Getting started with TeeChart and MonoTouch

By Pep Jorge @josepluisjorge | Steema Software | February 2013

Introduction

A few months ago Steema Software released a new TeeChart product called "TeeChart NET for iOS", and it is this that we will discuss in this article.

If you are a software developer using CSharp programming language and are thinking of porting your applications to the iOS (iPhone and iPad) then for sure you've heard of MonoTouch. MonoTouch allows you to create all kinds of applications from the MonoDevelop development environment (installed on a machine with Apple OSX) using c #. If you don't already have Monotouch you can download a trial version from www.xamarin.com.

TeeChart NET for iOS is a component library that allows programmers to add all kind of charts to their applications easily. The component library is available with source code (CSharp native code) or as a binary version. Both are adapted for use with MonoTouch.

The version of the product available on the website includes several coded examples to guide use, but this article aims to show how to populate the Charts, in a few steps, getting the data from a SQLite database, starting with a simple TeeChart example.

Using the TChart component with MonoTouch in a UIView

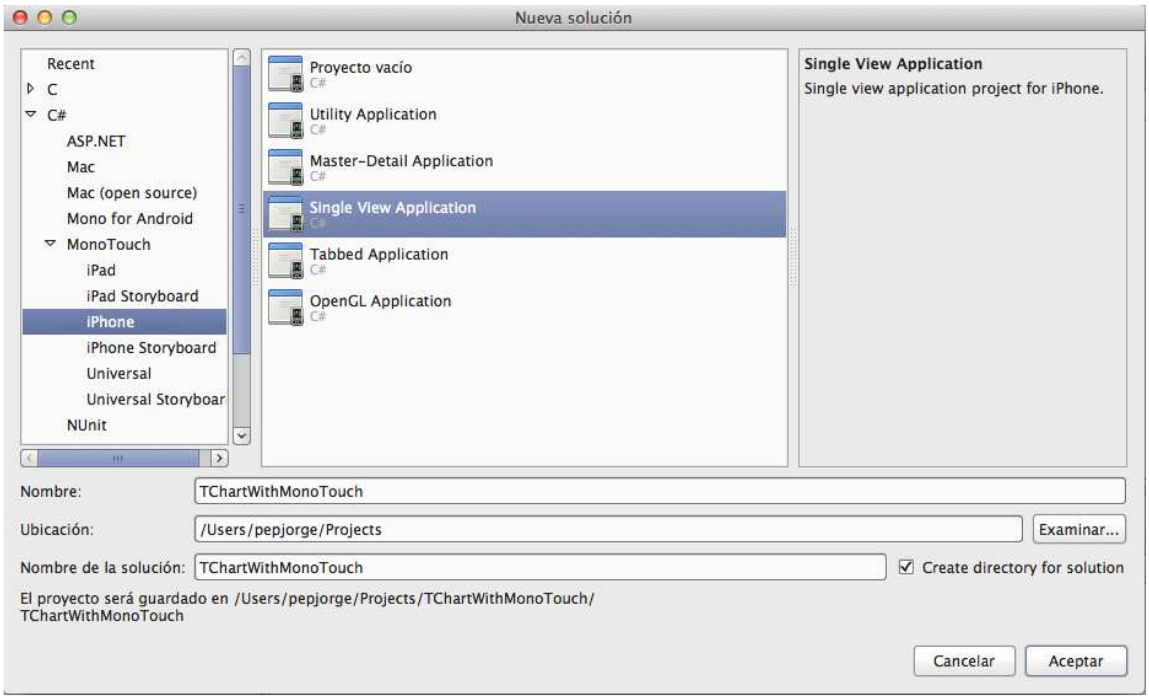
TChart is a data visualization control that creates a graphical representation of data. It can handle high amounts of data, from a database if required. It supports many different series (display) types; standard types like Line, Area, Bar, Pie, Gantt; professional 2D or 3D series types, like Surface, TriSurface, Contour and indicators like Circular Gauges, Horizontal Indicators, and more..

We're going to start this tutorial showing the required steps in order to create a simple iPhone application which will contain a basic Pie chart in an UIView. Then we'll continue with another example setting the chart series to source data from a database, specifically from a SQLite Table, loading the data into the same Series.

Hopefully I should be able to show you how easy it is to include Charts in our applications.

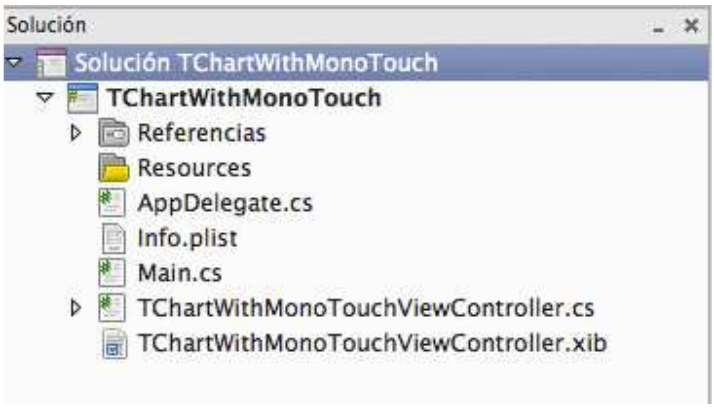
Step 1 – Creating the Project :2

Open the MonoDevelop IDE, and create a "Single View Application from File -> New solution -> C # -> MonoTouch -> iPhone -> Single View Application. We will give it a descriptive name (ie "TChartWithMonoTouch") and click OK.



(Figure 1 – Creating new Project)

Once this is done we'll see the Solution as in Figure 2. A ViewController with its XIB file will be created automatically to represent the view. The XIB file will allow us to customize the UIView through the XCode, by double clicking over the file, XCode will be opened automatically and the designer will appear, but we're going to skip this step as it's not necessary for our purposes for the moment :

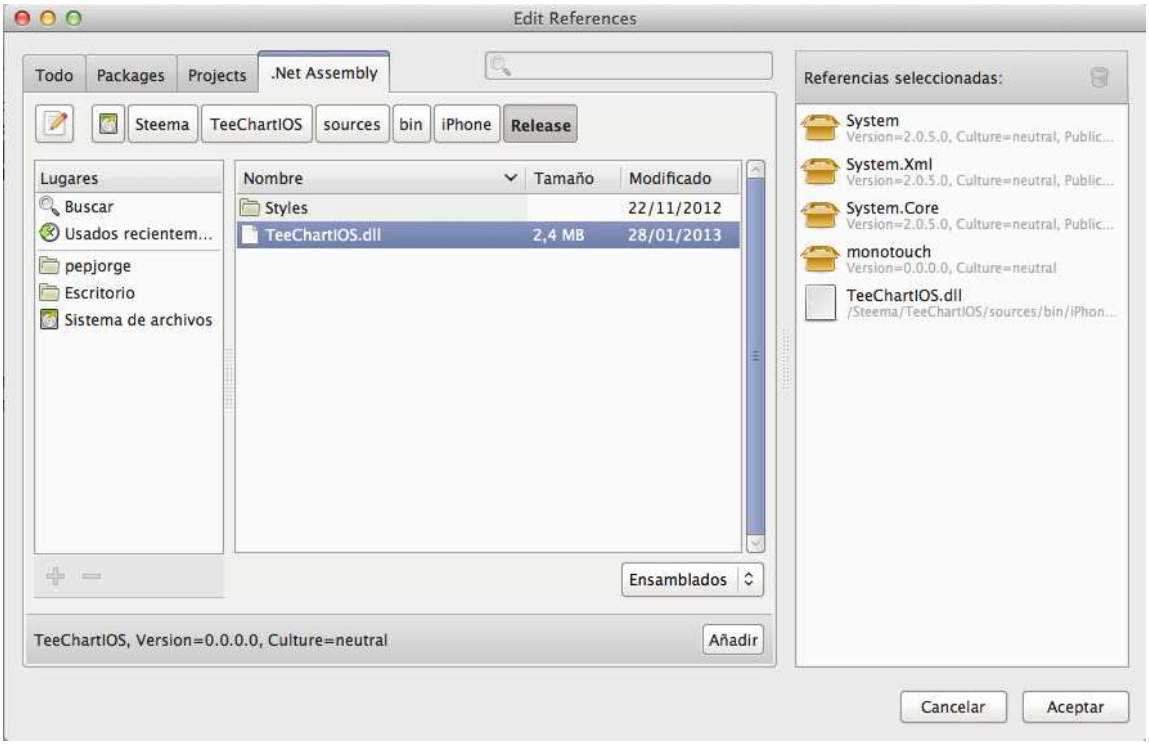


(Figure 2 –Project Solution)

Step 2 – Adding and configuring the TChart control :

In order to create the Chart control in our class controller the first thing to do is to add the "NET

TeeChart for iOS" library reference. Just do right click on "References" of the solution and select "Edit references", go to the Net Assembly tab and add the library TeeChartIOS.dll which is included into the installer. Then click OK.



(Figure 3 –Referencing to TeeChartIOS.dll)

Now that the TeeChart library is already referenced, open the TChartWithMonoTouchViewController.cs controller file and add the "using" line of code, so we can make use of all its objects, methods and properties.

```
using Steema.teeChart;
```

In the class define a new TChart control :

```
TChart chart1 = new TChart();
```

We've also to define the a dimension for the object, we'll do that into the ViewDidLoad() method, and we'll start defining a new Series type, Pie style for example:

```
// Specifing a Chart dimension
System.Drawing.RectangleF rect = new System.Drawing.RectangleF(0,0,320,460);
// Creating the Series type
Steema.TeeChart.Styles.Pie pie = new Steema TeeChart.Styles.Pie();
// Adding  Series to the Chart
chart1.Series.Add(pie);

// Loading data to the Pie series, we can use Random data just to test
pie.FillSampleValues(4);
```

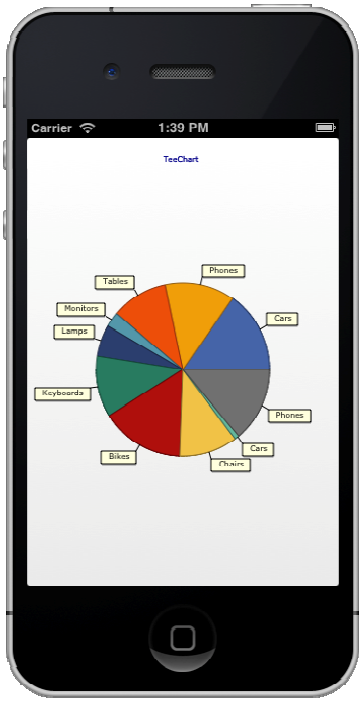
```
// or add  specific values for the Series
pie.Add(10);
pie.Add(20);
pie.Add(30);
pie.Add(40);

// Now will be time to customize our Chart and Series, we could change a few
specific characteristics to change its aspect

// Setting Chart to 2D and hiding legend
chart1.Aspect.View3D=false;
chart1.Legend.Visible=false;

// Setting Pie series as Circular, and marks visible
pie.Circled=true;
pie.Marks.Visible=true;
```

As you can see, we can easily configure the chart in a few minutes.
Now it is time to embed the Chart into the main View of the application. The TChart Control inherits from the UIView so it can be added as subview in any other views, while other controls can be added to the chart.

```
View.AddSubView(chart);
```

Step 3 – Running the Application on Simulator and Device

Running the application we should get the following result :



(Figure 4 –Running the Project)

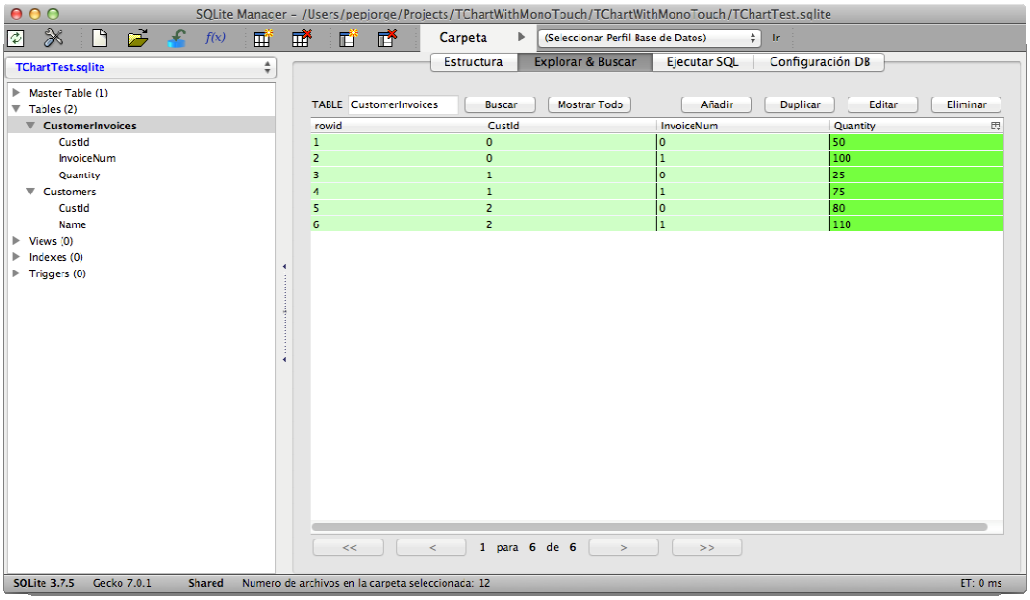
Adding DataBase functionality to the Project:

We now know how to create a chart, customize and display it on any view of our application. Let's now give more functionality to the chart, for example we could create an application which gets the data from a table of an SQLite database, the represented data from different customers will be displayed in a list. Then, when clicking over a specific customer, the Chart series will refresh automatically with new data, reflecting that data from the database.

We could try to use the same project we did before by changing a few lines of code.

Step 1 –Creating the SQLite DataBase and Tables :

The first thing to do is create our SQLite database and the two tables we need for our application. There are several tools to manage SQLite databases, I'm going to use the addon that can be installed in Firefox (through extensions):



(Figure 5 –Working with SQLite Manager)

We create our database that we will call "TChartTest", then the first table, which we will call "Customers", which will include two Fields:

CustId of type integer
Name of type text(30)

The second table will call it “CustomerInvoices”, and will contain the following Fields :

CustId of type integer
InvoiceNum of type integer
Quantity of type double

Once this is done, we’ll add some records with example data. By double-clicking over each table (through the manager), it will allow us to edit, add or remove the records. We’re going to add the following example data to the “Customers” table:

CustId Name

0 Nicole
1 John
2 Kate
3 James

And for the “CustomerInvoices” table:

CustId InvoiceNum Quantity

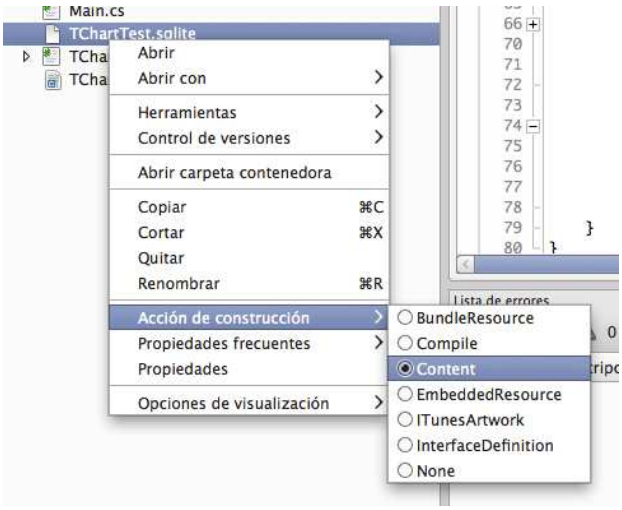
0	*0*	*50*
0	*1*	*100*
1	*0*	*25*
1	*1*	*75*
2	*0*	*80*
2	*1*	*110*

Step 2 –Using the Table data in the Project:

First of all, what we need to know is that to be able to work with SQLite dataBases in our project, we've to add a new Reference to the project, the Mono.Data.SQLite reference. You can add this by right-clicking over the References -> Edit References, and select it from the list of available references.

Now that all the data has been introduced, we save the dataBase to a file, which we've to add as a part of our application by right-clicking over the Project solution-> Add Files.

We also have to right-click over the DataBase file (already added on our Project) and change the action as "Content" via the Action-> Content option, this will make it part of the Project at deployment time.



(Figure 6 –Setting SQLite DataBase as content of project)

Once the dataBase is integrated in the Project, we can define a new small class (in the same controller file) which will define the structure of the data we'll use to store the information of the ítems, that will be displayed at the UITableView:

```
// Class to define the Items
public class ItemInfo
{
  public int CustID { get; set; }
  public string Name { get; set; }

  public ItemInfo(int custId,string name)
```

```
  {
    CustID = custId; Name = name;
  }
  public override string ToString ()
  {
    return string.Format("[ItemInfo: Value={0}, Text={1}]", CustID, Name);
  }
}
```

Step 3 –Adding the Chart and TableView objects to the View:

Let's prepare the principle view so that it shows a UIListView on half of the screen and the Chart for the rest of it. We could use XCode to add the component UIListView to our View but as it's quite a simple step I prefer to do it by code, it's quicker.

To start with, we modify the following initial project definitions inside the TChartWithMonoTouchViewController class.

```
// Creating the Chart
public TChart chart1 = new TChart();

// Creating the series type
public Steema.TeeChart.Styles.Bar bar;

// Creating the Table view list
public UITableView table;
public List<ItemInfo> tableItems;
```

To populate data in a UITableView and then update our chart by clicking on a Table Cell we need to create a DataSource and Delegate classes for our TableView, we first create the controls and assign its DataSource and Delegate within the ViewDidLoad method of our controller. We'll create and define the TableSource and TableDelegate classes themselves later in our code.

At the same time we'll specify various properties of the Chart and Series that we'll use to display the data from the DataBase (in this example Bar Series).

```
public override void ViewDidLoad ()
{
  base.ViewDidLoad ();

  // Adding the UITableView and items to the View
  table = new UITableView(new RectangleF(0,0,320,190));

  tableItems = new List<ItemInfo>();
  table.DataSource = new TableSource(this);
  table.Delegate = new TableDelegate(this);
```

```
  View.AddSubview(table);
}
```

We already have the UITableView in the main View, now let's add the chart at the bottom of the view. Add the following piece of code within the method again into the ViewDidLoad ():

```
// Setting the Chart dimension
System.Drawing.RectangleF rect = new System.Drawing.RectangleF(0,190,320,270);
chart1.Frame = rect;

// Setting automatic Zoom and Scroll to manual
chart1.Aspect.ZoomScrollStyle=Steema.TeeChart.Drawing.Aspect.ZoomScrollStyles.Ma
nual;

// Adding series to the chart
bar = new Steema.TeeChart.Styles.Bar();

// Some settings for the bar Series type
bar.BarStyle = Steema.TeeChart.Styles.BarStyles.Arrow;
bar.BarWidthPercent = 200;
bar.Marks.Style = Steema.TeeChart.Styles.MarksStyles.Value;
bar.ColorEach = true;

// Adding Bar series to the Chart
chart1.Series.Add(bar);

// Some more settings for the Chart
chart1.Aspect.View3D = false;
chart1.Legend.Visible = false;
chart1.Axes.Bottom.Title.Text = "Customer Invoices";
chart1.Axes.Left.AxisPen.Width = 1;
chart1.Axes.Left.Increment = 40;
chart1.Axes.Bottom.AxisPen.Width = 1;
chart1.Header.Text = "TeeChart NET for iOS";
chart1.Header.Font.Color = UIColor.Black.CGColor;
chart1.Panel.MarginTop = 0;
chart1.Walls.Back.Visible = false;
```

And finally add the Chart to the Main View as a subview:

```
View.AddSubview(chart1);
```

Step 4 –Creating the DataSource and Delegates for the Objects:

It's time to write our DataSource class for the TableView which we have assigned to our code above. This will load the data from our table "Customers", we will use a simple code to connect to our database, will execute our "select" and will return records to be introduced on our predefined list:

```
// DataSource for the TableView
public class TableSource : UITableViewDataSource
```

```
{
  private TChartWithMonoTouchViewController _controller;
  string cellIdentifier = "TableCell";

  public TableSource (TChartWithMonoTouchViewController controller)
  {
    _controller = controller;
    _controller.tableItems = LoadCustomerList();
  }

  public override int RowsInSection (UITableView tableview, int section)
  {
    return _controller.tableItems.Count;
  }

  private List<ItemInfo> LoadCustomerList()
  {
    // Necessary code to connecto with SQLita Database
    var conn = new SqliteConnection("Data Source=TChartTest.sqlite");
    var list = new List<ItemInfo>();

    // Here we'll do the select and load data into the list
    using (var cmd = conn.CreateCommand()
    {
      conn.Open();
      cmd.CommandText = "select CustId,Name from Customers";
      using(var reader = cmd.ExecuteReader())
      {
        while (reader.Read())
        {
          Int64 id = (Int64)reader["CustId"];
          list.Add(new ItemInfo((int)id , (string)reader["Name"]));
        }
      }
    }
    return list;
  }
}
```

..and now the Delegate class, that will be used when the user touches a Cell of the table, will refresh the Chart showing the invoices of the selected customer:

```
// Delegate for the TableView
public class TableDelegate : UITableViewDelegate
{
  private TChartWithMonoTouchViewController _controller;
  public TableDelegate(TChartWithMonoTouchViewController controller)
  {
    // We need to pass the controller to the constructor in order to use it
later
    _controller = controller;
  }
```

```
  public override void RowSelected(UITableView tableView, NSIndexPath indexPath)
  {
    LoadCustomerInvoices(_controller.tableItems[indexPath.Row].CustID);
    // tableView.DeselectRow (indexPath, true); // normal iOS behaviour is to
remove the blue highlight
  }

  // This method load data from specific customer to the Chart series
  private void LoadCustomerInvoices(int CustID)
  {
    var conn = new SqliteConnection("Data Source=TChartTest.sqlite");
    using (var cmd = conn.CreateCommand())
    {
      _controller.bar.Clear();
      conn.Open();
      cmd.CommandText ="select InvoiceNum,Quantity from CustomerInvoices where
CustId="+CustID;

      using(var reader = cmd.ExecuteReader())
      {
        while (reader.Read())
        {
          Double qt = (Double)reader["Quantity"];
         _controller.bar.Add(qt,(string)reader["InvoiceNum"].ToString());
        }
      }
    }
  }
}
```

Step 5 –Running and Testing the Project:

If we've followed all the steps correctly, we just have to run the application and look at the result, that should look like the following image :

(Figure 7 –Deploying the App)

Summary

You can download the solution here "TChartWithMonoTouch.zip" (The example does not include the TeeChartIOS.dll which is required to run into the Simulator or Device, non-customers can go to the Product downloads page in order to download the Eval version).

I hope this article has helped you get up and running with the TeeChart NET for iOS control with the use of MonoTouch.
If you have any questions or comments please feel free to contact us.