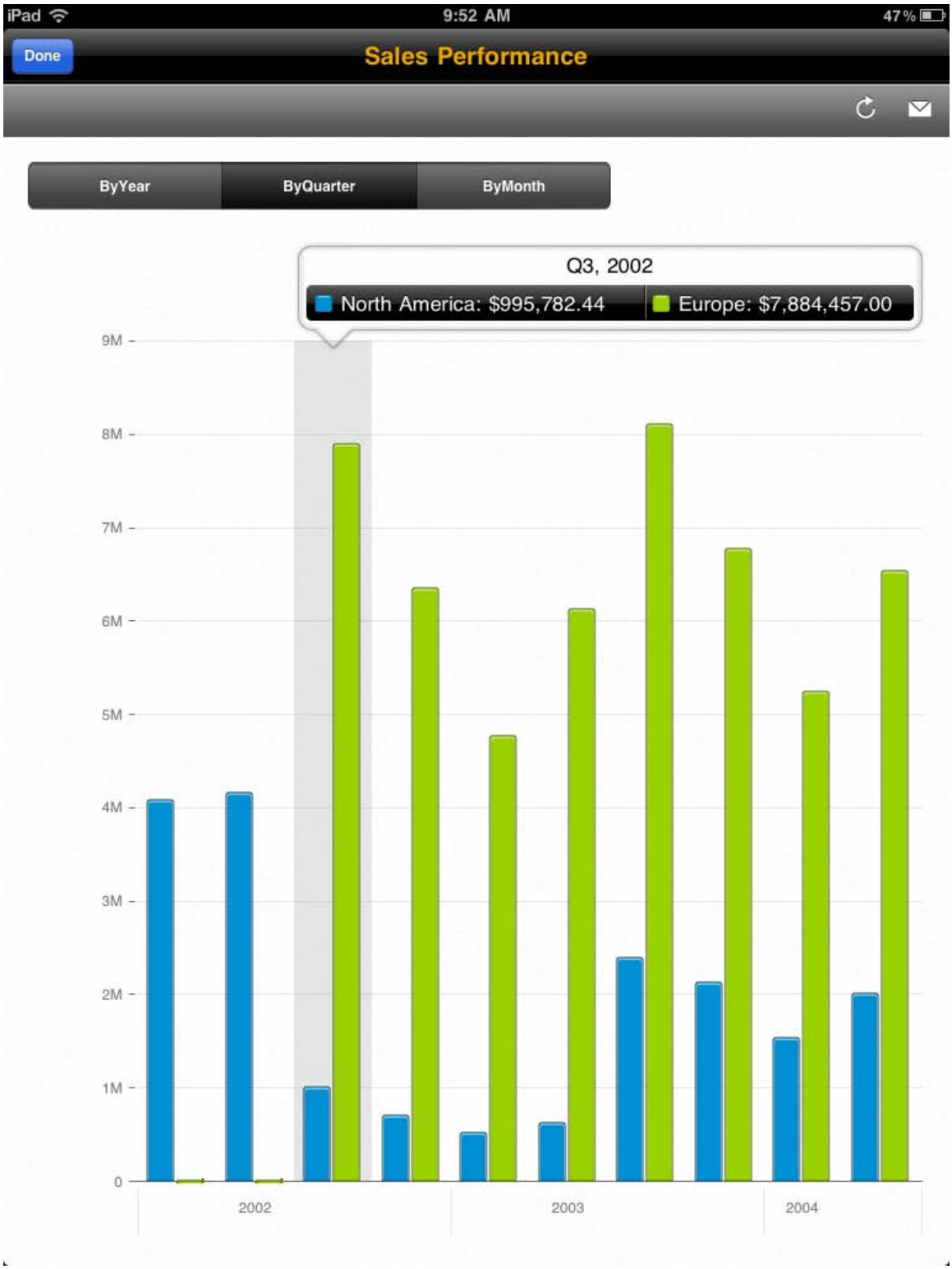


5. Click the **arrow image** in the upper-right corner to load the chart in full screen.

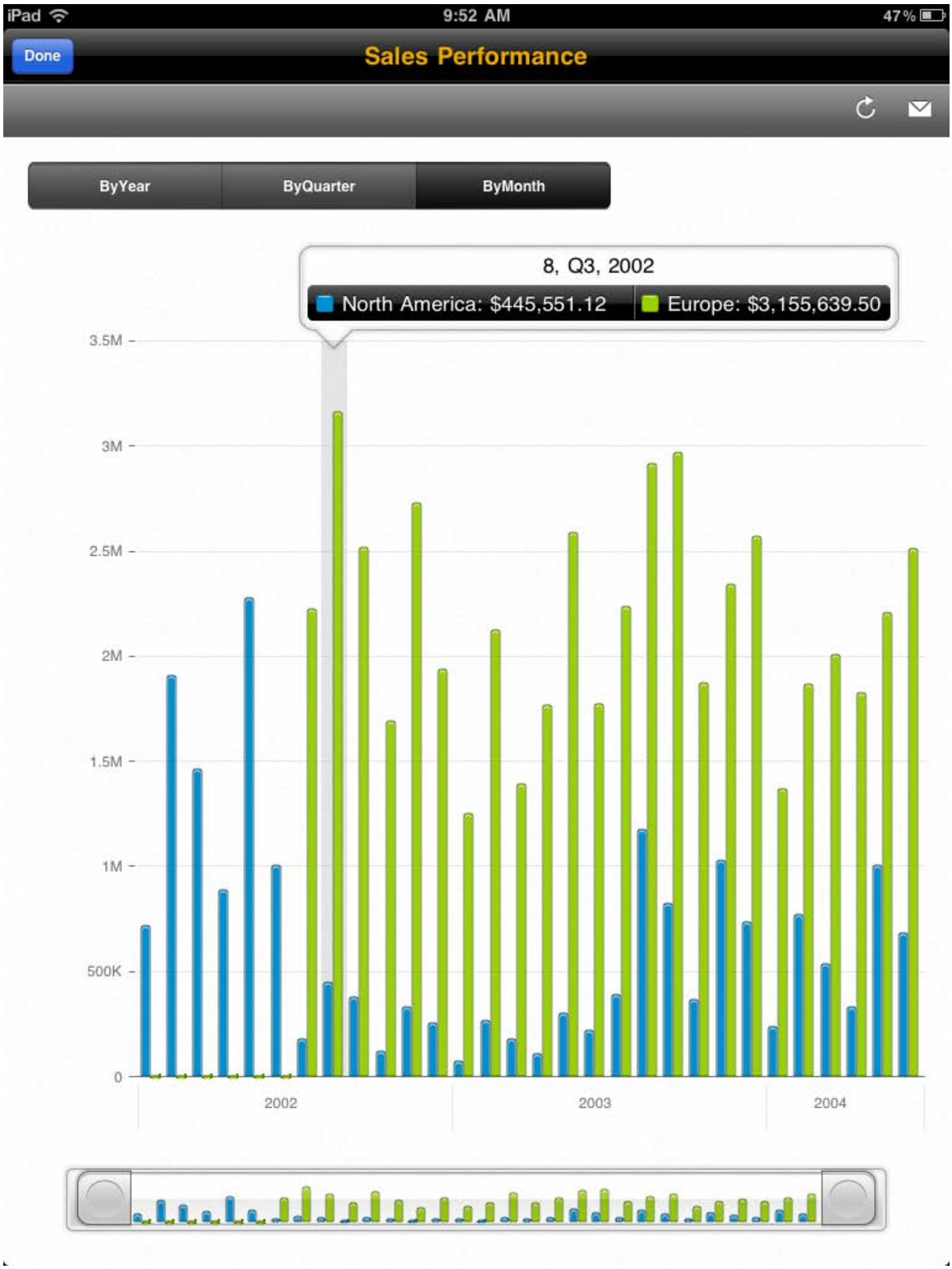
   The zooming chart loads in full screen with zooming level 1 (Sales ByYear) selected:



6. Click **ByQuarter**.



7. Click **ByMonth**.


iPad
9:52 AM
47%
Done
Sales Performance
ByYear
ByQuarter
ByMonth
8, Q3, 2002
North America: $445,551.12
Europe: $3,155,639.50
3.5M
3M
2.5M
2M
1.5M
1M
500K
0
2002
2003
2004

Lesson 4: Building Drilldown Charts Using Metadata

This lesson describes how to create three levels of drilldown chart on reseller sales data. At the highest level, the chart shows reseller sales, aggregated by product category. Tapping a particular product category loads a sales performance for the selected product category grouped by product subcategory in three regions. Tapping a particular region loads a detailed chart for that region.

This lesson describes:

- Configuring the metadata for a drilldown group with three levels in a drilldown chart
- Creating datasource queries for drilldown charts by passing query parameters
- User interactions in drilldown charts

Adding Methods to the ChartsDataSource Class

To create a drilldown chart with three levels of drilldown, create three methods: one to retrieve summary data, one to retrieve data based on selected product category, and one to retrieve data for a selected product category and region.

1. Open the `MAKitSample.xcodeproj` in the Xcode IDE.
2. In the Project Navigator, click the `ChartsDatasource.h` file and add these method definitions:

```
#import <Foundation/Foundation.h>
#import "sqlite3.h"

@interface ChartsDatasource : NSObject {
    sqlite3 *dbInstance;
}
...

//Drilldown Charts Datasource methods
- (NSArray *) ResellerSummary;
- (NSArray *) ResellerDataForProduct:(NSString *) product;
- (NSArray *) ResellerDataForProduct:(NSString *) product andRegion:(NSString *) region;
```

3. Click **File** > **Save**.
4. Click `ChartDatasource.m` file and add the method implementation code for the methods:

```
#import "ChartsDatasource.h"

@implementation ChartsDatasource

...

//Drilldown Charts Datasource methods
//ResellarSummary Data
-(NSArray *) ResellerSummary {

 const char* SELECT="SELECT ProductCategory, RegionGroup, round(sum( SalesAmount),2) "
                    " FROM ResellerSales Group By ProductCategory,RegionGroup ";

   NSMutableArray *data = [[NSMutableArray alloc] init];
   sqlite3_stmt *selectStmt = nil;
   if (sqlite3_prepare_v2(dbInstance, SELECT, -1, &selectStmt, NULL)==SQLITE_OK) {
           NSString *pc=@"";
           NSString *rg=@"";
          double amount =0;
        while (sqlite3_step(selectStmt)==SQLITE_ROW) {
          pc = [NSString stringWithUTF8String:(char *) sqlite3_column_text(selectStmt, 0)];
         rg = [NSString stringWithUTF8String:(char *) sqlite3_column_text(selectStmt, 1)];
         amount = sqlite3_column_double(selectStmt, 2);

         NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
        [dict setObject:pc forKey:@"ProductCategory"];
        [dict setObject:rg forKey:@"Region"];
        [dict setObject:[NSNumber numberWithDouble:amount] forKey:@"SalesAmount"];
        [data addObject:dict];
        [dict release];
     }
         }
  else {
           NSLog(@"Failed at Prepare Statement in ResellerSummary.
Reason %s",sqlite3_errmsg(dbInstance));
         }
         return data;
}

//ResellerData for specific product
- (NSArray *) ResellerDataForProduct:(NSString *) product {

    const char * SELECT = "SELECT  RegionGroup, ProductSubCategory, "
                "round(sum(SalesAmount),2)  FROM ResellerSales "
                " WHERE ProductCategory = ? GROUP BY RegionGroup, ProductSubCategory";


    NSMutableArray *data = [[NSMutableArray alloc] init];
    sqlite3_stmt *selectStmt = nil;
  if (sqlite3_prepare_v2(dbInstance, SELECT, -1, &selectStmt, NULL)==SQLITE_OK) {

       sqlite3_bind_text(selectStmt, 1, [product UTF8String], -1, SQLITE_STATIC);

          NSString *region=@"";
        NSString *psc=@"";
          double amount =0;

     while (sqlite3_step(selectStmt)==SQLITE_ROW) {
      region = [NSString stringWithUTF8String:(char *)sqlite3_column_text(selectStmt,0)];
      psc = [NSString stringWithUTF8String:(char *)sqlite3_column_text(selectStmt, 1)];
      amount = sqlite3_column_double(selectStmt, 2);

       NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
       [dict setObject:region forKey:@"Region"];
       [dict setObject:psc forKey:@"ProductSubCategory"];
       [dict setObject:[NSNumber numberWithDouble:amount] forKey:@"SalesAmount"];
       [data addObject:dict];
         [dict release];
        }
        }
    else {
       NSLog(@"Failed at Prepare Statement in ResellerProductByRegion. Reason %s",
sqlite3_errmsg(dbInstance));
  }
  return data;
}

//ResellerData for Selected product & region
- (NSArray *) ResellerDataForProduct:(NSString *) product andRegion:(NSString *) region {
```

```
    const char * SELECT = "SELECT Country,ProductSubCategory,round(sum(SalesAmount),2)  "
                          " FROM ResellerSales WHERE ProductCategory = ? AND RegionGroup = ? "
                          " GROUP BY Country,ProductSubCategory ";


    NSMutableArray *data = [[NSMutableArray alloc] init];
    sqlite3_stmt *selectStmt = nil;
    if (sqlite3_prepare_v2(dbInstance, SELECT, -1, &selectStmt, NULL)==SQLITE_OK) {
        sqlite3_bind_text(selectStmt, 1, [product UTF8String], -1, SQLITE_STATIC);
        sqlite3_bind_text(selectStmt, 2, [region UTF8String], -1, SQLITE_STATIC);

        NSString *country = @"";
        NSString *psc=@"";
       double amount =0;
       while (sqlite3_step(selectStmt)==SQLITE_ROW) {
          country = [NSString stringWithUTF8String:(char *) sqlite3_column_text(selectStmt, 0)];
          psc = [NSString stringWithUTF8String:(char *) sqlite3_column_text(selectStmt, 1)];
          amount = sqlite3_column_double(selectStmt, 2);

           NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
           [dict setObject:country forKey:@"Country"];
           [dict setObject:psc forKey:@"ProductSubCategory"];
           [dict setObject:[NSNumber numberWithDouble:amount] forKey:@"SalesAmount"];
           [data addObject:dict];
           [dict release];
      }
    }
  else {
  NSLog(@"Failed at Prepare Statement in ResellerProductByRegion. Reason %s",
sqlite3_errmsg( dbInstance));

         }
    return data;
}

@end
```

5. Click **File** > **Save**.

Creating XML Metadata File Configured with DrillDownGroup

Create a new XML metadata file to configure drilldown charts within the DrillDownGroup. Configure all the charts in the drilldown in the *<DrillDownGroup> … </DrillDownGroup>* tag in the metadata.

Individual drilldown levels are identified by `ChartSequence` attribute in the chart configuration.

1. In the MAKitSample project, click **File** > **New** > **New File.**
2. Select **iOS** > **Other** > **Empty** and click **Next.**
3. Enter `DrilldownChartMetadata.xml` in the **Save As** field and click **Save.**

   This creates the `DrilldownChartMetadata.xml` file in the Project Navigator.
4. Click `DrilldownChartMetadata.xml` and add this content to the file:

```
<?xml version="1.0" encoding="utf-8"?>
<MobileAnalytic MAKitVersion="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MAKitMetaFile.xsd">
  <Dashboard>

    <!-- Drilldown Chart configuration  -->
    <DrillDownGroup Name="DrilldownChart" Title="Drilldown By ProductCategory to Zooming Chart">
       <!-- Chart Sequence 1 -->
       <Chart Name="SalesSummary" Title="Reseller Sales Summary For All Regions"
                 ChartType="Column" ChartSequence="1" Query="ResellerSummary" ByCategory="1" >
          <Category Format="" Column="ProductCategory" DisplayName="ProductCategory" />
          <Series Format="" Column="Region" DisplayName="" />
          <Values>
             <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="" />
          </Values>
       </Chart>

       <!-- Chart Sequence 2 -->
       <Chart Name="SalesByRegion" Title="Reseller Sales By Region" ChartType="Column"
                                    ChartSequence="2" Query="ResaleByRegion">
           <Category Format="" Column="Region" DisplayName="RegionGroup" />
           <Series Format="" Column="ProductSubCategory" DisplayName="" />
           <Values>
             <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="" />
           </Values>
       </Chart>

       <!-- Chart Sequence 3 -->
       <Chart Name="SalesByCountry" Title="Reseller Sales By Country" ChartType="Column"
                                   ChartSequence="3" Query="ResaleByCountry">
           <Category Format="" Column="Country" DisplayName="Country" />
           <Series Format="" Column="ProductSubCategory" DisplayName="" />
           <Values>
             <Value Format="Currency" Expression="SalesAmount" ValueSequence="1" DisplayName="" />
            </Values>
       </Chart>

     </DrillDownGroup>
   </Dashboard>

   <DashboardLayout>
        <Landscape>
            <!-- Dashboard display in Landscape Orientation -->
            <Layout Name="DrilldownChart" Top="0" Left="0" Width="100" Height="80" />
        </Landscape>
        <Portrait>
            <!-- Dashboard display in Portrait Orientation -->
            <Layout Name="DrilldownChart" Top="0" Left="0" Width="100" Height="85" />
        </Portrait>
    </DashboardLayout>

</MobileAnalytic>
```

5. Click **File** > **Save.**

Adding Scripts in ChartQuery and ViewController Classes

Modifying ChartQuery Class

Add the code in `ChartQuery` class to execute the appropriate datasource query method depending on the drilldown level.

1. In the Project Navigator, click the `ChartsQuery.m` file.
2. Append this code to the `executeQuery` method with the existing script.

```
#import "ChartQuery.h"
#import "ChartsDatasource.h"
#import "ChartDataTable.h"

@implementation ChartQuery

#pragma mark -
#pragma mark MAQueryDelegate method

-(id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:
(NSDictionary*)args {

    ChartDataTable *dataTable = [[ChartDataTable alloc] init];


    ...

    else if ([queryName isEqualToString:@"ResellerSummary"]) {
        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        dataTable.tabularData = [cds ResellerSummary];
        [cds release];
    }

    else if ([queryName isEqualToString:@"ResaleByRegion"]) {
        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        NSString *product = (NSString *)[args objectForKey:@"ProductCategory"];
        dataTable.tabularData = [cds ResellerDataForProduct:product];
        [cds release];
    }

    else if ([queryName isEqualToString:@"ResaleByCountry"]) {
        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        NSString *product = (NSString *) [args objectForKey:@"ProductCategory"];
        NSString *region = (NSString *)[args objectForKey:@"Region"];

        dataTable.tabularData = [cds ResellerDataForProduct:product andRegion:
region];

        [cds release];

    }     return dataTable;
}

@end
```

3. Click **File** > **Save**.

Adding Script to MainViewController Class

1. In the Project Navigator, click the `MainViewController.m` file.
2. Add this code to the `ViewDidLoad` method with the existing script in the method:

```
- (void)viewDidLoad
{
    ...

    //DrillDown Chart
    dict = [[NSMutableDictionary alloc] init];
    [dict setObject:@"MAKit DrillDown Chart" forKey:@"Title"];
    [dict setObject:@"DrilldownChartMetadata" forKey:@"MetaData"];
    [self.viewControllers addObject:dict];
    [dict release];

}
```

3. Click **File** > **Save**.

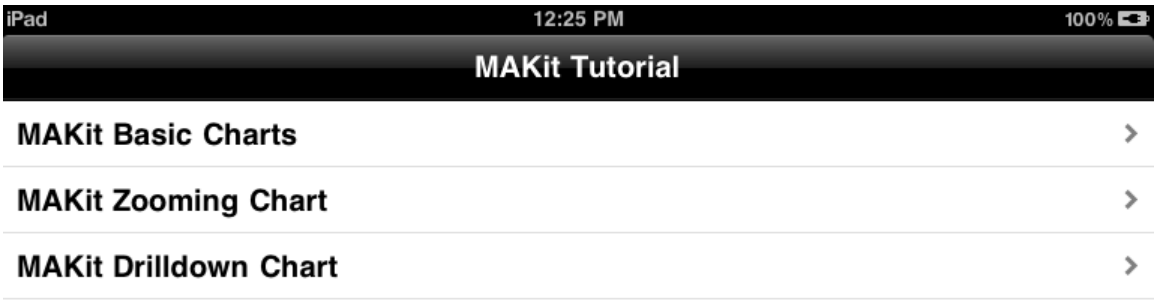
Deploying the Application

Deploy the MAKitSample application to the iPad Simulator for testing.

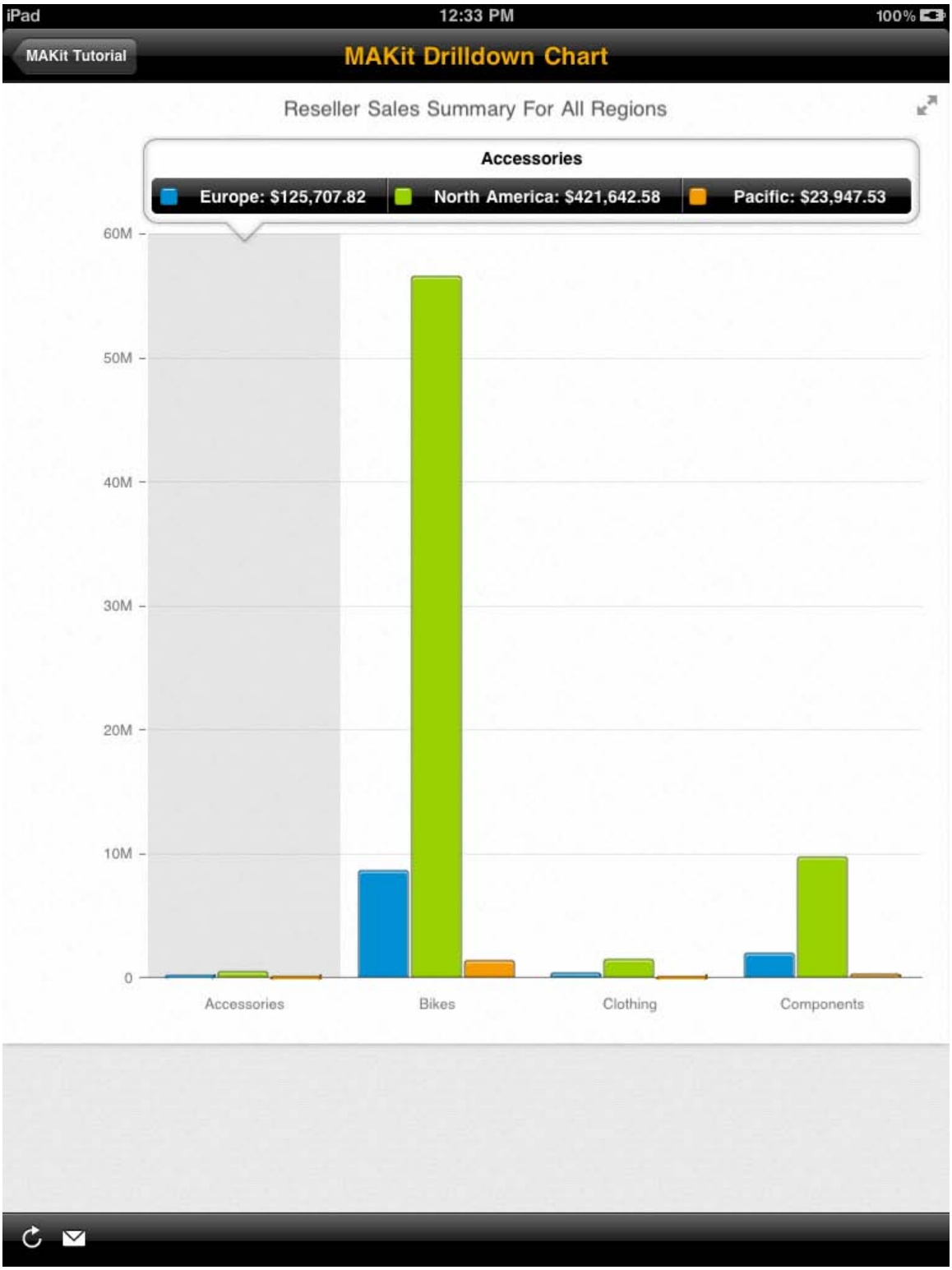
1. In the Xcode MAKitSample project, verify that the **Scheme** is set to **MAKitSample > iPad 4.3 Simulator**, and select **Product** > **Clean.**
2. Select **Product** > **Build.**

   If you correctly followed all the steps in the tutorial, you see a **`Build Succeeded`** message.
3. Select **Product** > **Run**.

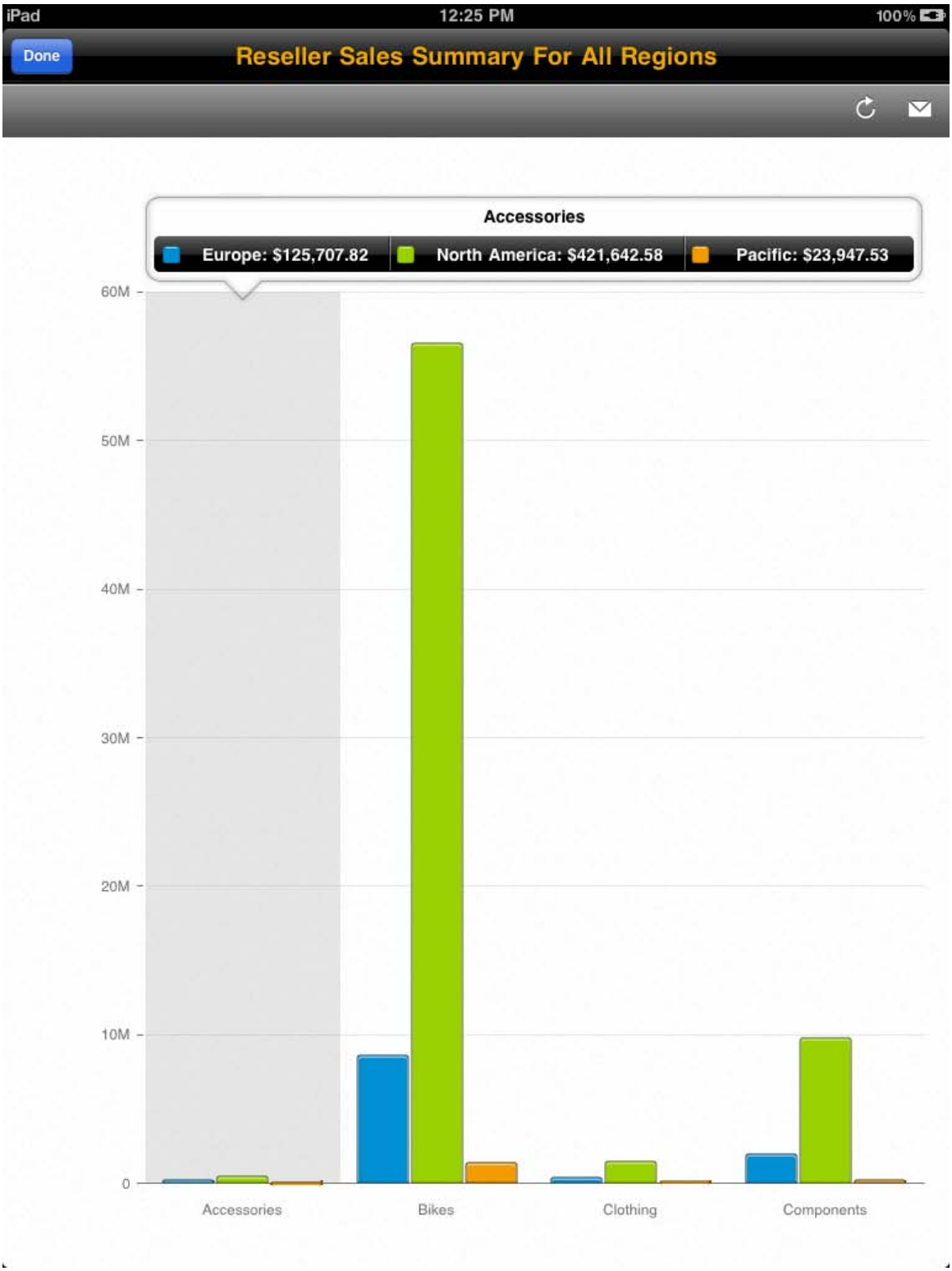
   The project is rebuilt and iPad Simulator starts and load the application. The initial screen now contains a "MAKit Drilldown Chart" row.



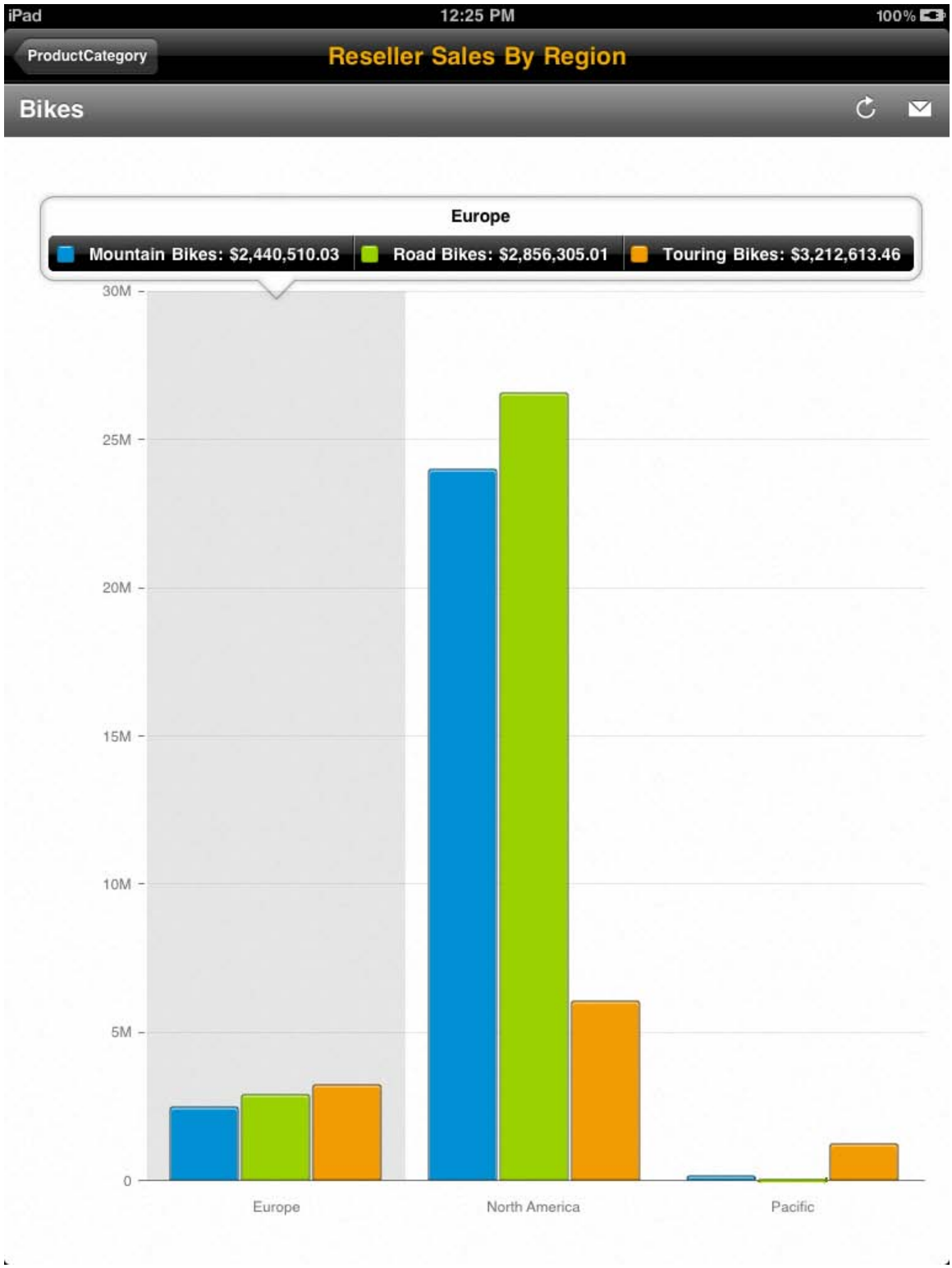
4. Click **MAKit Drilldown Chart**.



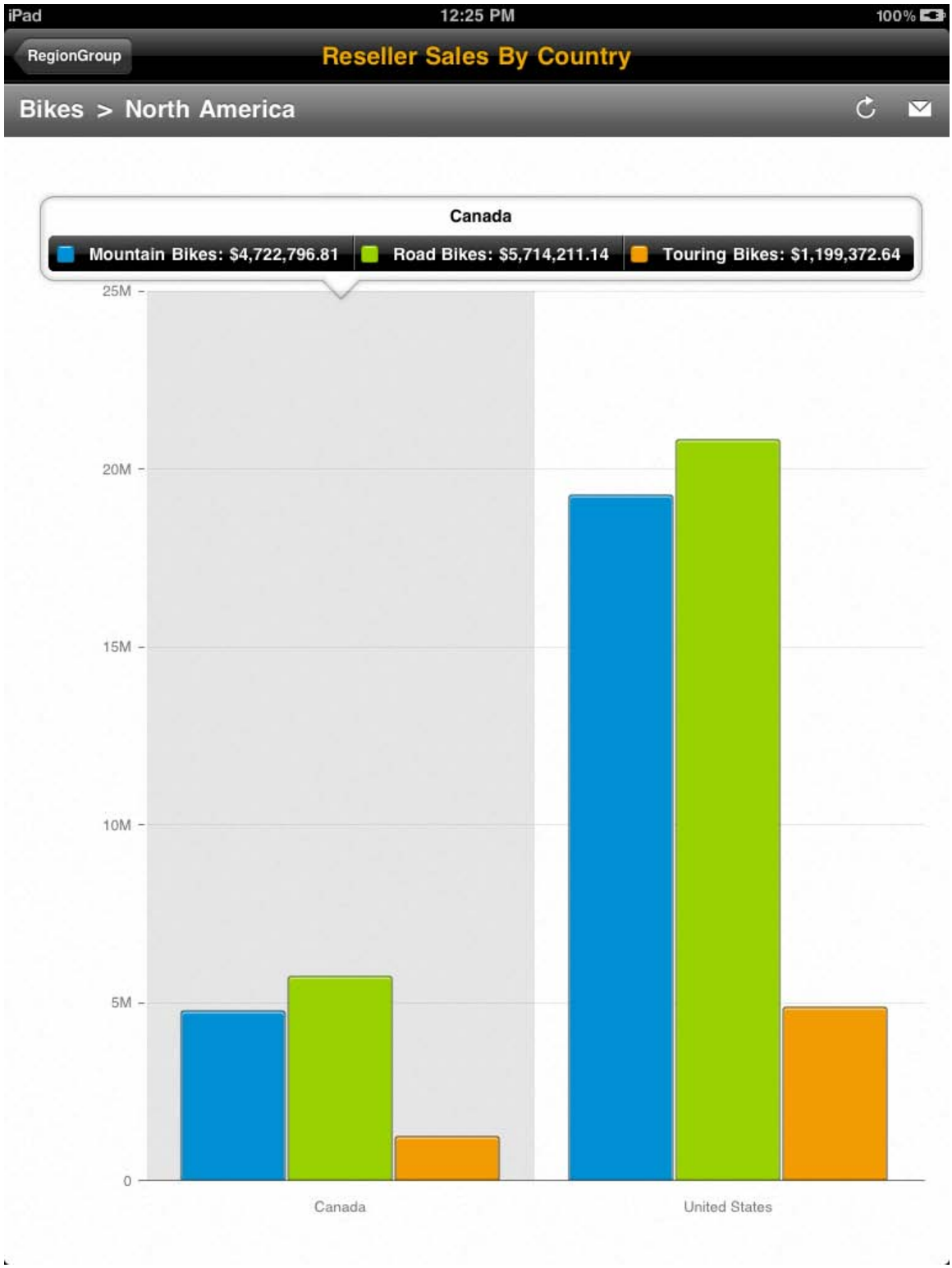
5. Click the **arrow image** in the upper-right corner to load the chart in full screen.



6. Double-tap **Bikes.**



7. Double-tap **North America.**


iPad
12:25 PM
100%
RegionGroup
Reseller Sales By Country
Bikes > North America
Canada
Mountain Bikes: $4,722,796.81
Road Bikes: $5,714,211.14
Touring Bikes: $1,199,372.64
25M
20M
15M
10M
5M
0
Canada
United States

Lesson 5: Adding Dynamic Analytics to Charts

Dynamic analytics is a powerful feature of MAKit that provides on-device analytic capability to give end users instant insight into dynamic data for effective decision making, and allow them to explore alternative scenarios and outcomes.

The core components of dynamic analytics include:

- Variable declaration
- Chart value-expression using the declared variables
- The DA Panel

This lesson describes:

- Declaring variables in metadata
- Creating expressions for chart value axis by using variables
- Creating the DA Panel and its layout
- Runtime behavior

Creating Dynamic Analytic XML Metadata Configuration

This lesson reuses the ChartDatasource method `NumberOfOrders` and the `OrderCount` query.

- Declare these two variables: `$globalpromotion` of type boolean and `$regionalsalesdrive` of type enum.
- Write a chart value expression as if `$globalpromotion` is true, double the `OrderCount` value (multiply it by 2)
- If `$globalpromotion` is false, determine which region currently has sales promotion (by checking the `$regionalsalesdrive` value) and apply the expression as `OrderCount * 6` for that region only.

1. In the MAKitSample project, click **File** > **New** > **New File.**
2. Select **iOS** > **Other** > **Empty** and click **Next.**
3. Enter `DAPanelMetadata.xml` in the **Save As** field and click **Save.**

   This creates the `DAPanelMetadata.xml` file in the Project Navigator.
4. Click `DAPanelMetadata.xml` and add this content to the file:

```
<?xml version="1.0" encoding="utf-8"?>
<MobileAnalytic MAKitVersion="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MAKitMetaFile.xsd">
  <Dashboard>

    <!-- Variable Declaration -->
    <Variables>
       <Variable Name="$globalpromotion" DisplayName="Is Global Sales Promotion On?"
                          VariableType="boolean" Value="0" />
       <Variable Name="$regionalsalesdrive" DisplayName="Sales Drive in  "
                          VariableType="enum" Selection="0">
             <Choices>
                <Choice Display="Eastern" Value="Eastern" />
                <Choice Display="Western" Value="Western" />
                <Choice Display="Central" Value="Central" />
                <Choice Display="South" Value="South" />
                <Choice Display="Canada" Value="Canada" />
             </Choices>
        </Variable>
     </Variables>

     <!-- Configure Chart Value Axis with Expression using Variables declared -->
    <Chart Name="DAChart" Title="Number Of Sales Orders" ChartType="Column" Query="OrderCount">
       <Category Format="" Column="Year" DisplayName="Sales Year" />
       <Series Format="" Column="Region" DisplayName="Region" />

        <!-- Expression. IF Global Sales Promotion is ON, THEN #of Orders is doubled.
             ELSE #of Orders is 6 times more in Regional Sales Drive END IF -->

        <Values>
           <Value Format="Rounded0"
                       Expression="if($globalpromotion==1,OrderCount * 2,
                                      if(Region==$regionalsalesdrive, OrderCount * 6,OrderCount))"
                       ValueSequence="1" DisplayName="Number Of Orders" />
        </Values>
      </Chart>

    <!-- DAPanel Configuration -->
      <DAPanel Name="SalesDAPanel" Title="Sales Dynamic Analytic Panel">
            <VariableName Name="$globalpromotion" Sequence="1" />
            <VariableName Name="$regionalsalesdrive" Sequence="2" />
      </DAPanel>

   </Dashboard>

   <DashboardLayout>
        <Landscape>
            <!-- Dashboard display in Landscape Orientation -->
            <Layout Name=" DAChart " Top="0" Left="0" Width="100" Height="70" />
            <Layout Name="SalesDAPanel" Top="71" Left="0" Width="100" Height="30" />
        </Landscape>
        <Portrait>
            <!-- Dashboard display in Portrait Orientation -->
            <Layout Name=" DAChart " Top="0" Left="0" Width="100" Height="70" />
            <Layout Name="SalesDAPanel" Top="71" Left="0" Width="100" Height="30" />
        </Portrait>
    </DashboardLayout>

</MobileAnalytic>
```

5. Click **File** > **Save.**

Adding Scripts in ChartQuery and ViewController Classes

Modifying ChartQuery Class

No script is needed in `ChartQuery` class; we are reusing the OrderCount query.

Adding Script to the MainViewController Class

1. In the Project Navigator, click the `MainViewController.m` file.
2. Add this code to the `ViewDidLoad` method with the existing script in the method:

```
- (void)viewDidLoad
{
    ...

    //Dynamic Analytic Chart
    dict = [[NSMutableDictionary alloc] init];
    [dict setObject:@"MAKit Dynamic Analytic Chart" forKey:@"Title"];
    [dict setObject:@"DAPanelMetadata" forKey:@"MetaData"];
    [self.viewControllers addObject:dict];
    [dict release];

}
```

3. Click **File** > **Save**.

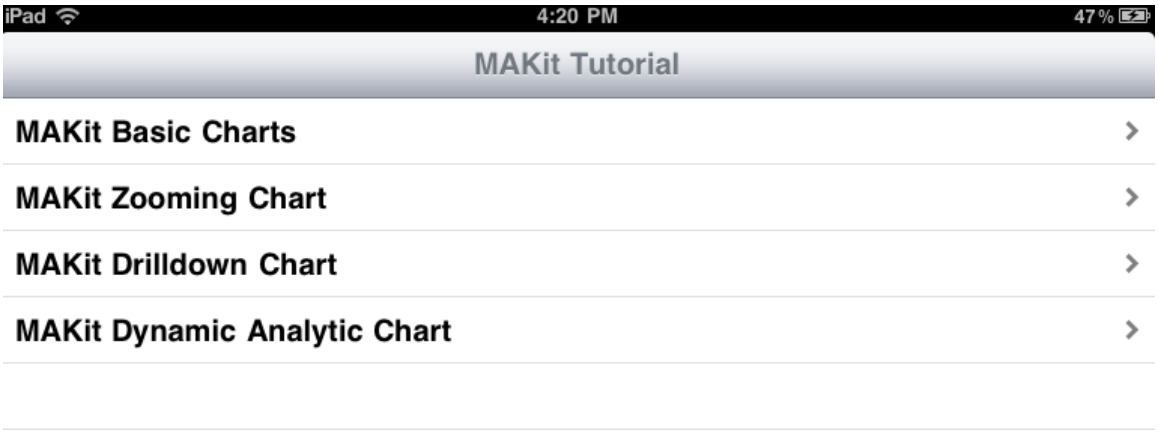
Deploying the Application

Deploy the MAKitSample application to the iPad Simulator for testing.

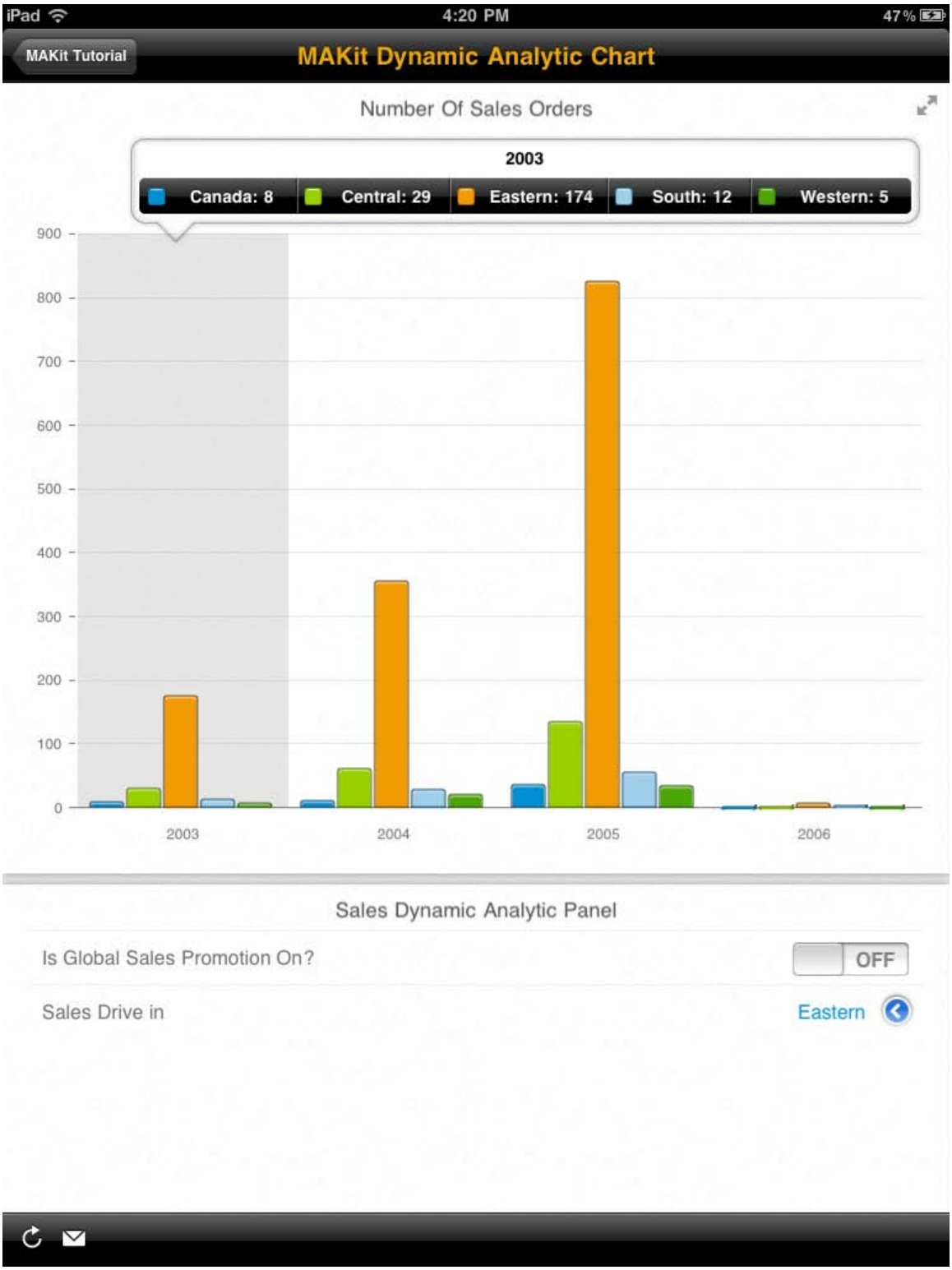
1. In the Xcode MAKitSample project, verify that the **Scheme** is set to **MAKitSample > iPad 4.3 Simulator**, and select **Product** > **Clean.**
2. Select **Product** > **Build.**

   If you correctly followed all the steps in the tutorial, you see a **`Build Succeeded`** message.
3. Select **Product** > **Run**.

   The project is rebuilt and iPad Simulator starts and load the application. The initial screen now contains an "MAKit Dynamic Analytic Chart" row.

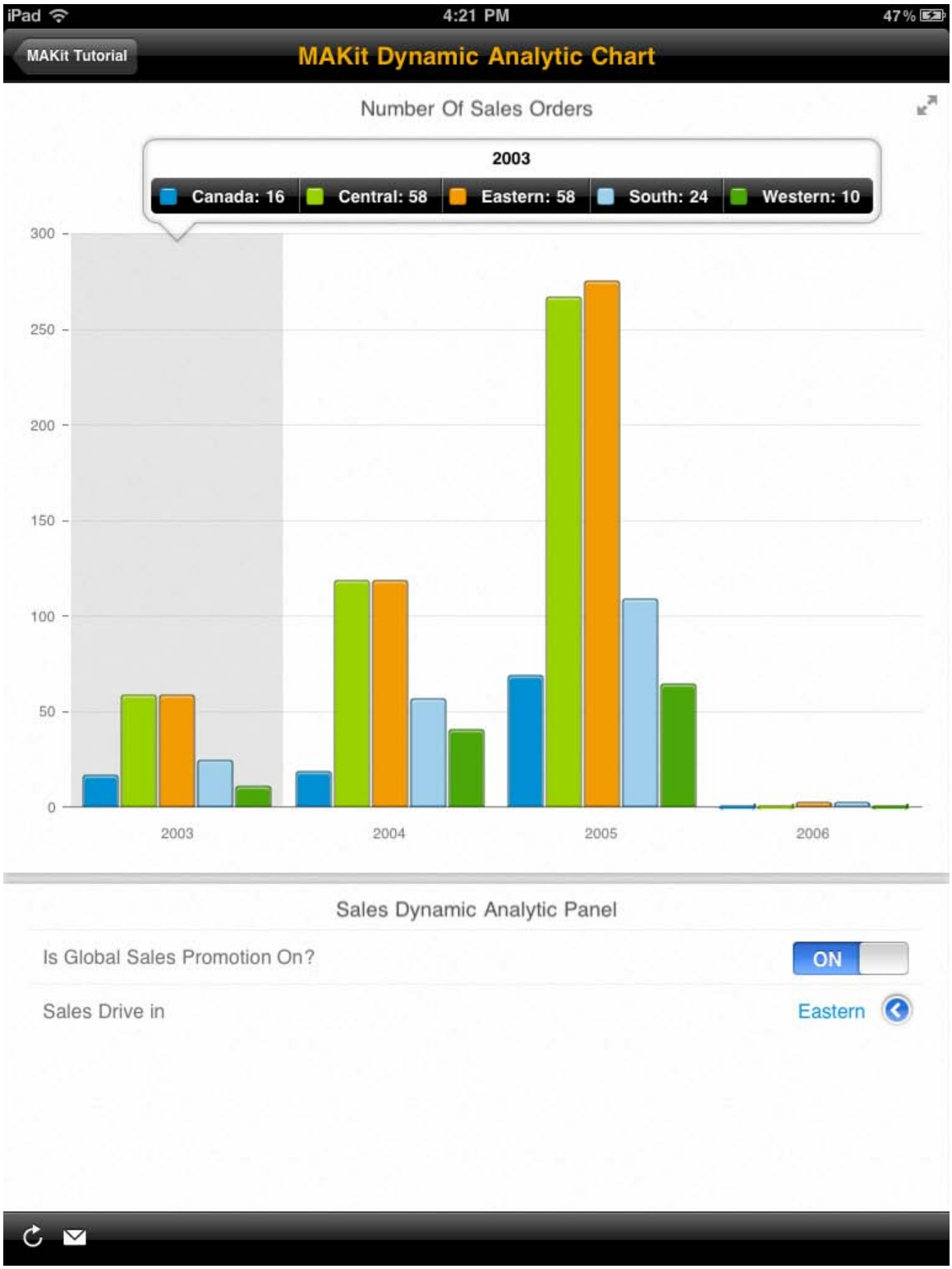


4. Click **MAKit Dynamic Analytic Chart**.



5. Set the Global Sales Promotion switch to ON.

   The OrderCount across all regions in all categories is doubled, and the chart redrawn.



Lesson 6: Adding What-If Analysis to Charts

What-if analysis is similar to dynamic analytics (DA). Both allow the user to change data behind the chart and immediately see the chart updated; both map the data to UI controls on the screen. However,

dynamic analytics changes the variables defined in expressions, while what-if analysis changes data retrieved from the datasource.

This lesson describes:

- Configuring what-if columns in XML metadata
- How what-if columns are mapped to UI controls
- Runtime behavior of what-if analysis

Adding Methods to ChartsDataSource Class

To illustrate what-if analysis, create a new datasource method in `ChartsDatasource` class and use some of the datasource columns to configure what-if columns in the metadata.

1. Open the `MAKitSample.xcodeproj` in the Xcode IDE if it is not already open.
2. In the Project Navigator, click the `ChartsDatasource.h` file and add these method definitions:

```
#import <Foundation/Foundation.h>
#import "sqlite3.h"

@interface ChartsDatasource : NSObject {
    sqlite3 *dbInstance;
}
...

//What-If Chart Datasource methods
- (NSArray *) ProductSales;
```

3. Click **File** > **Save**.
4. Click the `ChartDatasource.m` file and add the method implementation code:

```
#import "ChartsDatasource.h"

@implementation ChartsDatasource

...

//WhatIf Chart Datasource method
- (NSArray *) ProductSales {

     const char * SELECT = "SELECT  Region,ProductType||'-'||ProductName, UnitPrice,sum(sd.Quantity) "
                           " FROM SalesOrder_Header s, SalesOrder_Details sd, Product p "
                           " WHERE s.OrderID = sd.OrderID AND sd.ProductID = p.ProductID  AND "
                           " p.ProductID < 500  GROUP BY Region,ProductType,UnitPrice";



    NSMutableArray *data = [[NSMutableArray alloc] init];
    sqlite3_stmt *selectStmt = nil;
    if (sqlite3_prepare_v2(dbInstance, SELECT, -1, &selectStmt, NULL)==SQLITE_OK) {

        NSString *region=@"";
        NSString *product = @"";
        float unitprice=0.0;
        double qty =0;


       while (sqlite3_step(selectStmt)==SQLITE_ROW) {
            region = [NSString stringWithUTF8String:(char *) sqlite3_column_text(selectStmt, 0)];
            product = [NSString stringWithUTF8String:(char *) sqlite3_column_text(selectStmt, 1)];
           unitprice = sqlite3_column_double(selectStmt, 2);
           qty = sqlite3_column_double(selectStmt, 3);

            NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
            [dict setObject:region forKey:@"Region"];
            [dict setObject:product forKey:@"Product"];
            [dict setObject:[NSNumber numberWithFloat:unitprice] forKey:@"UnitPrice"];
            [dict setObject:[NSNumber numberWithDouble:qty] forKey:@"Quantity"];

            [data addObject:dict];
            [dict release];
         }
     }
    else {
        NSLog(@"Failed at Prepare Statement in ProductSales.Reason %s",sqlite3_errmsg(dbInstance));
    }
    return data;

}


@end
```

5. Click **File** > **Save**.

Creating XML Metadata File Configured with What-If Columns

Create a new XML metadata file to configure charts with what-if analysis. In the chart configuration metadata, add datasource columns in the *<WhatIfColumns> … </WhatIfColumns>* tag.

1. In the MAKitSample project, click **File** > **New** > **New File.**
2. Select **iOS** > **Other** > **Empty** and click **Next.**
3. Enter `WhatIfChartMetadata.xml` in the **Save As** field and click **Save.**

   This creates the `WhatIfChartMetadata.xml` file in the Project Navigator.
4. Click `WhatIfChartMetadata.xml` and add this content to the file:

```
<?xml version="1.0" encoding="utf-8"?>
<MobileAnalytic MAKitVersion="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MAKitMetaFile.xsd">
  <Dashboard>

    <!-- WhatIf Chart Configuration -->
   <Chart Name="WhatIfChart" Title="Product Sales Across Regions" ChartType="Column"
                                 Query="ProductSales">
      <Category Format="" Column="Region" DisplayName="Region" />
      <Series Format="" Column="Product" DisplayName="Product" />
      <Values>
         <Value Format="Currency" Expression="UnitPrice * Int(Quantity)" ValueSequence="1"
                                    DisplayName="Sales Amount" />
      </Values>

      <!-- What If Columns -->
      <WhatIfColumns>
          <Column Name="Quantity" DisplayName="Sales Quantity"  VariableType="integer"
                        Value="200" Min="1" Max="3000" />
          <Column Name="UnitPrice" DisplayName="Unit Price" VariableType="float"
                        Value="0.0" Min="0.0" Max="50.0" Step="0.5" Decimals="2" />
      </WhatIfColumns>

     </Chart>
   </Dashboard>

   <DashboardLayout>
        <Landscape>
            <!-- Dashboard display in Landscape Orientation -->
            <Layout Name=" WhatIfChart" Top="0" Left="0" Width="100" Height="80" />
        </Landscape>
        <Portrait>
            <!-- Dashboard display in Portrait Orientation -->
            <Layout Name=" WhatIfChart" Top="0" Left="0" Width="100" Height="85" />
        </Portrait>
    </DashboardLayout>

</MobileAnalytic>
```

5. Click **File** > **Save.**

Adding Scripts in ChartQuery and ViewController Classes

Modifying the ChartQuery Class

Add code to the `ChartQuery` class to execute the query method.

1. In the Project Navigator, click the `ChartsQuery.m` file.
2. Append this code to the `executeQuery` method with the existing script in the method:

```
#import "ChartQuery.h"
#import "ChartsDatasource.h"
#import "ChartDataTable.h"

@implementation ChartQuery

#pragma mark -
#pragma mark MAQueryDelegate method

-(id<MAQueryDataTable>)executeQuery:(NSString*)queryName withArgument:
(NSDictionary*)args {

    ChartDataTable *dataTable = [[ChartDataTable alloc] init];


    ...

    else if ([queryName isEqualToString:@"ProductSales"]) {
        ChartsDatasource *cds = [[ChartsDatasource  alloc] init];
        dataTable.tabularData = [cds ProductSales];
        [cds release];
    }

     return dataTable;
}

@end
```

3. Click **File** > **Save**.

Adding Script to the MainViewController Class

1. In the Project Navigator, click the `MainViewController.m` file.
2. Add this code to the `ViewDidLoad` method with the existing script in the method:

```
- (void)viewDidLoad
{
    ...

    //What-If Analysis Chart
    dict = [[NSMutableDictionary alloc] init];
    [dict setObject:@"MAKit What-If Analysis Chart" forKey:@"Title"];
    [dict setObject:@"WhatIfChartMetadata" forKey:@"MetaData"];
    [self.viewControllers addObject:dict];
    [dict release];

}
```

3. Click **File** > **Save**.

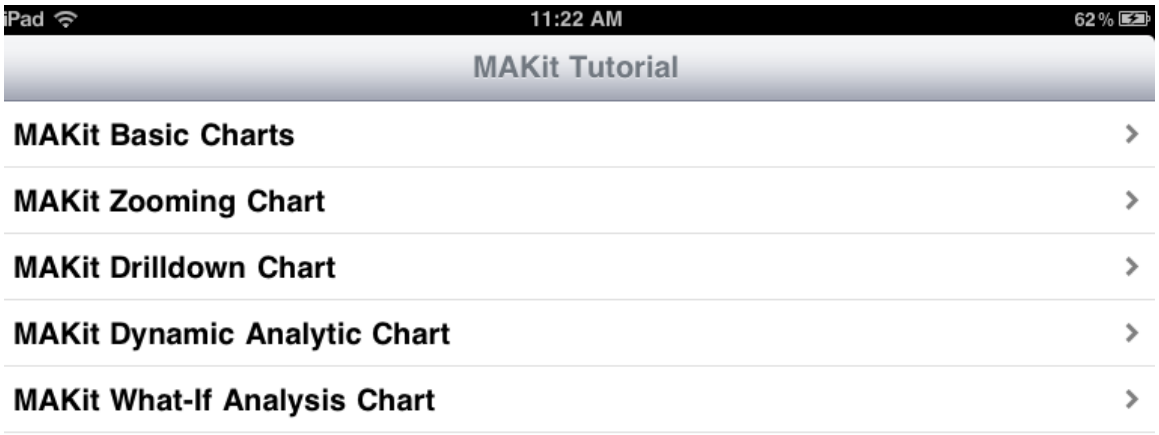
Deploying the Application

Deploy the MAKitSample application to the iPad Simulator for testing.

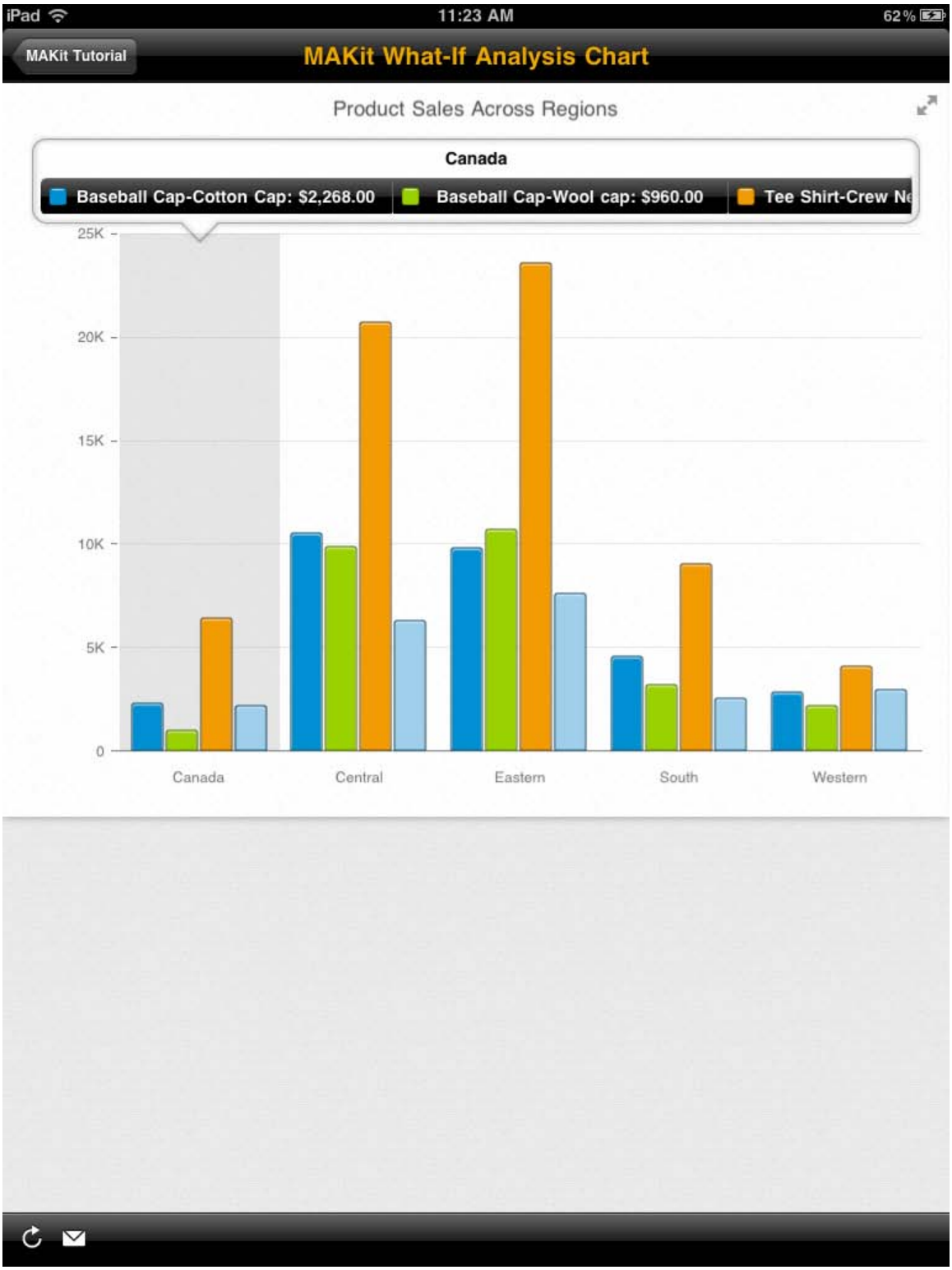
1. In the Xcode MAKitSample project, verify that the **Scheme** is set to **MAKitSample > iPad 4.3 Simulator**, and select **Product** > **Clean.**
2. Select **Product** > **Build.**

   If you correctly followed all the steps in the tutorial, you see a **`Build Succeeded`** message.
3. Select **Product** > **Run**.

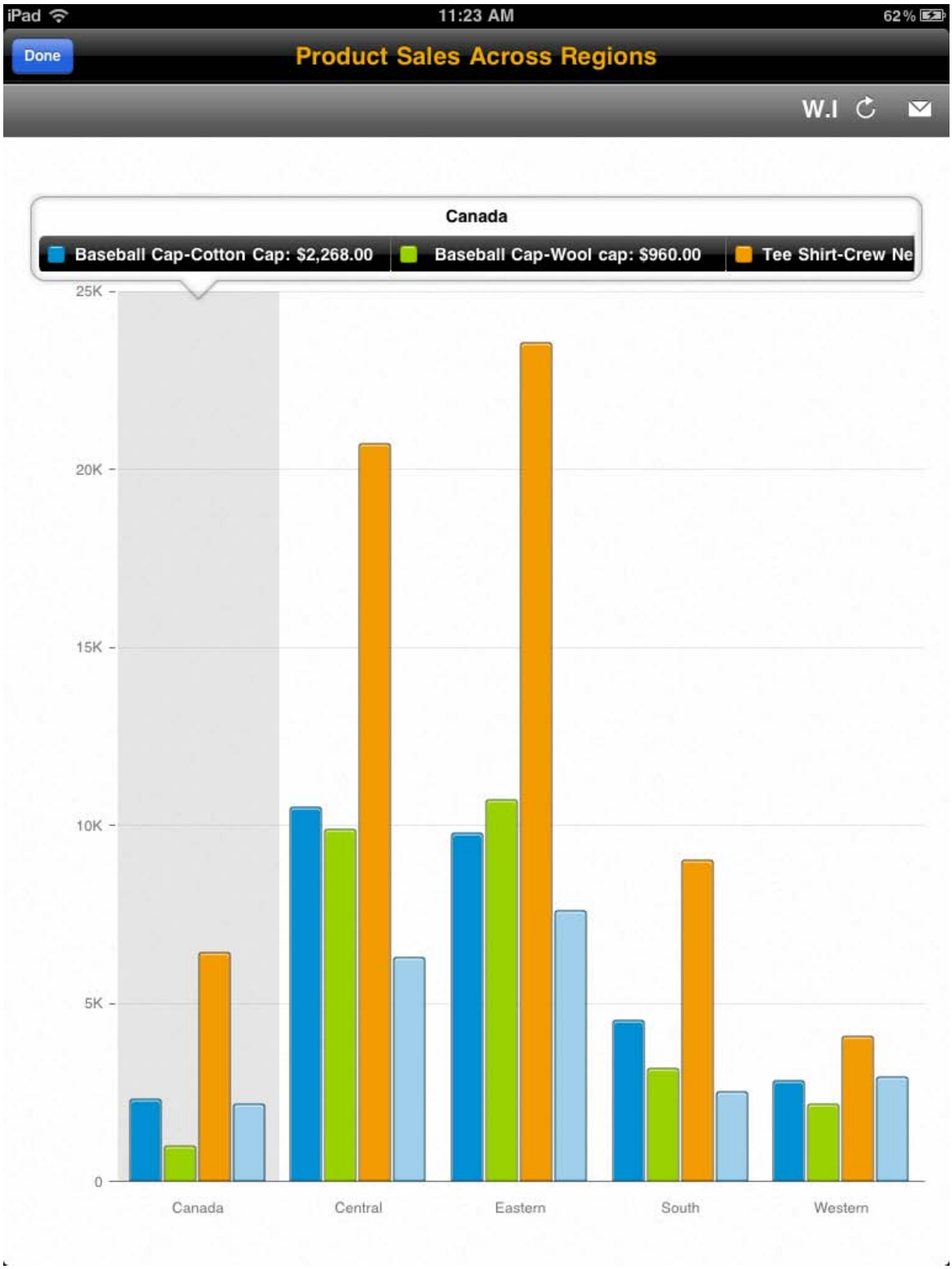
The project is rebuilt and iPad Simulator starts and load the application. The initial screen now contains an “MAKit What-If Analysis Chart” row.



4. Click **MAKit What-If Analysis Chart**.



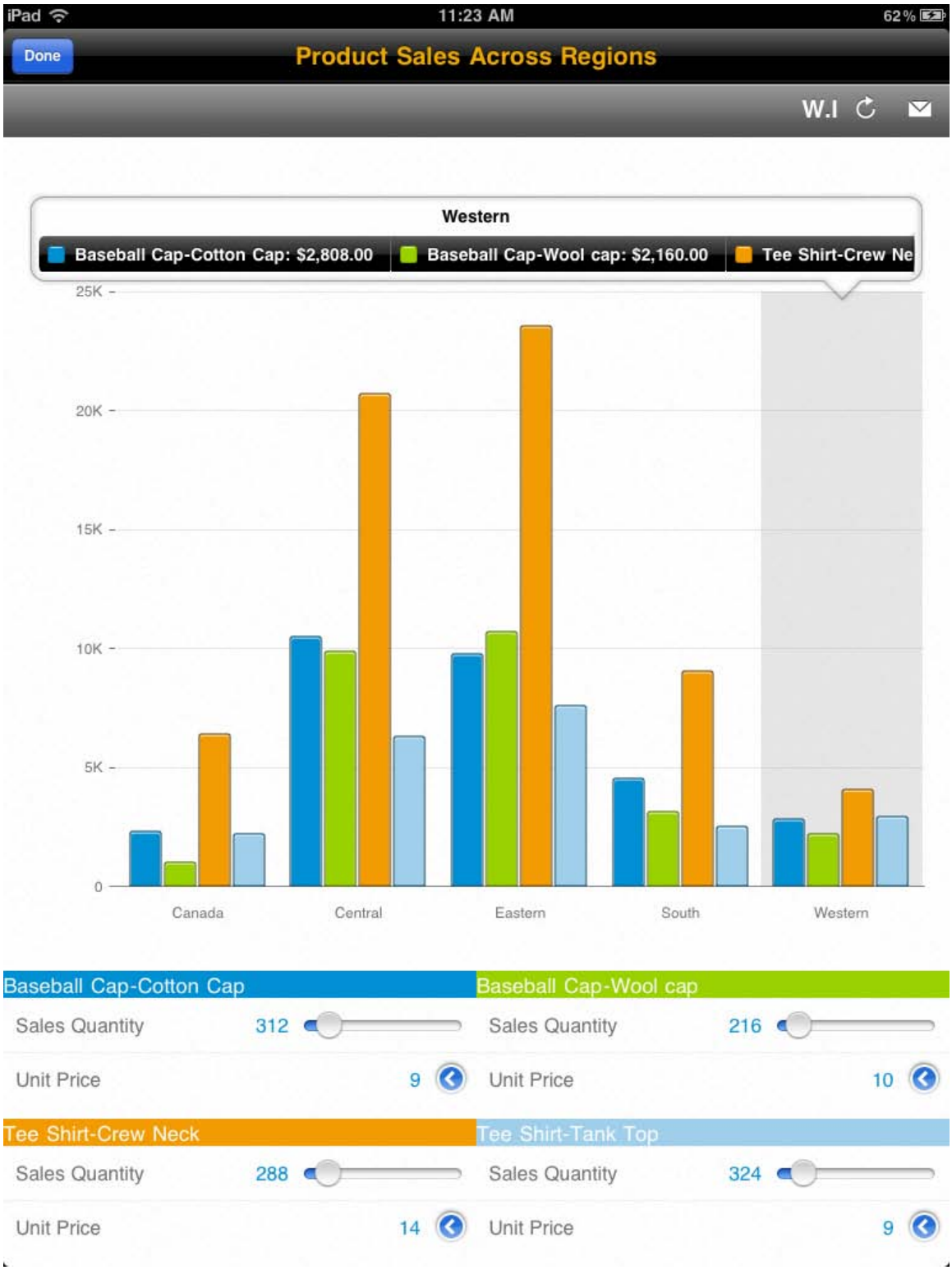
5. Click the **arrow image** in the upper-right corner to load the chart in full screen.



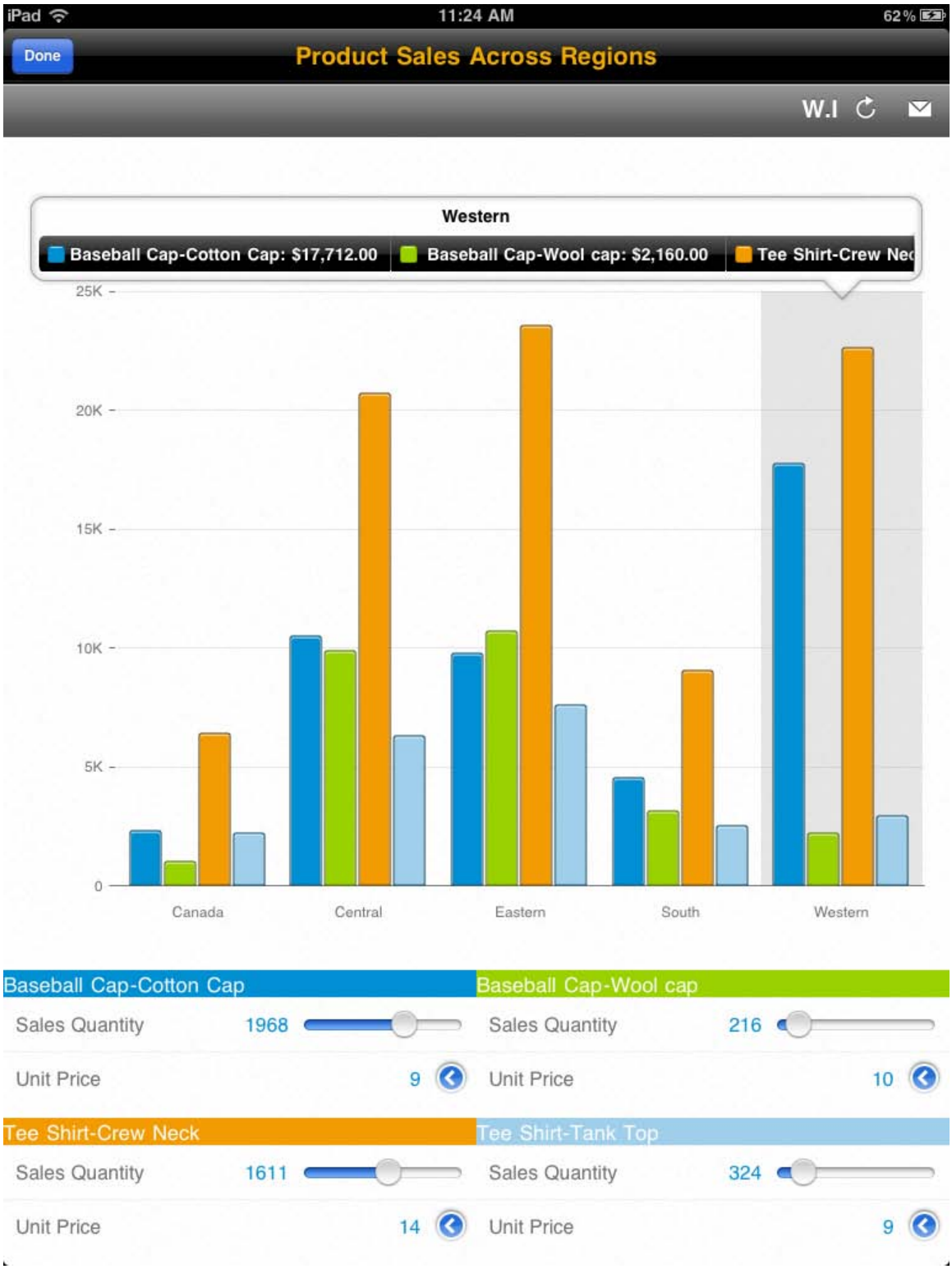
6. Tap the category "**Western.**"

   The Western region is selected and the value bubble is updated with values for the Western region.

7. While in full-view mode, tap the **W.I** toolbar button to see the what-if controls for the columns specified in the metadata.



8. Modify the values in the what-if controls to redraw the chart based on those values.



Lesson 7: MAKit Chart E-Mail Sharing

E-mail sharing is a built-in feature of all MAKit charts that are configured through metadata; no coding is required.

With MAKit e-mail sharing, a chart in its current state (as it appears in a device) is captured as an image and can be further annotated and sent to the business users or peers whose contacts are stored in the device.

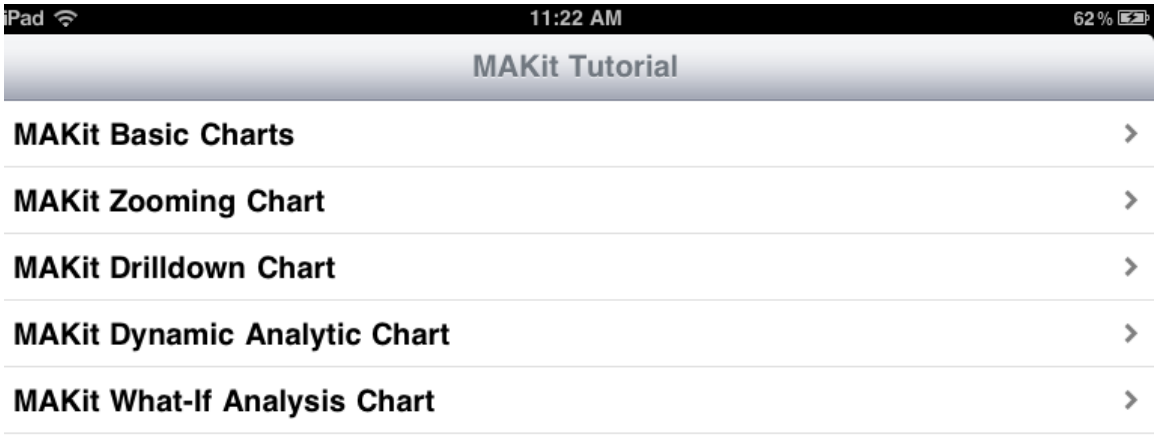
This lesson describes:

- Invoking the e-mail sharing feature
- Annotating a chart and sending it as an e-mail attachment

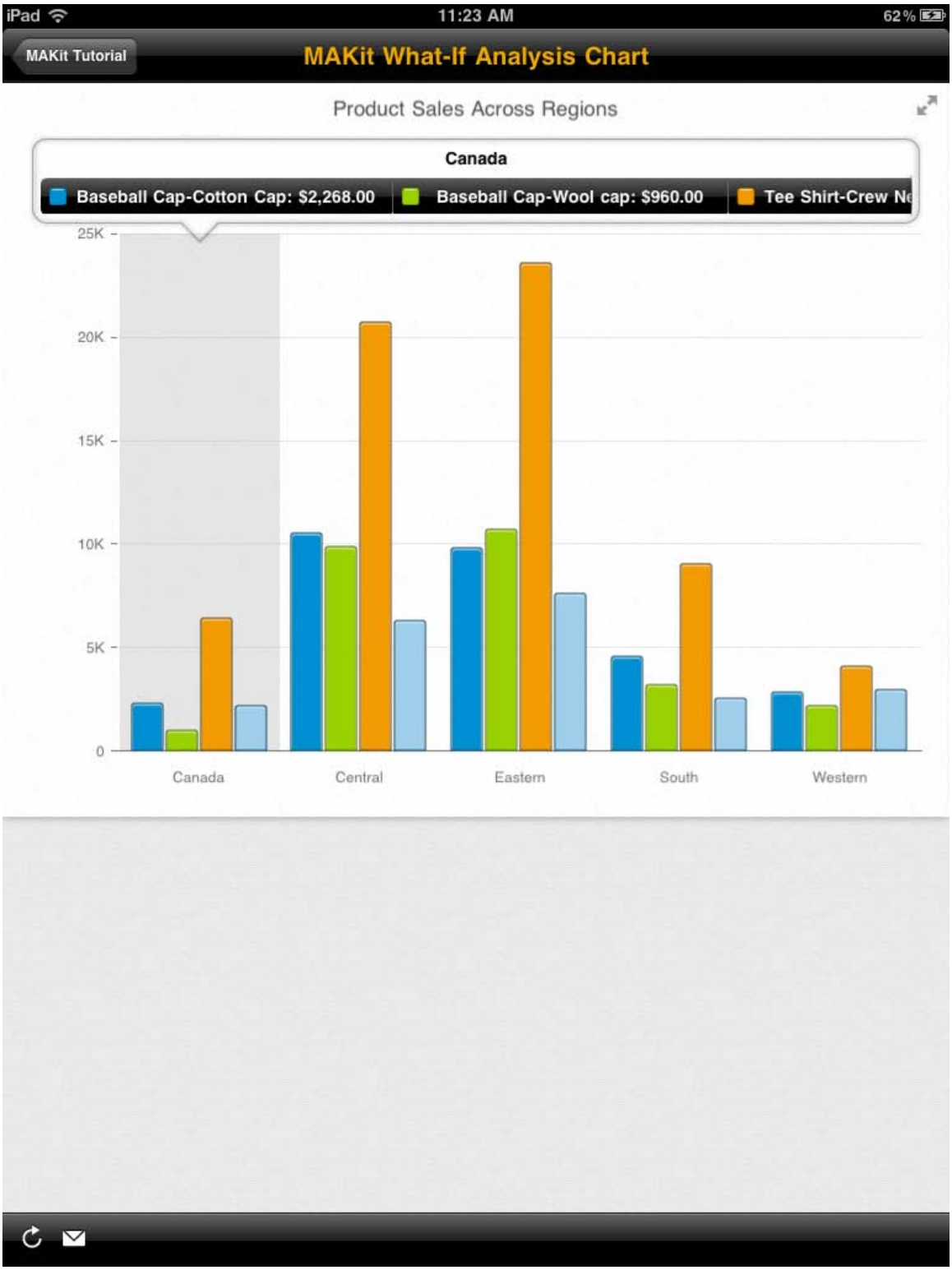
Sending a Chart Through E-mail

1. In the Xcode MAKitSample project, verify that the **Scheme** is set to **MAKitSample > iPad 4.3 Simulator.**
2. Select **Product** > **Run**.

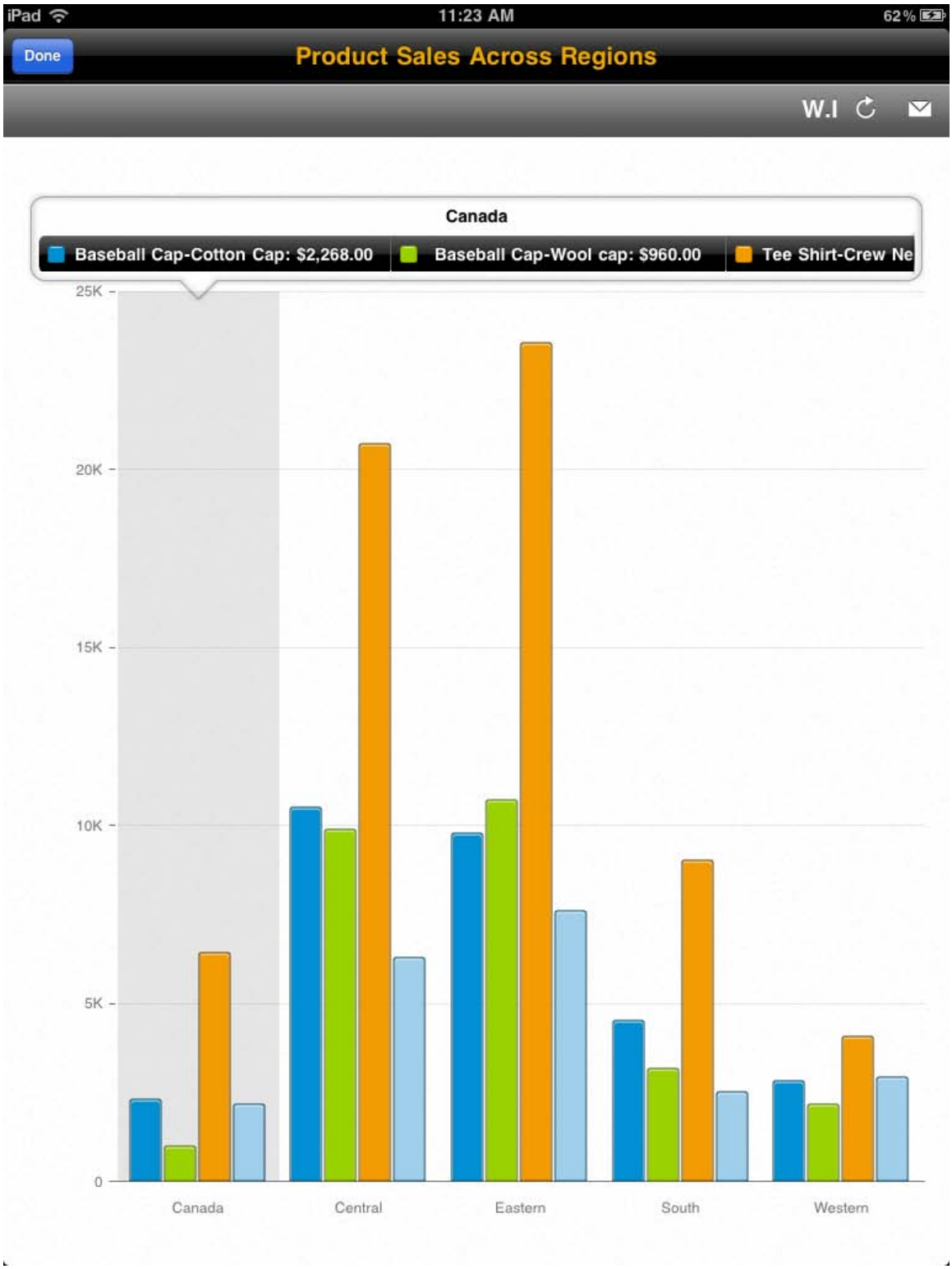
   The project is rebuilt and iPad Simulator starts and load the application.



3. Click **MAKit What-If Analysis Chart**.

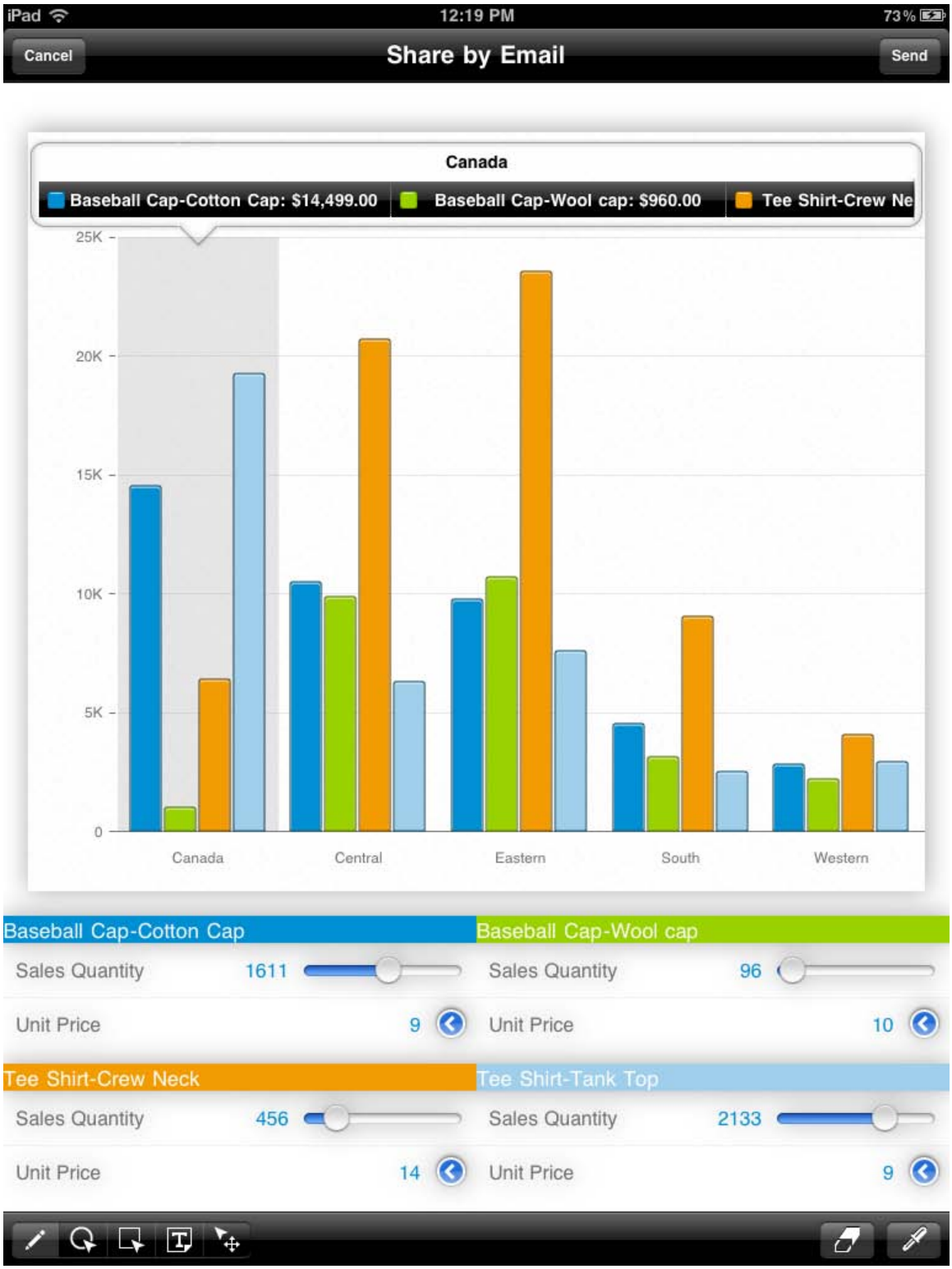


4. Click the **arrow image** in the upper-right corner to load the chart in full screen.

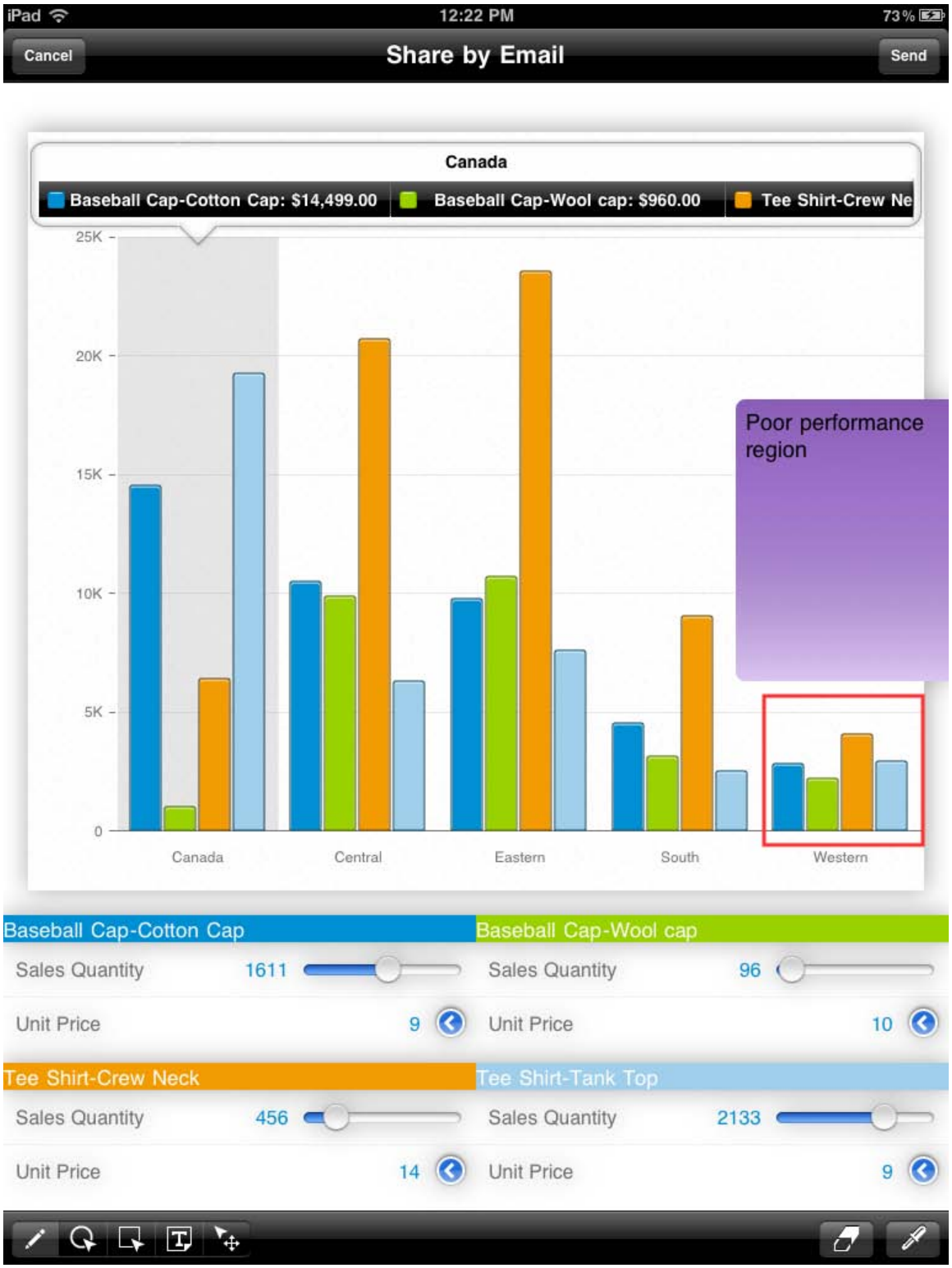


5. Tap the **W.I** toolbar button, which shows the what-if controls for the columns configured in the metadata.

6. Tap the mail image toolbar button. This captures the current screen image and loads it; the annotation toolbar appears below the image.



7. Use the annotation toolbar items to annotate the image as shown below:



8. Tap **Send** to load the annotated image in a new e-mail message as shown:

iPad
12:22 PM
73%
Cancel
Product Sales Across Regions
Send
To:
Cc/Bcc:
Subject: Product Sales Across Regions
Chart Snapshot
Canada
Baseball Cap-Cotton Cap: $14,499.00
Baseball Cap-Wool cap: $960.00
Tee Shirt-Crew Ne
25K
20K
15K
10K
5K
0
Canada
Central
Eastern
South
Western
Poor performance region
Baseball Cap-Cotton Cap
Sales Quantity 1611
Unit Price 9
Baseball Cap-Wool cap
Sales Quantity 96
Unit Price 10