WRITING NATIVE MOBILE APPS IN A FUNCTIONAL LANGUAGE

SUCCINCTLY

BY VASSILI KAPLAN

SUCCINCTLY E-BOOK SERIES

Syncfusion

www.dbooks.org

Writing Native Mobile Apps in a Functional Language Succinctly

By
Vassili Kaplan

Foreword by Daniel Jebaraj



Copyright © 2018 by Syncfusion, Inc.
2501 Aerial Center Parkway
Suite 200
Morrisville, NC 27560
USA
All rights reserved.

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: Ed Freitas

Copy Editor: Courtney Wright

Acquisitions Coordinator: Tres Watkins, content development manager, Syncfusion, Inc.

Proofreader: Darren West, content producer, Syncfusion, Inc.

www.dbooks.org

Table of Contents

The Story Behind the *Succinctly* Series of Books **7**

About the Author **9**

Chapter 1 Introduction **10**

What will be covered in this book? 10

Advantages of using CSCS for mobile 10

“Hello, World!” in CSCS 11

Customizable scripting in C# 14

An example of implementing a CSCS function 15

Summary 16

Chapter 2 Project Structure and Design **17**

Structure of a CSCS mobile project 17

Where CSCS code executes 19

Summary 25

Chapter 3 Placing Widgets on the Screen **27**

Layout concepts in CSCS 27

Locations in CSCS 28

Adding widgets 30

Implementing device orientation changes 33

Example: Currency Convertor 35

Summary 43

Chapter 4 Creating Custom Widgets **44**

Adding a custom combo box 44

Autocomplete with a trie data structure 48

Summary 55

Chapter 5 Adding Syncfusion Controls....56

Adding Syncfusion to the CSCS project....56

Stepper or Numeric Up Down....60

QR barcode....63

Barcode 39....64

Picker....65

Digital Gauge....68

Circular Gauge....69

Calendar....70

Image Editor....71

Busy Indicator....72

Data Grid....74

Spline Chart....76

Doughnut and Semi-Doughnut Charts....77

Summary....78

Chapter 6 Adding PDF, Word, and Excel Functionality from Syncfusion to CSCS....79

Adding PDF file creation to CSCS....79

Adding Microsoft Word file creation to CSCS....83

Adding Microsoft Excel file creation to CSCS....87

Summary....93

Chapter 7 Text-To-Speech and Speech Recognition in CSCS....94

Text-to-speech in CSCS....94

Speech recognition in CSCS....96

An example of text-to-speech and speech recognition....100

Summary....102

Chapter 8 In-App Purchases in CSCS....103

www.dbooks.org

Setting up In-App Purchases for iOS 103
Setting up In-App Billing for Android 104
Implementation 105
CSCS example 106
Summary 109
Chapter 9 Adding Mobile Advertising and Advanced Topics 110
Adding Google AdMob to CSCS 110
Debugging and unit testing 112
Localization 115
Settings or user defaults 116
Swipe, long click, and drag and drop 116
Scheduling events 117
Asynchronous programming in CSCS 118
Calling native code from CSCS 118
Conclusion 119
Appendix 120
Currently implemented CSCS functions 120

The Story Behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

www.dbooks.org

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to "enable AJAX support with one click," or "turn the moon to cheese!"

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and "Like" us on Face-book to help us spread the word about the *Succinctly* series!





About the Author

Vassili Kaplan is a former Microsoft Lync developer. He has been studying and working in a few countries, such as Russia, Mexico, the United States, and Switzerland.

He has a B.S. in Applied Mathematics from Instituto Tecnológico Autónomo de México, and an M.S. in Applied Mathematics with Specialization in Computational Sciences from Purdue University, West Lafayette, Indiana.

He currently lives with his wife and two kids in Zurich, Switzerland and works as a freelancer for various Swiss banks. He is mostly programming in C++, C#, Python, and CSCS.

In his spare time, he works on the CSCS language, which is the topic of this book. His other hobbies are traveling, biking, badminton, and enjoying a glass of good red wine.

You can contact him either at vassilik@gmail.com or at his website: http://www.iLanguage.ch.

Chapter 1 Introduction

"Code is like humor. When you have to explain it, it is bad."
Cory House

What will be covered in this book?

In this book, we are going to see how to develop native mobile apps in a customizable scripting language. In my previous e-book, *Implementing a Custom Language Succinctly*, I showed how to create this customizable language. For brevity, I called this language CSCS (Customizable Scripting in C#). My previous e-book is not a prerequisite for this book, although it is a "nice-to-have," since it has a few details about the CSCS language that we will not cover in this book.

Here's what we'll cover:

- How to write native mobile apps for iOS and Android in a scripting language.
- How to place widgets on the screen programmatically (no storyboards, XML, or XAML needed—it's all in a simple scripting language).
- How to add custom widgets to CSCS.
- How to implement basic functionality in CSCS: text-to-speech and voice recognition, in-app purchases, scheduling events, etc.
- How to add other Xamarin Frameworks and NuGet packages that can be used in CSCS.
- How to do Unit Testing in CSCS.
- How to call native C# code from CSCS and get the results back.
- Examples of complete apps written in pure CSCS.

The base developing platform will be Microsoft Visual Studio (either on Windows or on macOS) with Xamarin. I personally use the free one, Visual Studio Community 2017 on macOS.

The CSCS language does not have to be used exclusively for developing a mobile app. It can also be used as a complement to facilitate creating common widgets, their placements, and any other functionality—CSCS is 100 percent customizable.

You can also write the whole app in CSCS—I did it with the iLanguage app. The same CSCS code is used for both iOS and Android. The code for iLanguage app is in the source code download—the entry point is the **iLanguage.cscs** file, which imports a few other CSCS files.

I also wrote an app to test CSCS scripts on iOS and on Android. You can test all of the scripts that you see in this book and also experiment with your own. The app itself is obviously written in CSCS as well. The source code is in the **cscs.cscs** file.

Advantages of using CSCS for mobile

Why should you use a scripting language to write an app?

- There is much less coding. As you will see in this book, it takes only a very few lines of code to place a widget on the screen and populate it with some data and images.
- The same code is used for both iOS and Android. You will also see that other platforms, like Windows Phone, can be easily added to CSCS as well.
- Even though Xamarin.Forms can share up to 75 percent of common code between iOS and Android, or even more if only common controls are used, CSCS goes further than that because it is easier to add features shared by multiple platforms. Once you have C# wrappers over the native code, you use same CSCS code to call them behind the scenes for all platforms.
- Quicker debugging: no need to recompile the code when debugging the app, since the changes are done in the scripting part, not in C#.
- The CSCS parser is not only open source, but also included with every project, so it is easy to modify the existing functionality or add a new one on the fly—compare this with C# or Python.

"Hello, World!" in CSCS

Code Listing 1 contains CSCS code to create a few labels, a button, and an event triggered when the user clicks on a button.

Code Listing 1: "Hello, World!" in CSCS

```
AutoScale();
SetBackgroundColor("cyan");

locClickme = GetLocation("ROOT", "CENTER", "ROOT", "CENTER");
AddButton(locClickme, "buttonClickme", "Click me", 200, 80);
AddAction(buttonClickme, "clickme_click");
SetFontSize(buttonClickme, 12);

locVersionLabel = GetLocation(buttonClickme, "ALIGN_LEFT",
                              buttonClickme, "TOP");
AddLabel(locVersionLabel, "versionLabel", "", 360, 60);
SetFontSize(versionLabel, 12);

locSizeLabel = GetLocation(buttonClickme, "CENTER",
                           buttonClickme, "BOTTOM");
AddLabel(locSizeLabel, "sizeLabel", "", 360, 60);
SetFontSize(sizeLabel, 10);

locOrientationLabel = GetLocation(buttonClickme, "RIGHT",
                                  buttonClickme, "CENTER");
AddLabel(locOrientationLabel, "orientationLabel", "", 160, 80);
SetFontSize(orientationLabel, 10);

clicks = 0;
function clickme_click(sender, arg)
{
  clicks++;
```


www.dbooks.org

```
  if (clicks == 1) {
    SetText(versionLabel, "Hello, " + _DEVICE_INFO_ + ", " +
                          _VERSION_INFO_, "left");
  } elif (clicks == 2) {
    SetText(sizeLabel, "Size: " + DisplayWidth + "x" + DisplayHeight +
                       ". Locale: " + GetDeviceLocale(), "center");
  } elif (clicks == 3) {
    SetText(orientationLabel, "Orientation: " + Orientation, "center");
  } else {
    SetText(buttonClickme, "Clicks: " + clicks, "center");
  }
}
```

See the results of running the “Hello, World!” script on actual devices in Figure 1. The image on the left shows it running on an iPhone 6, and the one on the right shows it running on a Wiko Lenny4 Android phone.

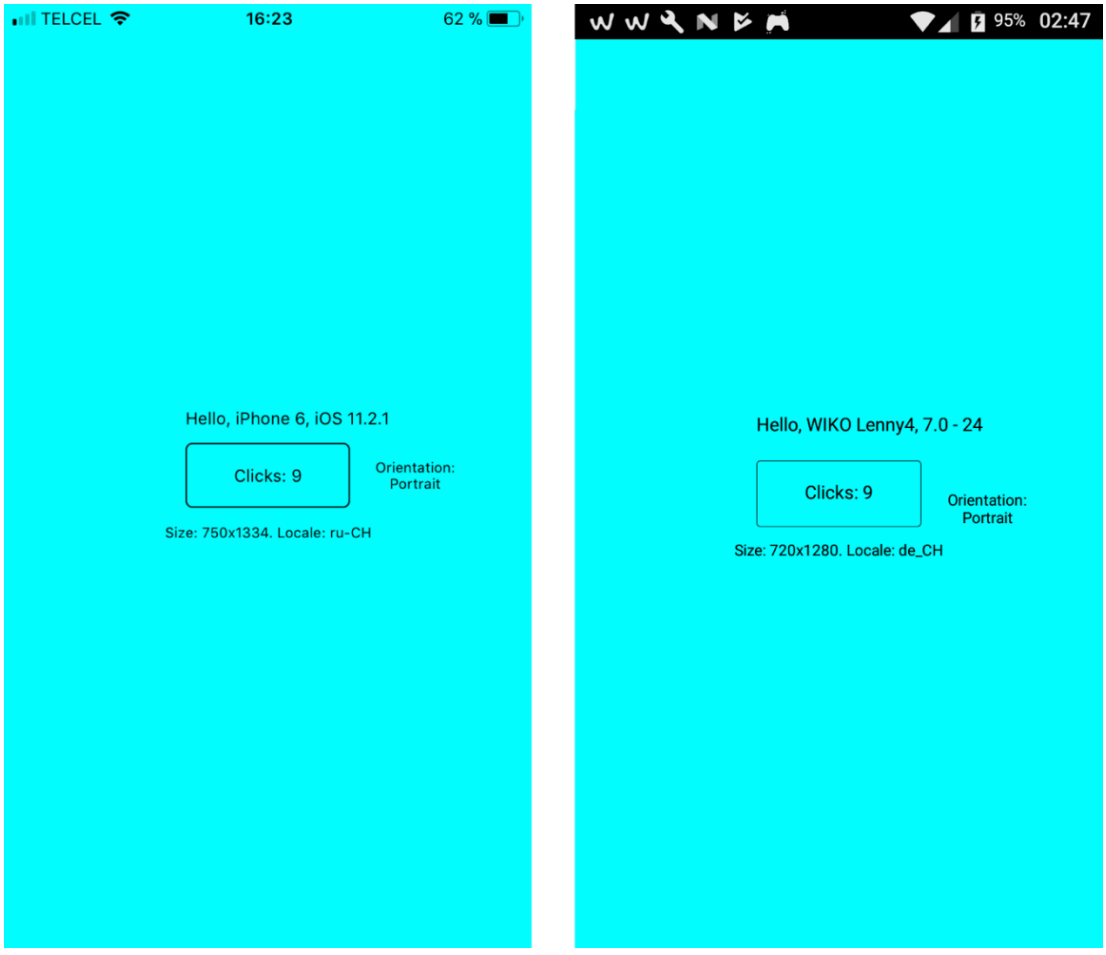


Figure 1: Running the "Hello, World!" Script for iPhone (left) and Android (right)

Let's analyze Code Listing 1. **AutoScale()**scales the widgets according to the screen resolution.



Tip: Using the `AutoScale()` function, the width of a widget will be twice as large on a device with a width of 960 pixels, than on a device with a width of 480 pixels.

You can also use the auto-scale functionality per widget. We will see more details about this function in the next chapter.

SetBackgroundColor("cyan") will change the background color on the running device. There is also a **SetBackground(imageName)** function, which sets an image as a background.

The next two lines create a location and a button, which will be placed in that location:

```
locClickme = GetLocation("ROOT", "CENTER", "ROOT", "CENTER");
AddButton(locClickme, "buttonClickme", "Click me", 200, 80);
```

The general signature for creating a location on the screen is the following:

```
GetLocation(HorizontalReference, PlacementX,
            VerticalReference,   PlacementY);
```

Horizontal or vertical references are just some other widgets on the screen, or a **ROOT**, meaning the main screen.

To connect an action to the event of clicking on a button, the following function is used:

```
AddAction(buttonClickme, "clickme_click");
```

The callback function, in our case **clickme_click**, always has two parameters, a sender and an argument. As you can see, this CSCS concept has been borrowed from C#, as have a few others.

The **clickme_click()** function has a few calls to other functions, namely to: **_DEVICE_INFO_**, **_VERSION_INFO_**, **DisplayWidth**, **DisplayHeight**, **GetDeviceLocale**, and **Orientation**.

All of these functions are implemented in C#. None of them require a parameter—in this case, the opening and closing parentheses after the function name are optional. We will talk about the implementation of some of these functions later on.

www.dbooks.org

Customizable scripting in C#

Syntax-wise, CSCS is a mixture of Python and C#. But CSCS is not object-oriented, even though it has some features of object-oriented languages. One of them is polymorphism: there are many CSCS functions that accept different widgets as arguments, like **`SetFontColor()`**, **`SetValue()`**, **`SetText()`**, and many others. We will talk about these functions later on.

The CSCS language is implemented in C# and it is based on the *split-and-merge* algorithm. Let's briefly review the split-and-merge algorithm, which forms a basis of CSCS. You can check out the complete description here.

The algorithm consists of two steps. In the first step, the string containing an expression is split into a list of so-called "variables." Each variable consists of a number or a string and an action that must be applied to it. The numbers are internally represented as doubles, so in essence, they can be integers or Booleans.

Actions are different for numbers and strings. For numbers, the actions can be all possible operations we can do with the numbers, for example, **+**, **–**, *****, **/** or **%**, Boolean operators (**!**, **&&**, **||**), and bitwise operations (**&**, **|**, **^**). For strings, the actions can be a **+** (string concatenation) or Boolean comparisons. For convenience, we denote the action of the last variable in the list as **)**.

As soon as we get any function or an opening parentheses in the expression string, we recursively apply the whole split-and-merge algorithm to the expression in parentheses or to a function, and then replace them with the calculated result.

At the end of the first step, we are going to have a list of variables, each one consisting of either a number or a string, and an action that will be applied.

In the second step, we merge the list of variables created in the first step. This list is merged one by one, starting from the first variable in the list. Merging means applying the action of the left variable to the values of the left and right variables. The resulting variable will have the same action as the right variable.

The variables can only be merged if the priority of the action of the left variable is greater than or equal to the priority of the action of the right variable (for instance, the priority of multiplication is greater than the priority of addition). If two variables cannot be merged, we temporarily move to the next-right variable, in order to try to merge it with the variable next to it, and so on, recursively. As soon as the right variable has been merged with the variable on its right, we return to the original, left variable, and try to re-merge it with the newly created right variable.

Eventually, we will be able to merge the whole list into one variable, since sooner or later we will reach the last variable of the list that has the lowest priority, and therefore, can be merged with any variable on its left. The value of the remaining variable will be the final result.

An example of implementing a CSCS function

Let's see an example of the implementation of the **_DEVICE_INFO_** function that you saw in Code Listing 1. The expected outcome of this function is the name of the device being used (for example, iPhone 6, or Wiko Lenny4).

There are just two steps in adding a new function to CSCS:

1. Implement the **Evaluate()** method of a class deriving from the **ParserFunction** class.
2. Register the class implemented in the previous step with the parser.

The implementation is different for iOS and Android. Code Listing 2 has an Android implementation. The iOS implementation is unfortunately, and surprisingly, much more complicated, at least as of iOS 11.2, and will be skipped (the curious reader can check it out in the accompanying source code, which can be downloaded).

Code Listing 2: Implementation of the GetDeviceInfoFunction Class for Android

```
class GetDeviceInfoFunction: ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    string deviceName = Android.OS.Build.Brand;
    string model      = Android.OS.Build.Model;
    if (!model.Contains("Android")) {
      // Simulators have "Android" in both, Brand and Model.
      deviceName += " " + model;
    }

    return new Variable(deviceName);
  }
}
```

Once a class, deriving from the **ParserFunction** class, is implemented, we can register it with the parser as follows:

```
ParserFunction.RegisterFunction("_DEVICE_INFO_",
                                new GetDeviceInfoFunction());
```

The general signature is:

```
ParserFunction.RegisterFunction(functionName, aParserFunction);
```

This way you can register any function with the parser. We are going to see many more examples of this later on in this book.

www.dbooks.org

Summary

In this chapter we saw the split-and-merge algorithm and how it is used to implement a typical CSCS function. This algorithm is used behind the scenes in all CSCS functions, but you do not have to know it to use it directly—it's all done behind the scenes by the CSCS parser.

In the next chapter, we are going to see the overall Xamarin project structure that can be used for customized scripting with CSCS.

Chapter 2 Project Structure and Design

"Design is how it works."
Steve Jobs

Structure of a CSCS mobile project

Figure 2 contains a screenshot of a common Xamarin project using CSCS.

Scripting.Shared is a shared project containing the code common to both Android and iOS. In particular, it contains all the files needed for parsing the CSCS code. **Parsing.cs** contains the implementation of the split-and-merge algorithm.

ParserFunction.cscs contains the implementation of the base class **`ParserFunction`**. When adding a new functionality to CSCS, a new class will override the **`ParserFunction`** class (see Code Listing 2 in the previous chapter).

Functions.Flow.cs, **Functions.Math.cs**, and **Functions.OS.cs** contain the implementation of the basic functions, not related to the mobile devices (operations with strings, loops, exceptions, mathematical functions, etc.). **Interpreter.cs** is a class that glues together parsing functions with the parsing flow.

The **Shared.Resources** folder contains resources used by both platforms. In particular, it contains the CSCS scripts—I placed them directly under the Resources folder. For example, the code mentioned in Code Listing 1 is located in the **Resources\fhello.cscs** file. The Android and iOS projects have links to this file.

The **drawable** subfolder contains all the images used by both platforms. Each particular project, iOS, or Android, has its own Resources folder with links to the shared projects—therefore, changes in the shared folder will be automatically propagated to the iOS and Android folders.

You can also set up the project structure differently, for example, with each project having its own set of images of different sizes, depending on the screen size.

The functions common to iOS and Android CSCS are in the **CommonFunctions.cs** file in the shared project folder.

If iOS and Android code is different, the implementations are in the **iOSFunctrions.cs** file for iOS and in the **DroidFunctions.cs** file for Android, respectively.

If the code is different on both platforms, but the differences are minimal, you can still have it in the **CommonFunctions.cs** file using the preprocessor macros. See Code Listing 3 for an example of extracting the version number for an Android and an iOS device.

www.dbooks.org

Code Listing 3: Implementation of the GetVersionNumberFunction Class

```
class GetVersionNumberFunction: ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
#if __ANDROID__
    string strVersion = Android.OS.Build.VERSION.Release;
#elif __IOS__
    string strVersion = UIKit.UIDevice.CurrentDevice.SystemVersion;
#endif
    return new Variable(strVersion);
  }
}
```

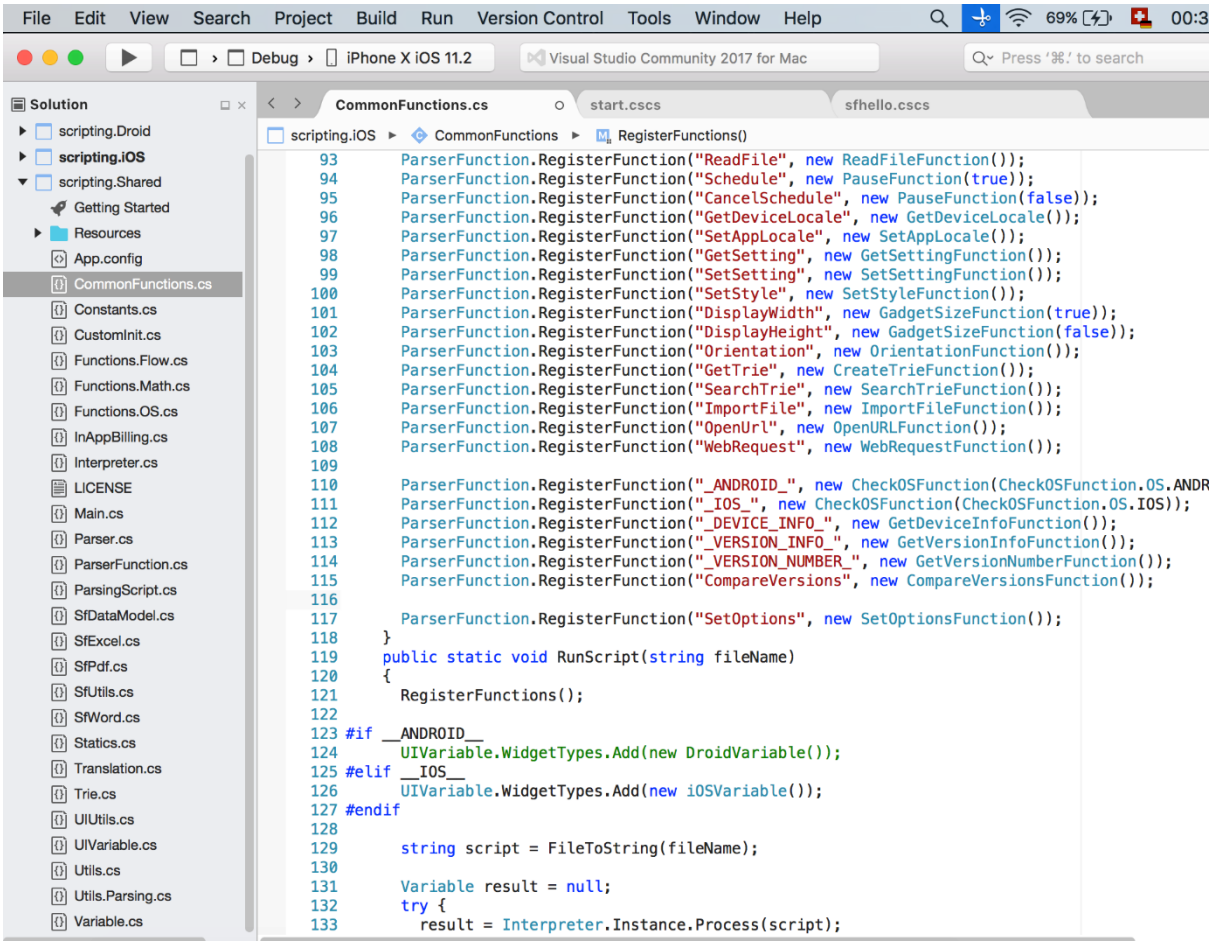


Figure 2: General Structure of a CSCS Xamarin Project (On macOS)

Where CSCS code executes

The actual execution of the CSCS code happens inside of the **`CommonFunctions.RunScript()`** method shown in Code Listing 4. This method also registers all the common functions with the parser.

Code Listing 4: The Implementation of the RunScript Method

```
public static void RunScript(string fileName)
{
  RegisterFunctions();
  string script = FileToString(fileName);

  Variable result = null;
  try {
    result = Interpreter.Instance.Process(script);
  } catch (Exception exc) {
    Console.WriteLine("Exception: " + exc.Message);
    Console.WriteLine(exc.StackTrace);
    ParserFunction.InvalidateStacksAfterLevel(0);
    throw;
  }
}

public static void RegisterFunctions()
{
  ParserFunction.RegisterFunction("GetLocation",
                                  new GetLocationFunction());
  ParserFunction.RegisterFunction("AddWidget", new AddWidgetFunction());
  // ... all other functions ...
}
```

`CommonFunctions.RunScript()` is called from **`CustomInit.InitAndRunScript()`**, which initializes project-specific data and registers all project-specific widget types and all custom functions with the parser. See Code Listing 5 for an example of such a method.

Code Listing 5: The Implementation of the CustomInit.InitAndRunScript Method

```
public static void InitAndRunScript()
{
  UIVariable.WidgetTypes.Add(new AdMob());
  UIVariable.WidgetTypes.Add(new SfWidget());

  string fileName = "start.cscs";

  // All custom functions go here ...
  ParserFunction.RegisterFunction("SfPdfNew", new CreatePdf());
  ParserFunction.RegisterFunction("SfPdfOpen", new OpenPdf());
```


www.dbooks.org

```
  ParserFunction.RegisterFunction("SfSetPdfText", new SetPdfText());
  ParserFunction.RegisterFunction("SfSetPdfImage", new SetPdfImage());
  ParserFunction.RegisterFunction("SfSetPdfLine", new SetPdfLine());
  ParserFunction.RegisterFunction("SfSetPdfPie", new SetPdfPie());
  ParserFunction.RegisterFunction("SfSetPdfFont", new SetPdfFont());
  ParserFunction.RegisterFunction("SfSavePdf", new SavePdf());
  // ...

  CommonFunctions.RunScript(fileName);
}
```

In particular, it specifies that the **`start.cscs`** script will run. Note that you can import other CSCS script files by calling the **`ImportFile()`** CSCS function from **start.cscs**, for example:

```
ImportFile("sfhello.cscs");
```

> ***Tip: In this way you can have different apps with identical C# code but different CSCS code. You just specify a different name when calling the `ImportFile()` function in start.cscs. There are some examples of this in the accompanying source code.***

Where in the workflow do we execute the CSCS script? The answer is different for iOS and Android.

For Android, it's a bit trickier than for iOS: we have to execute the script after the global layout has been initialized so that the script not only can start adding widgets, but also has access to all of the layout information, like sizes, orientation, main window parameters, and so on.

Unfortunately, it's too early to run the CSCS script at the end of the **`MainActivity.OnCreate()`** method, but fortunately, Android provides a way: we can register a Global Layout listener with a **`ViewTreeObserver`** object.

We register it not at the end of the **`MainActivity.OnCreate()`** method, but at the end of the **`MainActivity.OnResume()`** method, which is the best place according to the Android activity lifecycle. See
Code Listing 6 for details.

Code Listing 6: Running a CSCS Script on Android

```
// MainActivity
bool m_scriptRun = false;

protected override void OnResume()
{
  base.OnResume();
  if (!m_scriptRun) {
    ViewTreeObserver vto = TheLayout.ViewTreeObserver;
    vto.AddOnGlobalLayoutListener(new LayoutListener());
```

```
    m_scriptRun = true;
  }
}

public class LayoutListener : Java.Lang.Object,
                              ViewTreeObserver.IOnGlobalLayoutListener
{
  public void OnGlobalLayout()
  {
    var vto = MainActivity.TheLayout.ViewTreeObserver;
    vto.RemoveOnGlobalLayoutListener(this);

    // Run CSCS Script here:
    CustomInit.InitAndRunScript();
  }
}
```

We use the Boolean variable **`m_scriptRun`** to make sure that the CSCS script is executed only once, because we do not need to run the script every time the device orientation changes. In this case, we run only specific functions responsible for widget placement—we'll see how to do that in the next chapter.



Note: Remove the `m_scriptRun` Boolean check if you want the script to run every time the device orientation changes or the application comes back from the background.

The situation is simpler in iOS—we can execute the CSCS script from the **`AppDelegate.FinishedLaunching()`** method.

In iOS, I created a custom view controller, with the method **`Run()`**, which actually runs the CSCS script: see Code Listing 7.

Code Listing 7: Running CSCS Script on iOS

```
public void Run()
{
  // If there is no TabBar, move the tabs view down:
  OffsetTabBar();
  this.ViewControllerSelected += OnTabSelected;

  CustomInit.InitAndRunScript();

  if (m_selectedTab >= 0) {
    SelectTab(m_selectedTab);
  } else if (TabBar != null) {
    TabBar.Hidden = true;
  }
}
```


www.dbooks.org

The base class for all of the widgets is **`UIVariable`**, which derives from the core CSCS variable class. It has a common functionality for iOS and for Android. A fragment of this class is described in
Code Listing 8.

Code Listing 8: A Fragment of the UIVariable Class

```
public class UIVariable : Variable
{
  public Action<string, string> ActionDelegate;
  public static List<UIVariable> WidgetTypes = new List<UIVariable>();
  protected static int m_currentTag;

  public enum UIType {
    NONE, LOCATION, VIEW, BUTTON, LABEL, TEXT_FIELD, TEXT_VIEW,
    PICKER_VIEW, PICKER_IMAGES,LIST_VIEW, COMBOBOX, IMAGE_VIEW,
    SWITCH, SLIDER, STEPPER, SEGMENTED, CUSTOM
  };

  public UIVariable(UIType type, string name = "",
                    UIVariable refViewX = null,
                    UIVariable refViewY = null)
  {
    WidgetType = type;
    WidgetName = name;
    RefViewX   = refViewX;
    RefViewY   = refViewY;
  }

  public UIType WidgetType     { get; set; }
  public string WidgetName     { get; set; }
  public int Width             { get; set; }
  public int Height            { get; set; }
  public int X                 { get; set; }
  public int Y                 { get; set; }
  public int TranslationX      { get; set; }
  public int TranslationY      { get; set; }

  public string RuleX          { get; set; }
  public string RuleY          { get; set; }

  public UIVariable Location   { get; set; }
  public UIVariable RefViewX   { get; set; }
  public UIVariable RefViewY   { get; set; }
  public UIVariable ParentView { get; set; }

  public double MinVal         { get; set; }
  public double MaxVal         { get; set; }
```

```
  public double CurrVal        { get; set; }
}
```

The **iOSVariable** class derives from **UIVariable**, and has a concrete implementation for iOS. Its fragment is shown in Code Listing 9. Its Android equivalent is shown in Code Listing 10.

Code Listing 9: A Fragment of the iOSVariable Class

```
public class iOSVariable : UIVariable
{
  public iOSVariable(UIType type, string name,
                     UIView viewx = null, UIView viewy = null) :
                     base(type, name)
  {
    m_viewX = viewx;
    m_viewY = viewy;
    if (type != UIType.LOCATION && m_viewX != null) {
      m_viewX.Tag = ++m_currentTag;
    }
  }

  public virtual iOSVariable GetWidget(string widgetType,
                   string widgetName, string initArg, CGRect rect)
  { /* ... */ }
  public virtual bool SetValue(string value1, string value2 = "")
  { /* ... */ }
  public virtual double GetValue()
  {
    double result = 0;
    if (m_viewX is UISwitch) {
      result = ((UISwitch)m_viewX).On ? 1 : 0;
    } else if (m_viewX is UISlider) {
      result = ((UISlider)m_viewX).Value;
    } else if (m_viewX is UIStepper) {
      result = ((UIStepper)m_viewX).Value;
    } else if (m_viewX is UIPickerView) {
      result = ((TypePickerViewModel)(((UIPickerView)m_viewX).Model)).
                 SelectedRow;
    } else if (m_viewX is UISegmentedControl) {
      result = ((UISegmentedControl)m_viewX).SelectedSegment;
    }
    /* And so on ... */
    return result;
  }
  public virtual bool SetText(string text, string alignment = null)
  { /* ... */ }
  public virtual string GetText()
  { /* ... */ }
```


www.dbooks.org

```
  public virtual void AddData(List<string> data, string varName,
                              string title, string extra)
  { /* ... */ }
  public virtual void AddImages(List<UIImage> images,
                                string varName, string title)
  { /* ... */ }
  public virtual bool SetFont(string name, double size = 0)
  { /* ... */ }
  public virtual bool SetFontSize(double val)
  { /* ... */ }
  public virtual bool SetFontColor(string colorStr)
  { /* ... */ }
  public virtual bool SetBold(double size = 0)
  { /* ... */ }
  public virtual bool SetItalic(double size = 0)
  { /* ... */ }
  public virtual bool SetNormalFont(double size = 0)
  { /* ... */ }
  public virtual bool SetBackgroundColor(string colorStr,
                                         double alpha = 1.0)
  { /* ... */ }
  public static UIView GetView(string viewName, ParsingScript script)
  {/* ... */}
  public UIView GetParentView()
  {
    iOSVariable parent = ParentView as iOSVariable;
    if (parent != null) {
      return parent.ViewX;
    }
    return AppDelegate.GetCurrentView();
  }

  UIView m_viewX;
  UIView m_viewY;
}
```

Code Listing 10: A Fragment of the DroidVariable Class

```
public class DroidVariable: UIVariable
{
  public DroidVariable(UIType type, string name, View viewx = null,
                       View viewy = null) : base(type, name)
  {
    m_viewX = viewx;
    m_viewY = viewy;
    if (type != UIType.LOCATION && m_viewX != null) {
      m_viewX.Tag = ++m_currentTag;
      m_viewX.Id = m_currentTag;
```

```
    }
  }

  public void SetViewLayout(int width, int height)
  {
    DroidVariable refView = Location?.RefViewX as DroidVariable;
    m_viewX = MainActivity.CreateViewLayout(width, height,
                                            refView?.ViewLayout);
  }

  public virtual DroidVariable GetWidget(string widgetType,
                                string widgetName, string initArg,
                                           int width, int height)
  { /* ... */ }
  public static Size GetLocation(View view)
  {
    if (view == null) {
      return null;
    }
    int[] outArr = new int[2];
    view.GetLocationOnScreen(outArr);
    return new Size(outArr[0], outArr[1]);
  }

  public static View GetView(string viewName, ParsingScript script)
  {
    if (viewName.Equals("root", StringComparison.OrdinalIgnoreCase)) {
      return null;
    }
    ParserFunction func = ParserFunction.GetFunction(viewName);
    Variable viewValue = func.GetValue(script);
    DroidVariable viewVar = viewValue as DroidVariable;
    return viewVar.ViewX;
  }

  /* Same SetXXX and GetXXX functions as for iOSVariable class... */

  View m_viewX;
  View m_viewY;
  LayoutRules m_layoutRuleX;
  LayoutRules m_layoutRuleY;
}
```

Summary

In this chapter, we reviewed the general structure of a project using CSCS scripting and where in the project workflow the CSCS code is executed. It is done differently for iOS and for Android.

www.dbooks.org

We skipped a few details, since there is really a lot of code. However, since the devil is in the details, for the full picture, check out the accompanying source code.

In the next chapter, we are going to dig into the details of placing widgets on the screen in a platform-independent way.

Chapter 3 Placing Widgets on the Screen

"Any sufficiently advanced technology is indistinguishable from magic."

Arthur C. Clarke

Layout concepts in CSCS

Different frameworks use different layout methods for placing widgets on the screen—most allow you to place widgets on the screen programmatically. Many also allow you to visually drag-and-drop a widget anywhere on the screen.

For developing iOS apps with XCode, Apple created the Auto Layout concept and the Storyboard. The widgets are placed on the screen according to the constraints among themselves. When you run your app on a device or in a simulator, the layout may look different from what you see on a Storyboard, where you dropped your widgets. This is due to the different sizes of the Storyboard for different devices.

For Android development, the layout is often created defining the placement rules in the XML layout files. Xamarin.Forms uses XAML files.

For CSCS development, we won't have a familiar drag-and-drop functionality. All the placements of widgets are done programmatically. It is the same code for Android and for iOS.

In exchange, we will have more control over where to place the widgets. The learning curve is quicker—there are no XML schemas, no XAML files, and no Auto Layouts (I personally think anyone who understands the workable concept of Auto Layouts in under a week is a genius of our time).

For the layout definition, I used a mixture of the iOS and Android approaches. From the Auto Layout, I applied the following—for the unique widget location, I need to have three concepts:

- Widget horizontal placement (relative to other widgets or to the main window).
- Widget vertical placement (relative to other widgets or to the main Window).
- Widget size.

The widget size can be absolute or relative. In the latter case, it will be automatically extended or contracted, depending on the device size.

That's it! I am not sure why the horizontal and vertical placements are allowed to be defined multiple times, and inconsistently, in both iOS and Android, leading to conflicts; these conflicts may be resolved with unexpected results during runtime. In CSCS, no conflicts are possible by its layout design (how it works).

For the implementation, I took an approach similar to the concept of the Relative Layout in Android. The difference with **`RelativeLayout`** is that it's not possible to have multiple

www.dbooks.org

definitions of the widget placement parameters—in Android, you can use the **`ApplyRule()`** method for setting the widget placement rules an unlimited number of times, and Android won't complain about any contradicting rules.

Let's now start with the implementation. Some parts of what follows have been published in CODE Magazine and MSDN Magazine.

Locations in CSCS

A widget definition in CSCS has either one or two statements. The first statement is the definition of a location—where to place the widget—and the second one is the actual widget definition, including its size.



Tip: The same location can be used for different widgets.

One scenario is to show the user multiple widgets, one at a time, located in the same place—an example of that is shown in **sfscript.cscs**.

We already saw an example of placing widgets in Code Listing 1:

```
locSizeLabel = GetLocation(buttonClickme, "CENTER",
                           buttonClickme, "BOTTOM");
AddLabel(locSizeLabel, "sizeLabel", "", 360, 60);
```

The general syntax of the location command is the following:

```
GetLocation(HorizontalReference, PlacementX,
            VerticalReference,   PlacementY,
            AdjustmentX = 0,     AdjustmentY = 0,
            UseScale = true,     ScaleValue = 0.0,
            Parent = null);
```

Here is the meaning of the parameters:

- **`HorizontalReference`**: The name of another widget for the horizontal placement. It can be the string **`ROOT`**, meaning the parent widget or the main screen.
- **`PlacementX`**: A horizontal placement relative to the widget in the **`HorizontalReference`**. We'll discuss possible values below.
- **`VerticalReference`**: The name of another widget for the vertical placement. It can be the string **`ROOT`**, meaning the parent widget or the main screen.
- **`PlacementY`**: A vertical placement relative to the widget in the **`VerticalReference`**.

- **AdjustmentX**: The number of pixels the widget must be moved horizontally (this number can be relative, depending on the **AutoScale** option which we will discuss below). It can also be negative: the positive direction goes from left to right.

- **AdjustmentY**: The number of pixels the widget must be moved vertically. It can also be negative: the positive direction goes from top to bottom. See also **AutoScale** below.

- **UseScale**: Whether to apply a scaling option to the widget. If **true**, the adjustment specified in **ScaleValue**, discussed next, will be used.

- **ScaleValue**: The measure used for adjusting the size of this widget. It overrides the value provided as an argument for the **AutoScale()** function.

- **Parent**: The parent of this widget. If null, the widget will be added to the Main Layout on Android or to the Root View Controller View on iOS.

The possible values for the **PlacementX** and **PlacementY** parameters are very similar to the constants of the `Android` **RelativeLayout.LayoutParams** class.

They can be any of the following: **CENTER**, **LEFT**, **RIGHT**, **TOP**, **BOTTOM, ALIGN_LEFT**, **ALIGN_RIGHT**, **ALIGN_TOP**, **ALIGN_BOTTOM**, **ALIGN_PARENT_LEFT**, **ALIGN_PARENT_RIGHT**, **ALIGN_PARENT_TOP**, or **ALIGN_PARENT_BOTTOM**.

The **GetLocation()** function is used for both iOS and Android. Check out its implementation for iOS in Code Listing 11. The Android implementation is similar.

Code Listing 11: Implementation of the GetLocationFunction Class on iOS

```
public class GetLocationFunction : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 4, m_name);

    string viewNameX = args[0].AsString();
    string ruleStrX  = args[1].AsString();
    string viewNameY = args[2].AsString();
    string ruleStrY  = args[3].AsString();

    int leftMargin   = Utils.GetSafeInt(args, 4);
    int topMargin    = Utils.GetSafeInt(args, 5);

    bool autoResize  = Utils.GetSafeInt(args, 6, 1) == 1;
    if (autoResize) {
      double multiplier = Utils.GetSafeDouble(args, 7);
      AutoScaleFunction.TransformSizes(ref leftMargin, ref topMargin,
                  (int)UtilsiOS.GetRealScreenWidth(), multiplier);
    }
```


www.dbooks.org

```
    Variable parentView = Utils.GetSafeVariable(args, 8, null);

    UIView referenceViewX = iOSVariable.GetView(viewNameX, script);
    UIView referenceViewY = iOSVariable.GetView(viewNameY, script);
    iOSVariable location = new iOSVariable(UIVariable.UIType.LOCATION,
                viewNameX + viewNameY, referenceViewX, referenceViewY);

    location.SetRules(ruleStrX, ruleStrY);
    location.ParentView = parentView as UIVariable;

    double screenRatio = UtilsiOS.GetScreenRatio();
    location.TranslationX = (int)(leftMargin / screenRatio);
    location.TranslationY = (int)(topMargin / screenRatio);

    return location;
  }
}
```

To register the **`GetLocation()`** function with the parser, the following command must be added in the initialization phase (see Code Listing 4):

```
ParserFunction.RegisterFunction("GetLocation", new GetLocationFunction());
```

Adding widgets

Once we have a location, we can create a widget. The general structure of this function is:

```
AddWidget(widgetType, location, widgetName, initString, width, height);
```

There are shortcuts of this command for each widget type, for example:

```
AddLabel(location,     widgetName, initString, width, height);
AddButton(location,    widgetName, initString, width, height);
AddImageView(location, widgetName, initString, width, height);
```

And so on. Calling the functions above is equivalent to calling the following:

```
AddWidget("Label",     location, widgetName, initString, width, height);
AddWidget("Button",    location, widgetName, initString, width, height);
AddWidget("ImageView", location, widgetName, initString, width, height);
```

Here is the meaning of the parameters:

- **`location`**: The location of the widget, defined in the previous section.
- **`widgetName`**: the name of the widget. It will be always a global variable, even if used inside of a function.

- **`initString`**: The initialization parameter of the widget. For example, it will be a text on a label or a title on a button.
- **`width`**: The width of the widget.
- **`height`**: The height of the widget.

Check out a fragment of the **`AddWidget()`** function implementation in Code Listing 12. The implementation is for iOS. The Android implementation is very similar.

Code Listing 12: A Fragment of the AddWidgetFunction Class

```
public class AddWidgetFunction : ParserFunction
{
  public AddWidgetFunction(string widgetType = "", string extras = "")
  {
    m_widgetType = widgetType;
    m_extras = extras;
  }
  protected override Variable Evaluate(ParsingScript script)
  {
    string widgetType = m_widgetType;
    int start = string.IsNullOrEmpty(widgetType) ? 1 : 0;
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 2 + start, m_name);

    if (start == 1) {
      widgetType = args[0].AsString();
      Utils.CheckNotEmpty(script, widgetType, m_name);
    }

    iOSVariable location = args[start] as iOSVariable;
    Utils.CheckNotNull(location, m_name);

    double screenRatio = UtilsiOS.GetScreenRatio();

    string varName = args[start + 1].AsString();
    string config = Utils.GetSafeString(args, start + 2);
    int width = (int)(Utils.GetSafeInt(args, start + 3) / screenRatio);
    int height = (int)(Utils.GetSafeInt(args, start + 4) / screenRatio);

    bool autoResize = Utils.GetSafeInt(args, start + 5, 1) == 1;
    if (autoResize) {
      double multiplier = Utils.GetSafeDouble(args, start + 6);
      AutoScaleFunction.TransformSizes(ref width, ref height,
      (int)UtilsiOS.GetRealScreenWidth(), multiplier);
    }

    CGSize parentSize = location.GetParentSize();

    location.X = UtilsiOS.String2Position(location.RuleX,
                    location.ViewX, location, parentSize, true);
```


www.dbooks.org

```
    location.Y = UtilsiOS.String2Position(location.RuleY,
                    location.ViewY, location, parentSize, false);

    location.X += location.TranslationX;
    location.Y += location.TranslationY;

    CGRect rect = new CGRect(location.X, location.Y, width, height);

    iOSVariable widgetFunc = GetWidget(widgetType, varName,config, rect);
    var currView = location.GetParentView();
    currView.Add(widgetFunc.ViewX);

    iOSApp.AddView(widgetFunc);

    ParserFunction.AddGlobal(varName, new GetVarFunction(widgetFunc));
    return widgetFunc;
  }

  public static iOSVariable GetWidget(string widgetType,
                  string widgetName, string initArg, CGRect rect)
  {
    for (int i = 0; i < UIVariable.WidgetTypes.Count; i++) {
      iOSVariable var = UIVariable.WidgetTypes[i] as iOSVariable;
      var widget = var.GetWidget(widgetType, widgetName, initArg, rect);
      if (widget != null) {
        return widget;
      }
    }
    return null;
  }
}
```

The code responsible for the actual creation of the widgets is in the **`iOSVariable`** class. Its fragment is shown in Code Listing 13.

For complete implementation, including the implementation of the function **`UtilsiOS.String2Position()`**, which returns an actual point on the screen, refer to the accompanying source code.

Code Listing 13: A Fragment of the iOSVariable.GetWidget Method

```
public virtual iOSVariable GetWidget(string widgetType, string widgetName,
                            string initArg, CGRect rect)
{
  UIVariable.UIType type = UIVariable.UIType.NONE;
  UIView widget = null;
  iOSVariable widgetFunc = null;
  switch (widgetType) {
```

```
    case "Button":
      type = UIVariable.UIType.BUTTON;
      widget = new UIButton(rect);
      ((UIButton)widget).SetTitle(initArg, UIControlState.Normal);
      AddBorderFunction.AddBorder(widget);
      break;
    case "Label":
      type = UIVariable.UIType.LABEL;
      widget = new UILabel(rect);
      ((UILabel)widget).Text = initArg;
      break;
    case "TextEdit":
    //All other widgets go here ...
  }
  if (widgetFunc == null) {
    widgetFunc = new iOSVariable(type, widgetName, widget);
  }
  return widgetFunc;
}
```

To register the functionality of adding new widgets with the parser, the following commands must be added in the initialization phase (see Code Listing 4):

```
ParserFunction.RegisterFunction("AddWidget", new AddWidgetFunction());
ParserFunction.RegisterFunction("AddButton",
                                new AddWidgetFunction("Button"));
ParserFunction.RegisterFunction("AddLabel",
                                new AddWidgetFunction("Label"));
```

And so on, for each widget.

Implementing device orientation changes

It is not so straightforward to write an app that has different layouts on different device orientations. Even the heavyweights, like LinkedIn, Facebook, Amazon, Uber, The Weather Channel, and many others, have implemented their apps only in the Portrait mode (at least for iOS, at the time of this writing); nothing happens with these apps when you change your device orientation.

In this section our goal is to simplify changing the layouts when the device orientation changes. Here we are going to see the Android implementation (you can find the iOS implementation in the accompanying source code).

First of all, we must add the following in the **`MainActivity`** class definition:

```
ConfigurationChanges = ConfigChanges.ScreenSize |
                       ConfigChanges.Orientation |
```


www.dbooks.org

```
ConfigChanges.KeyboardHidden
```

Basically, it means that the app will handle the orientation changes by itself. Without this change, the **MainActivity.OnCreate()** method would be called every time there is a change in the orientation (there are also some other cases when it would be called, in particular when the app comes back from the background to the foreground).

Next, the **MainActivity.OnConfigurationChange()** method is overridden:

```
public override void OnConfigurationChanged(Configuration newConfig)
{
  OnOrientationChange?.Invoke(newConfig.Orientation ==
                        Orientation.Portrait ? "Portrait" : "Landscape");
  base.OnConfigurationChanged(newConfig);
}
```

It uses the **OnOrientationChange** event handler, defined as follows:

```
public delegate void OrientationChange(string newOrientation);
public static OrientationChange OnOrientationChange;
```

Now we are ready to register the CSCS functions to be triggered on orientation changes:

```
ParserFunction.RegisterFunction("RegisterOrientationChange",
                              new RegisterOrientationChangeFunction());
```

Code Listing 14 shows the implementation of the **RegisterOrientationChangeFunction** class.

Code Listing 14: Implementation of the RegisterOrientationChangeFunction Class

```
public class RegisterOrientationChangeFunction : ParserFunction
{
  static string m_actionPortrait;
  static string m_actionLandscape;
  static string m_currentOrientation;

  protected override Variable Evaluate(ParsingScript script)
  {
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 2, m_name);

    m_actionPortrait  = Utils.GetSafeString(args, 0);
    m_actionLandscape = Utils.GetSafeString(args, 1);
    bool startNow     = Utils.GetSafeInt(args, 2, 1) != 0;

    if (startNow) {
      PerformAction(MainActivity.Orientation, true);
    }
```

```
    MainActivity.OnOrientationChange += (newOrientation) => {
      DeviceRotated();
    };
    return Variable.EmptyInstance;
  }
  static void DeviceRotated()
  {
    string currentOrientation = MainActivity.Orientation;
    if (m_currentOrientation == currentOrientation) {
      return;
    }

    PerformAction(currentOrientation);
  }
  static void PerformAction(string orientation, bool isInit = false)
  {
    m_currentOrientation = orientation;
    int currentTab = MainActivity.CurrentTabId;

    if (!isInit) {
      MainActivity.RemoveAll();
    }

    string action = orientation.Contains("Portrait") ?
                                m_actionPortrait: m_actionLandscape;

    UIVariable.GetAction(action, "\"ROOT\"",
            "\"" + (isInit ? "init" : m_currentOrientation) + "\"");

    if (!isInit && currentTab >= 0) {
      MainActivity.TheView.ChangeTab(currentTab);
    }
  }
}
```

The **`UIVariable.GetAction()`** function executes the passed action. The action is usually the name of a CSCS function. The next section shows an example of changing the device layout on orientation changes.

Example: Currency Convertor

In this section, we are going to create the Currency Convertor app in CSCS from scratch. As the source of the currency rates, we'll be using the currencylayer.com service. Their site provides an easy-to-use web service, with the first 1,000 requests per month provided for free. It also provides the exchange rates for Bitcoin, which will be the theme of this app. To get data, you

www.dbooks.org

need a unique key, which you must supply with every request. You get that key upon registering at their website.

Let's first see how the app looks in portrait mode in Figure 3, and in landscape mode in Figure 4, for both iPhone and Android phones.

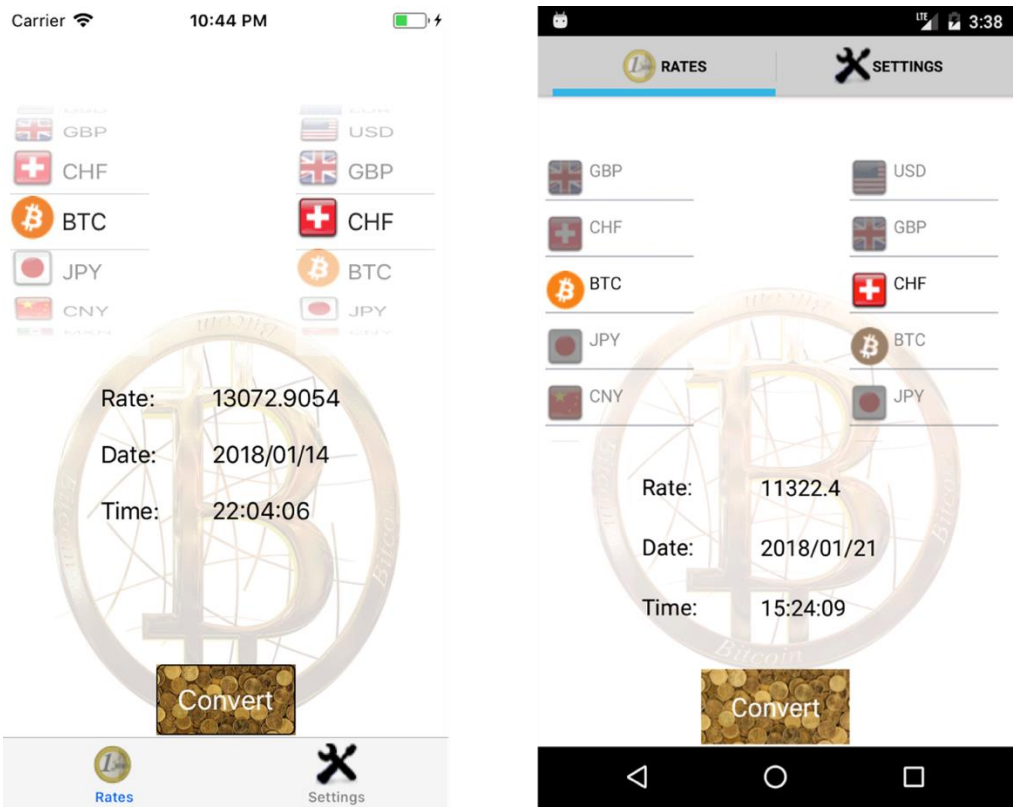


Figure 3: Currency Convertor in the Portrait Mode for iOS (left) and Android (right)

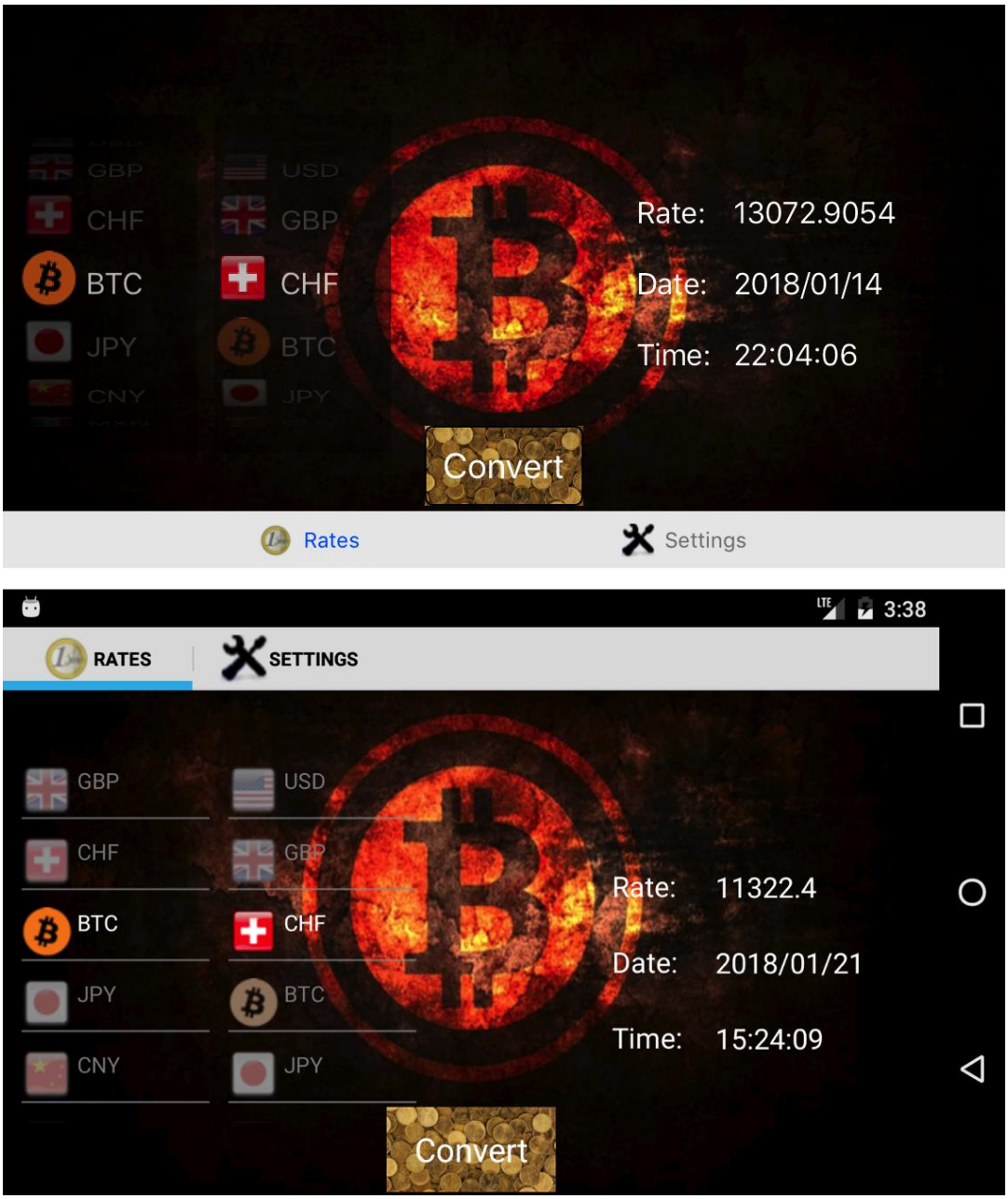


Figure 4: Currency Convertor in the Landscape Mode for iOS (top) and Android (bottom)

Code Listing 15 contains the complete app implementation. The functions **`on_about`** and **`on_refresh`** are the callbacks registered with the parser via the **`AddAction`** function.

The **`on_about`** function will be executed when the user clicks on the **Powered by** button in the **Settings** tab (this tab is not shown on the screenshots). The **`on_refresh`** function is called when the user clicks on the **Convert** button—the actual rate calculation will be done then.

www.dbooks.org

Code Listing 15: The Implementation of the Currency App in CSCS

```
function on_about(sender, arg)
{
  OpenUrl("http://www.currencylayer.com");
}

function on_refresh(sender, arg)
{
  currency1 = GetText(cbCurrency);
  currency2 = GetText(cbCurrency2);
  currency_request(currency1, currency2);
}

function currency_request(currency1, currency2)
{
  if (currency1 == currency2) {
    // A shortcut for the same currency:
    time = Now("HH:mm:ss");
    date = Now("yyyy/MM/dd");
    rate = 1;
  } else {
    url = apiUrl + currency1 + "," + currency2;
    try {
      data = WebRequest(url);
    } catch (exception) {
      WriteConsole(exception.Stack);
      ShowToast("Couldn't get rates. " + exception);
      SetText(labelRateValue, "Error");
      return;
    }

    // To parse: {"success":true,"timestamp":1515937446,"source":"USD","quo
tes":{"USDEUR":0.819304,"USDCHF":0.967604}}
    WriteConsole(data);

    rate = -1;
    try {
      timestamp = StrBetween(data, "\"timestamp\":", ",");
      time = Timestamp(timestamp, "HH:mm:ss");
      date = Timestamp(timestamp, "yyyy/MM/dd");
      rate1 = double(StrBetweenAny(data, "USD" + currency1 + "\":", ",}"));
      rate2 = double(StrBetweenAny(data, "USD" + currency2 + "\":", ",}"));
      WriteConsole("Extracted " + rate1 + "," + rate2);
      if (rate1 > 0) {
        rate = Substring(decimal(rate2 / rate1), 0, 10);
      }
    } catch (exception) {
      WriteConsole(exception.Stack);
```

```
      ShowToast("Couldn't get rates. " + exception);
      SetText(labelRateValue, "Error");
      return;
    }
  }

  SetText(labelRateValue, rate);
  SetText(labelDateValue, date);
  SetText(labelTimeValue, time);
}

function init()
{
  currencies = { "EUR",   "USD",   "GBP",   "CHF",   "BTC",  "JPY",
                 "CNY",   "MXN",   "RUB",   "BRL",   "SAR"};
  flags      = { "eu_EU", "en_US", "en_GB", "de_CH", "BTC",  "ja_JP",
                 "zh_CN", "es_MX", "ru_RU", "pt_BR", "ar_SA"};

  AddWidgetData(cbCurrency, currencies);
  AddWidgetImages(cbCurrency, flags);
  SetSize(cbCurrency, 80, 40);
  SetText(cbCurrency, "BTC");

  AddWidgetData(cbCurrency2, currencies);
  AddWidgetImages(cbCurrency2, flags);
  SetSize(cbCurrency2, 80, 40);
  SetText(cbCurrency2, "CHF");

  SetImage(buttonRefresh, "coins");
  AddAction(buttonRefresh, "on_refresh");
  SetFontColor(buttonRefresh, "white");
  SetFontSize(buttonRefresh, 20);

  AddAction(aboutButton, "on_about");
  SetText(aboutButton, "Powered by currencylayer.com");

  labelsFontSize = 17;
  SetFontSize(aboutButton,    labelsFontSize);
  SetFontSize(labelRate,      labelsFontSize);
  SetFontSize(labelDate,      labelsFontSize);
  SetFontSize(labelTime,      labelsFontSize);
  SetFontSize(labelRateValue, labelsFontSize);
  SetFontSize(labelDateValue, labelsFontSize);
  SetFontSize(labelTimeValue, labelsFontSize);
}

function on_portrait(sender, arg)
{
  AddOrSelectTab("Rates", "rates_active.png", "rates_inactive.png");
```


www.dbooks.org

```
  SetBackground("bitcoin_portrait.png");

  locCurrency = GetLocation("ROOT", "LEFT", "ROOT", "TOP", 10, 80);
  AddSfPicker(locCurrency, "cbCurrency", "75", 200, 380);
  SetBackgroundColor(cbCurrency, "white", 0);
  SetFontColor(cbCurrency, "black");

  locCurrency2 = GetLocation("ROOT", "RIGHT", cbCurrency, "CENTER", -20);
  AddSfPicker(locCurrency2, "cbCurrency2", "75", 200, 380);
  SetBackgroundColor(cbCurrency2, "white", 0);
  SetFontColor(cbCurrency2, "black");

  locRateLabel = GetLocation("ROOT", "CENTER", cbCurrency, "BOTTOM",
                             -80, 20);
  AddLabel(locRateLabel, "labelRate", "Rate:", 200, 80);
  SetFontColor(labelRate, "black");

  locRateValue = GetLocation("ROOT", "CENTER", labelRate, "CENTER", 100);
  AddLabel(locRateValue, "labelRateValue", "", 240, 80);
  SetFontColor(labelRateValue, "black");

  locDateLabel = GetLocation("ROOT", "CENTER", labelRate, "BOTTOM", -80);
  AddLabel(locDateLabel, "labelDate", "Date:", 200, 80);
  SetFontColor(labelDate, "black");

  locDateValue = GetLocation("ROOT", "CENTER", labelDate, "CENTER", 100);
  AddLabel(locDateValue, "labelDateValue", "", 240, 80);
  SetFontColor(labelDateValue, "black");

  locTimeLabel = GetLocation("ROOT", "CENTER", labelDate, "BOTTOM", -80);
  AddLabel(locTimeLabel, "labelTime", "Time:", 200, 80);
  SetFontColor(labelTime, "black");

  locTimeValue = GetLocation("ROOT", "CENTER", labelTime, "CENTER", 100);
  AddLabel(locTimeValue, "labelTimeValue", "", 240, 80);
  SetFontColor(labelTimeValue, "black");

  locRefresh = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, -4);
  AddButton(locRefresh, "buttonRefresh", "Convert", 200, 100);

  AddOrSelectTab("Settings", "settings_active.png", "settings_inactive.png"
);
  locAbout = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, -4);
  AddButton(locAbout, "aboutButton", "", 360, 100);
  AddBorder(aboutButton, 1, 5, "black");
  SetFontColor(aboutButton, "black");
}

function on_landscape(sender, arg)
```

```
{
  AddOrSelectTab("Rates", "rates_active.png", "rates_inactive.png");
  SetBackground("bitcoin_landscape.png");

  locCurrency = GetLocation("ROOT", "LEFT", "ROOT", "CENTER", 20);
  AddSfPicker(locCurrency, "cbCurrency", "75", 200, 380);
  SetBackgroundColor(cbCurrency, "black", 0);
  SetFontColor(cbCurrency, "white");

  locCurren2 = GetLocation(cbCurrency, "RIGHT", cbCurrency, "CENTER", 20);
  AddSfPicker(locCurren2, "cbCurrency2", "75", 200, 380);
  SetBackgroundColor(cbCurrency2, "black", 0);
  SetFontColor(cbCurrency2, "white");

  locDateLabel = GetLocation("ROOT", "CENTER", "ROOT", "CENTER", 200);
  AddLabel(locDateLabel, "labelDate", "Date:", 100, 80);
  SetFontColor(labelDate, "white");

  locDateValue = GetLocation(labelDate, "RIGHT", labelDate, "CENTER", 10);
  AddLabel(locDateValue, "labelDateValue", "", 220, 80);
  SetFontColor(labelDateValue, "white");

  locRateLabel = GetLocation("ROOT", "CENTER", labelDate, "TOP", 200);
  AddLabel(locRateLabel, "labelRate", "Rate:", 100, 80);
  SetFontColor(labelRate, "white");

  locRateValue = GetLocation(labelRate, "RIGHT", labelRate, "CENTER", 10);
  AddLabel(locRateValue, "labelRateValue", "", 220, 80);
  SetFontColor(labelRateValue, "white");

  locTimeLabel = GetLocation("ROOT", "CENTER", labelDate, "BOTTOM", 200);
  AddLabel(locTimeLabel, "labelTime", "Time:", 100, 80);
  SetFontColor(labelTime, "white");

  locTimeValue = GetLocation(labelTime, "RIGHT", labelTime, "CENTER", 10);
  AddLabel(locTimeValue, "labelTimeValue", "", 220, 80);
  SetFontColor(labelTimeValue, "white");

  locRefresh = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, -4);
  AddButton(locRefresh, "buttonRefresh", "Convert", 180, 90);

  AddOrSelectTab("Settings", "settings_active.png",
                             "settings_inactive.png");
  locAbout = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, -8);
  AddButton(locAbout, "aboutButton", "", 360, 100);
  AddBorder(aboutButton, 1, 5, "white");
  SetFontColor(aboutButton, "white");
}
```


www.dbooks.org

```
AutoScale();

key = "<Place Your CurrencyLayer.com Key Here>";
apiUrl = "http://apilayer.net/api/live?access_key=" + key + "&currencies=";

RegisterOrientationChange("on_portrait", "on_landscape");
init();

if (Orientation == "Portrait") {
  on_portrait("", "");
} else {
  on_landscape("", "");
}

SelectTab(0);
```

Here is the CSCS function that registers the **on_portrait** function when the device is in portrait mode, and the **on_landscape** function in landscape mode:

```
RegisterOrientationChange("on_portrait", "on_landscape");
```

Note that with CSCS, you can easily configure tabs with different images for cases when the tab is active or inactive. The function to add a new tab is:

```
AddTab(Tab_Name, Picture_Active, Picture_Inactive);
```

Using the following function:

```
AddOrSelectTab(Tab_Name, Picture_Active, Picture_Inactive);
```

When the tab already exists, it won't be added, but just selected to place widgets in it. Figure 5 shows active and inactive tabs for an iOS device.

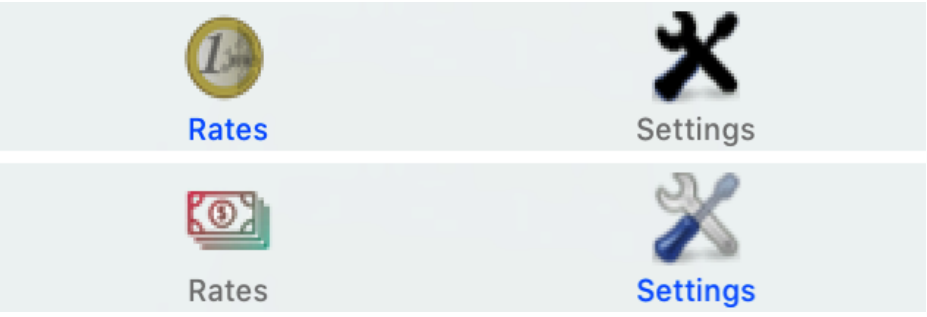


Figure 5: Active and Inactive Tabs on iOS

You probably noticed the funny currency pickers that we used with the **AddSfPicker** function, especially on Android.

These are not the standard pickers, but the ones provided by Syncfusion. I will explain how to add Syncfusion widgets later on, but for now, note that they are treated in CSCS code as any other widgets.

Summary

In this chapter, we saw how to place widgets on the screen in a platform-independent manner. The key concept is a location, where we place a widget. The same location can be reused for placing different widgets. We also saw how this method of widget placement can be used for the device orientation changes.

So far we have seen mostly standard iOS and Android widgets. In the next chapter, we are going to see how to implement and use custom widgets in CSCS.

www.dbooks.org

Chapter 4 Creating Custom Widgets

"It is supposed to be automatic, but actually, you have to push this button."
John Brunner

Adding a custom combo box

Unfortunately, not all mobile operating systems have all common widgets. Among others, iOS is missing a combo box (also known as a drop-down menu), and Android is missing a stepper. In this section we are going to see how to add a combo box to iOS, and you can see how it's done on Android in accompanying source code.

There are two ways of adding a custom widget. The first one is to derive it from the **`UIVariable`** class and then register it separately with the parser. We will see how to do this in the next chapter with Syncfusion® controls. This method is good if you need to quickly detach the widget class from the build.

Another way to create a custom widget is to make it a first-class citizen, registering it in the core **`iOSVariable.GetWidget()`** method, as shown in Code Listing 13. See Code Listing 16 for details.

Code Listing 16: A fragment of the iOSVariable.GetWidget() to add a custom Widget

```
public virtual iOSVariable GetWidget(string widgetType, string widgetName,
                              string initArg, CGRect rect)
{
  //...
  switch (widgetType) {
    //...
    case "Combobox":
      type = UIVariable.UIType.COMBOBOX;
      widgetFunc = new iOSVariable(type, widgetName, widget);
      widgetFunc.CreateCombobox(rect, initArg);
      break;
    // Other widgets ...
  }
}
```

The advantage of registering a custom widget in the **`iOSVariable.GetWidget()`** method is to have a uniform CSCS code for Android and iOS, considering that the standard Android combo box has been added to the **`DroidVariable.GetWidget()`** method.

The iOS custom combo box architecture is the following: it will be implemented as a **`UIButton`** containing some text and an image on its right, showing an arrow. This is before the user selects an item. As soon as the user clicks on the button to select an item, the combo box will be converted to a **`UIPicker`** at the bottom of the screen. This **`UIPicker`** will have a smaller

button on top, and when it's clicked, the **UIPicker** option will be selected. This is very similar to what you get if you see a drop-down menu on a web page with the Safari browser on an iPhone. See the implementation details in Code Listing 17.

Code Listing 17: Implementation of the iOSVariable.CreateCombobox() Method

```
public void CreateCombobox(CGRect rect, string argument)
{
  UIView parent = GetParentView();

  UIView mainView  = AppDelegate.GetCurrentView();
  int mainHeight   = (int)mainView.Frame.Size.Height;
  int mainWidth    = (int)mainView.Frame.Size.Width;

  int pickerHeight = Math.Min(mainHeight / 3, MIN_HEIGHT);
  int pickerWidth  = Math.Min(mainWidth, MIN_WIDTH);
  int pickerY      = mainHeight - pickerHeight + 20;

  m_picker         = new UIPickerView();
  m_button         = new UIButton();
  m_button2        = new UIButton();

  m_button.Frame   = rect;

  m_picker.Frame   = new CGRect(0, pickerY, pickerWidth, pickerHeight);
  m_button2.Frame  = new CGRect(0, pickerY - 20, pickerWidth, 40);

  string alignment = "", color1 = "", color2 = "", closeLabel = "";
  Utils.Extract(argument, ref alignment, ref color1, ref color2,
                        ref closeLabel);
  m_alignment      = alignment;
  Tuple<UIControlContentHorizontalAlignment, UITextAlignment> al =
                       AlignTitleFunction.GetAlignment(alignment);

  m_viewY = new UIView();
  m_viewY.Frame = new CGRect(0, 0, mainWidth, mainHeight);

  TypePickerViewModel model = new TypePickerViewModel(
                                  AppDelegate.GetCurrentController());
  m_picker.ShowSelectionIndicator = true;
  m_picker.Hidden = true;
  m_picker.Model = model;

  if (!string.IsNullOrEmpty(color1)) {
    m_viewY.BackgroundColor = UtilsiOS.String2Color(color1);
    if (string.IsNullOrEmpty(color2)) {
      color2 = color1;
    }
    m_picker.BackgroundColor = UtilsiOS.String2Color(color2);
  }
```


www.dbooks.org

```
m_button.BackgroundColor = UIColor.Clear;
m_button.SetTitleColor(UIColor.Black, UIControlState.Normal);
m_button.Hidden = false;
m_button.Layer.BorderWidth = 1;
m_button.Layer.CornerRadius = 4;
m_button.Layer.BorderColor = UIColor.LightGray.CGColor;
UIImage img = UtilsiOS.CreateComboboxImage(rect);
m_button.SetBackgroundImage(img, UIControlState.Normal);
m_button.ImageView.ClipsToBounds = true;
m_button.ContentMode = UIViewContentMode.Right;
m_button.HorizontalAlignment = al.Item1;
m_button.TouchUpInside += (sender, e) => {
  ResetCombos();
  m_button2.Hidden = false;
  m_picker.Hidden  = false;
  model = m_picker.Model as TypePickerViewModel;

  string text = GetText();
  int row = model.StringToRow(text);
  model.Selected(m_picker, (int)row, 0);
  mainView.BecomeFirstResponder();
  mainView.AddSubview(m_viewY);
};

if (string.IsNullOrEmpty(closeLabel)) {
  closeLabel = "X";
}
m_button2.SetTitle(closeLabel + "\t", UIControlState.Normal);
m_button2.HorizontalAlignment =
    UIControlContentHorizontalAlignment.Right;
m_button2.BackgroundColor = UIColor.FromRGB(100, 100, 100);
m_button2.SetTitleColor(UIColor.White, UIControlState.Normal);
m_button2.Hidden = true;
m_button2.TouchUpInside += (sender, e) => {
  m_button2.Hidden = true;
  m_picker.Hidden = true;
  string text = model.SelectedText;
  SetText(text, alignment, true /* triggered */);
  ActionDelegate?.Invoke(WidgetName, text);

  m_viewY.RemoveFromSuperview();
  mainView.BecomeFirstResponder();
};

mainView.AddSubview(m_button);
m_viewY.AddSubview(m_picker);
m_viewY.AddSubview(m_button2);
```

```
  m_viewX = m_button;
  m_viewX.Tag = ++m_currentTag;
}
```

An example of using the combo box in the CSCS code is shown in Code Listing 18. It will create a combo box widget on the top of the screen. The **"center:red:clear"** initialization parameter means that the text in the combo box is centered, the combo box (**UIPicker**) has a red background color, and the **UIButton** on the top of the screen has a clear (transparent) background.

Code Listing 18: Adding a Combo Box in the CSCS Code

```
locComboWidgets = GetLocation("ROOT", "CENTER", "ROOT", "TOP", 10, 0);
AddCombobox(locComboWidgets, "comboWidgets", "center:red:clear", 360, 60);

sfWidgets = {"CircularGauge", "DigitalGauge", "QRBarcode", "Code39Barcode",
    "BusyIndicator","SplineGraph", "DoughnutGraph", "SemiDoughnutGraph",
    "DataGrid", "Picker", "Excel", "Pdf", "Word"};
AddWidgetData(comboWidgets, sfWidgets, "", "center");
```

The result of the code in Code Listing 18 is shown in Figure 6.

Note that we don't have to use the **"center:red:clear"** initialization; it's only done for brevity. You can achieve the same effect by calling the generic **SetValue()** and **SetBackgroundColor()** functions like this:

```
SetBackgroundColor(comboWidgets, "clear");
SetValue(comboWidgets, "alignment", "center");
SetValue(comboWidgets, "backgroundcolorpicker", "red");
```

You can also change the **X** appearing on the button above the **UIPicker** and all the colors. For example, if you want to have **Done** instead of **X** appear in the yellow color on the green background, use the following CSCS code:

```
SetValue(comboWidgets, "backgroundcolorbutton2", "green");
SetValue(comboWidgets, "fontcolor2", "yellow");
SetValue(comboWidgets, "text2", "Done");
```


www.dbooks.org

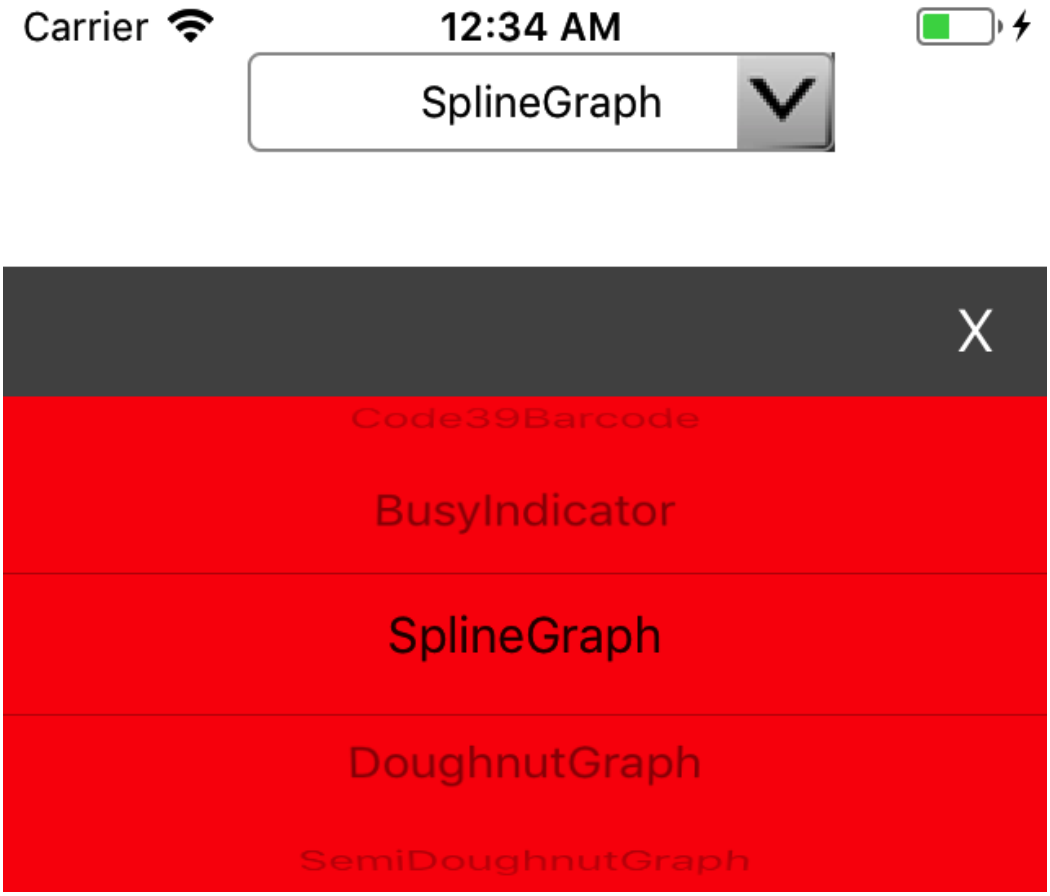


Figure 6: A Custom Combo Box for iOS

To register our combo box widget with the parser, the usual statement is used (the same statement is used for both the standard combo box on Android, and the custom one on iOS):

```
ParserFunction.RegisterFunction("AddCombobox",
                                new AddWidgetFunction("Combobox"));
```

Autocomplete with a trie data structure

Autocomplete is a feature, suggesting the whole word or phrase while the user is typing. One of the algorithms for auto-completion involves the trie data structure (the name comes from "re**trie**val"); sometimes it is also called a *prefix tree*. Trie is very efficient when you're searching for strings that start with a given prefix. The time complexity to search or to insert a string into the trie is at most the length of the string, because for each letter in the string, a new tree level is inserted (unless it already exists in the trie).

The trie itself is not a UI element; therefore, we can add it to the CSCS by extending the **`Variable`** class. See Code Listing 19 for details.

Code Listing 19: A Fragment of the Implementation of a Trie

```
public class WordHint
{
  string m_text;

  public int Id { get; }
  public string OriginalText { get; }
  public string Text { get { return m_text; } }

  public WordHint(string word, int id)
  {
    OriginalText = word;
    Id = id;
    m_text = UIUtils.RemovePrefix(OriginalText);
  }
}

public class TrieCell
{
  string m_name;
  WordHint m_wordHint;

  Dictionary<string, TrieCell> m_children =
      new Dictionary<string, TrieCell>();

  public int Level { get; set; }
  public WordHint WordHint { get { return m_wordHint; } }
  public Dictionary<string, TrieCell> Children {get { return m_children; }}

  public TrieCell(string name = "", WordHint wordHint = null,
                  int level = 0)
  {
    if (wordHint != null && wordHint.Text == name) {
      m_wordHint = wordHint;
    }

    m_name = name;
    Level = level;
  }

  public bool AddChild(WordHint wordHint)
  {
    if (!string.IsNullOrEmpty(m_name) && !wordHint.Text.StartsWith(
                      m_name, StringComparison.OrdinalIgnoreCase)) {
      return false;
    }

    int newLevel = Level + 1;
```


www.dbooks.org

```
    bool lastChild = newLevel >= wordHint.Text.Length;

    string newName = lastChild ? wordHint.Text :
                                 wordHint.Text.Substring(0, newLevel);
    TrieCell oldChild = null;
    if (m_children.TryGetValue(newName, out oldChild)) {
      return oldChild.AddChild(wordHint);
    }

    TrieCell newChild = new TrieCell(newName, wordHint, newLevel);
    m_children[newName] = newChild;

    if (newLevel < wordHint.Text.Length) {
      // if there are still chars left, add a grandchild recursively.
      newChild.AddChild(wordHint);
    }

    return true;
  }
}

public class Trie : Variable
{
  TrieCell m_root;

  public Trie(List<string> words)
  {
    m_root = new TrieCell();

    int index = 0;
    foreach (string word in words) {
      AddWord(word, index++);
    }
  }

  void AddWord(string word, int index)
  {
    WordHint hint = new WordHint(word, index);
    m_root.AddChild(hint);

    string text = hint.Text;
    int space = text.IndexOf(' ');
    while (space > 0) {
      string candidate = text.Substring(space + 1);
      if (!string.IsNullOrWhiteSpace(candidate)) {
        hint = new WordHint(candidate, index);
        m_root.AddChild(hint);
```

```
      }
      if (text.Length < space + 1) {
        break;
      }
      space = text.IndexOf(' ', space + 1);
    }
  }

  public void Search(string text, int max, List<WordHint> results)
  {
    text = UIUtils.RemovePrefix(text);
    TrieCell current = m_root;

    for (int level = 1; level <= text.Length && current != null; level++) {
      string substr = text.Substring(0, level);
      if (!current.Children.TryGetValue(substr, out current)) {
        current = null;
      }
    }

    if (current == null) {
      return; // passed text doesn't exist
    }
    AddAll(current, max, results);
  }

  void AddAll(TrieCell cell, int max, List<WordHint> results)
  {
    if (cell.WordHint != null && !cell.WordHint.Exists(results)) {
      results.Add(cell.WordHint);
    }
    if (results.Count >= max) {
      return;
    }

    foreach (var entry in cell.Children) {
      TrieCell child = entry.Value;
      AddAll(child, max, results);

      if (results.Count >= max) {
        return;
      }
    }
  }
}
```

To register the trie with the parser, we use the following statements:

www.dbooks.org

```
ParserFunction.RegisterFunction("GetTrie",    new CreateTrieFunction());
ParserFunction.RegisterFunction("SearchTrie", new SearchTrieFunction());
```

The **CreateTrieFunction()** function is trivial since it just initializes the trie. Check out the implementation of the SearchTrieFunction class in Code Listing 20.

Code Listing 20: The Implementation of the SearchTrieFunction Class

```
public class SearchTrieFunction : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 2, m_name);

    Trie trie = Utils.GetSafeVariable(args, 0, null) as Trie;
    Utils.CheckNotNull(trie, m_name);

    string text = args[1].AsString();
    int max = Utils.GetSafeInt(args, 2, 10);

    List<WordHint> words = new List<WordHint>();
    trie.Search(text, max, words);

    List<Variable> results = new List<Variable>(words.Count);
    foreach (WordHint word in words) {
      results.Add(new Variable(word.Id));
    }

    return new Variable(results);
  }
}
```

Basically, **SearchTrieFunction** creates and returns a list of matching words. The list consists of word IDs, which must be then associated with the words. Code Listing 21 shows how.

Code Listing 21: The Implementation of the Autocomplete in CSCS

```
AutoScale(1.0);
findMaxWords = 10;
language1 = "en-US";
language2 = "es-MX";
words[language1] = {"bat", "bicycle", "big", "bill", "bone" };
words[language2] = {"barba", "barco", "bicicleta", "billete", "bonito"};
wordsFound   = {};
countryPics  = {};

locTextEdit = GetLocation("ROOT", "CENTER", "ROOT", "TOP", 0, 10);
AddTextEdit(locTextEdit, "textEdit", "Type word dog", 480, 60);
AddAction(textEdit,        "find_text");
```

```
locListView = GetLocation("ROOT", "CENTER", textEdit, "BOTTOM", 0, 4);
AddListView(locListView, "listView", "", 480, 720);
SetVisible(listView, false);

function find_text(sender, arg)
{
  SetVisible(listView, false);
  text = GetText(textEdit);
  if (size(text) == 0) {
    return;
  }

  wordsFound  = {};
  countryPics = {};
  total = search_voice(0, text, language1);
  if (total < findMaxWords) {
    total = search_voice(total, text, language2);
  }
  if (total == 0) {
    return;
  }

  SetVisible(listView, true);
  AddWidgetData(listView, wordsFound);
  AddWidgetImages(listView, countryPics);
  if (firstSearch && total > 0) {
    AddAction(listView, "list_chosen");
    firstSearch = 0;
  }
}

function search_voice(total, text, language)
{
  searchTrie = GetTrie(language, words[language]);
  results = SearchTrie(searchTrie, text, findMaxWords - total);
  for (id : results) {
    wordsFound[total]  = words[language][id];
    countryPics[total] = language;
    total++;
  }
  return total;
}
```

Note that the curly braces denote either a map or a list in CSCS, unlike Python. The results of running Code Listing 21 after the user has typed **bi**, are shown in Figure 7.

www.dbooks.org



Figure 7: Running Autocomplete on iOS (left) and Android (right)

There are just 10 words used in the dictionary. They were added in these statements:

```
words[language1] = {"bat", "bicycle", "big", "bill", "bone" };
words[language2] = {"barba", "barco", "bicicleta", "billete", "bonito"};
```

How do you add more words that you read from a file? As an example, I added a **dictionary.txt** file as an asset to the sample project in the accompanying source code. It contains more than 1,500 words in 10 languages.

The words are tab separated. American English is in the second column, and Mexican Spanish is in the seventh.

Code Listing 22 shows how to read data from a file and assign it to CSCS data structures. Note that the **dictionary.txt** file contains some language-specific information in the first four rows—that's why we skip them.

Code Listing 22: Reading and Processing Data From a File in CSCS

```
WriteConsole(Now(), " Starting reading file.");

lines = ReadFile("dictionary.txt");
words[language1] = {};
words[language2] = {};
lineNr = 0;
for (line : lines) {
  if (++lineNr < 5) {
    continue;
  }
  tokens = tokenize(line, "\t");
  if (size(tokens) < 7) {
    continue;
  }
  add(words[language1], tokens[1]);
  add(words[language2], tokens[6]);
```

```
}
WriteConsole(Now(), " Added ", size(words[language1]), " words");
```

Summary

In this chapter, we saw how to implement custom widgets in CSCS, taking as a GUI example a combo box, and as a non-GUI example, an autocomplete using a trie data structure.

In the next chapter, we are going to look into adding custom widgets that are developed by someone else and imported into the project.

www.dbooks.org

Chapter 5 Adding Syncfusion Controls

"Good artists copy, great artists steal."
Pablo Picasso

In this chapter, we are going to see how to add controls not developed by us, but by someone else. As an example, I decided to use Syncfusion controls. Why?

- They look really cool and are very similar on iOS and Android.
- They are easy to use and configure.
- They are easy to include in the CSCS Framework, making them look as a part of CSCS.
- There is a Community License that allows free use of the Syncfusion framework, provided your revenue is below $1 million and there are up to five developers in your company that will be working with this framework.
- Even the Community License includes email support.
- I enjoy using free stuff. Don't you? (If your revenue is more than $1 million or you have more than five developers, you will need to buy a license here).

It goes without saying that this book is published by Syncfusion not because I advertise their controls: they explicitly told me that I should use any product that fits this book the best.

Adding Syncfusion to the CSCS project

The easiest way to figure out how the Syncfusion controls work is to download the Syncfusion sample project. You can add the Syncfusion controls to your project in two different ways:

- As DLLs from the sample project to the **References** section of your project.
- As NuGet Packages via **Your Project** > **Packages** > **Add Packages**

Figure 8 shows how your Visual Studio project will look after adding the Syncfusion NuGet packages.

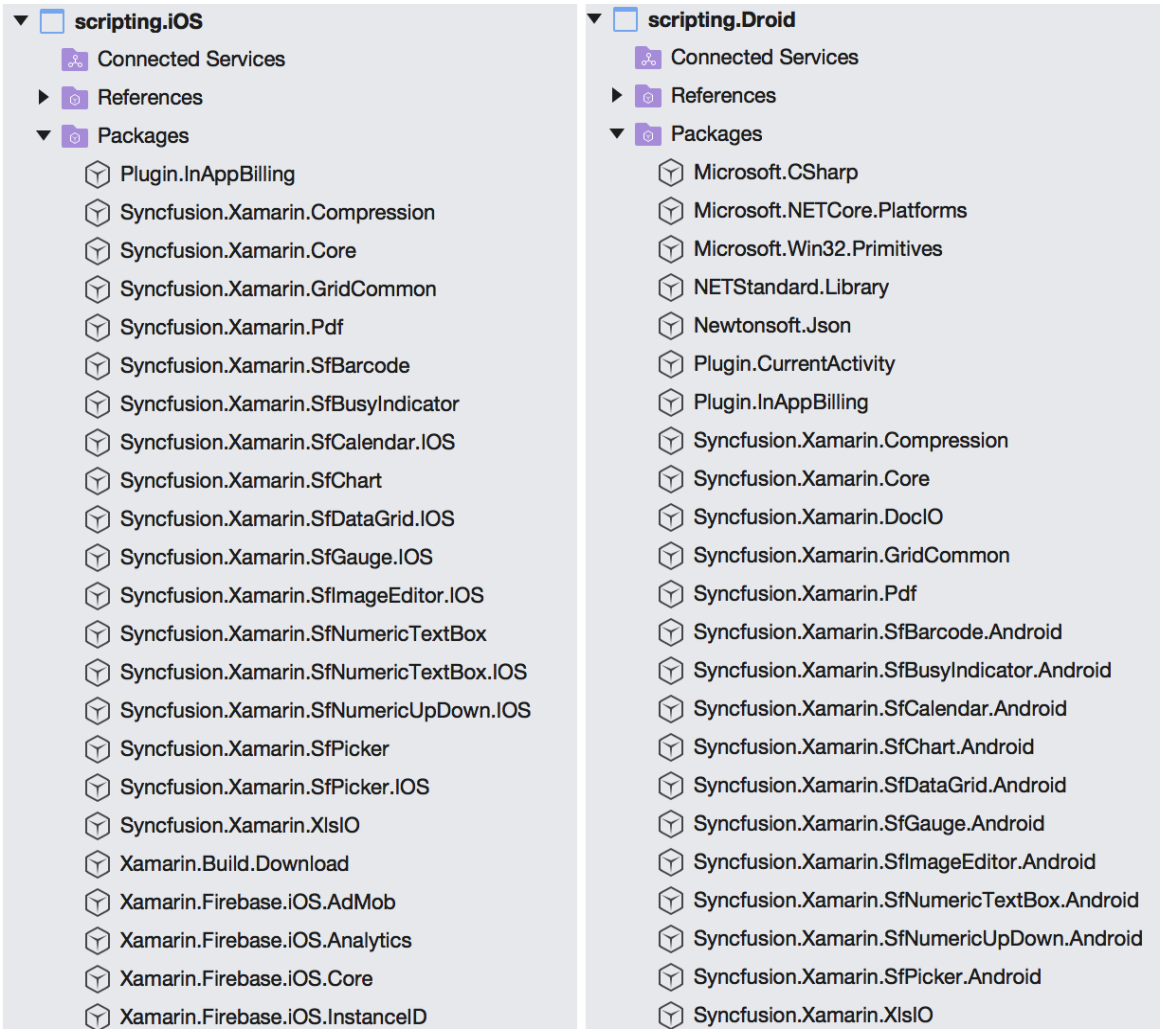


Figure 8: Syncfusion iOS and Android NuGet Packages (on macOS)

For using the controls, I created the **`SfWidget`** class in both Xamarin.iOS and Xamarin.Android. I didn't want to put the common class into the shared project since there are too many differences between the iOS and Android Syncfusion implementations (for example, **`SFBarcode`** on iOS versus **`SfBarcode`** on Android, and so on).

So I added all of the Syncfusion widgets to the **`SfWidget`** class, deriving from the **`iOSVariable`** class on iOS, and from the **`DroidVariable`** class on Android. A fragment of the iOS version is shown in Code Listing 23. The Android version looks very similar.

Code Listing 23: A Fragment of the SfWidget Class on iOS

```
public class SfWidget : iOSVariable
{
  SyncFusionType  m_type;
  SFBarcode       m_barcode;
```


www.dbooks.org

```
SFDigitalGauge  m_digitalGauge;
SFCircularGauge m_circularGauge;
SFCircularScale m_circularScale;
SFNeedlePointer m_needlePointer;
SFRangePointer  m_rangePointer1, m_rangePointer2;
SFNumericUpDown m_stepper;
SFBusyIndicator m_busyIndicator;
SfPicker        m_picker;
SFChart         m_chart;

SfDataGrid      m_grid;
int             m_rowIndex;
DataModel       m_model;
UIView          m_detailView;

string          m_data;
CGRect          m_rect;

List<string>    m_strings;
List<UIImage>   m_pics;

UITextAlignment m_alignment = UITextAlignment.Left;
UIColor m_bgColor           = UIColor.Clear;
UIColor m_fontColor         = UIColor.Black;

static bool     m_init;

public enum SyncFusionType {
  NONE, QR_BARCODE, CODE39_BARCODE, CIRCULAR_GAUGE, DIGITAL_GAUGE,
  STEPPER, BUSY_INDICATOR,PICKER, SPLINE_GRAPH, DOUGHNUT_GRAPH, DATA_GRID
};

public SyncFusionType SfType { get; set; }
public SfWidget(SyncFusionType type, string name, string text,
                CGRect rect) :
    base(UIType.CUSTOM, name)
{
  m_type = type;
  m_data = text;
  m_rect = rect;
  SfType = type;
  switch (type) {
    case SyncFusionType.BUSY_INDICATOR:
      CreateBusyIndicator();
      break;
    case SyncFusionType.QR_BARCODE:
      CreateQRBarcode();
      break;
    case SyncFusionType.CODE39_BARCODE:
```

```
        CreateCode39Barcode();
        break;
      case SyncFusionType.PICKER:
        CreatePicker();
        break;
      case SyncFusionType.STEPPER:
        CreateStepper();
        break;
      case SyncFusionType.CIRCULAR_GAUGE:
        CreateCircularGauge();
        break;
      case SyncFusionType.DIGITAL_GAUGE:
        CreateDigitalGauge();
        break;
      case SyncFusionType.DATA_GRID:
        CreateDataGrid();
        break;
      case SyncFusionType.DOUGHNUT_GRAPH:
      case SyncFusionType.SPLINE_GRAPH:
        CreateGraph();
        break;
    }
    ViewX.Tag = ++m_currentTag;
  }
  public override Variable Clone()
  {
    SfWidget newVar = (SfWidget)this.MemberwiseClone();
    return newVar;
  }

  public override iOSVariable GetWidget(string widgetType,
           string widgetName, string initArg, CGRect rect)
  {
    switch (widgetType) {
      case "SfQRBarcode":
        return new SfWidget(SfWidget.SyncFusionType.QR_BARCODE,
                            widgetName, initArg, rect);
      case "SfCode39Barcode":
        return new SfWidget(SfWidget.SyncFusionType.CODE39_BARCODE,
                            widgetName, initArg, rect);
      case "SfCircularGauge":
        return new SfWidget(SfWidget.SyncFusionType.CIRCULAR_GAUGE,
                            widgetName, initArg, rect);
      case "SfDigitalGauge":
        return new SfWidget(SfWidget.SyncFusionType.DIGITAL_GAUGE,
                            widgetName, initArg, rect);
      case "SfStepper":
        return new SfWidget(SfWidget.SyncFusionType.STEPPER,
```


www.dbooks.org

```
                            widgetName, initArg, rect);
      case "SfBusyIndicator":
        return new SfWidget(SfWidget.SyncFusionType.BUSY_INDICATOR,
                            widgetName, initArg, rect);
      case "SfDataGrid":
        return new SfWidget(SfWidget.SyncFusionType.DATA_GRID,
                            widgetName, initArg, rect);
      case "SfDoughnutGraph":
        return new SfWidget(SfWidget.SyncFusionType.DOUGHNUT_GRAPH,
                            widgetName, initArg, rect);
      case "SfSplineGraph":
        return new SfWidget(SfWidget.SyncFusionType.SPLINE_GRAPH,
                            widgetName, initArg, rect);
      case "SfPicker":
        return new SfWidget(SfWidget.SyncFusionType.PICKER,
                            widgetName, initArg, rect);
    }
    return null;
  }
```

I registered all the Syncfusion controls and related functions within the **`CustomInit`** class; see Code Listing 5. I informed the parser that it needs to look for a particular type of a widget in this statement in **`CustomInit`**:

```
UIVariable.WidgetTypes.Add(new SfWidget());
```

Next, let's look at some of the Syncfusion controls.

Stepper or Numeric Up Down

Figure 9 contains a screenshot of a stepper (or a *Numeric Up Down,* in Syncfusion terms). There are three kinds of steppers, depending on the stepper configuration: the buttons could be either on the left, right, or on both sides of the widget. They look somewhat different when the buttons are on both sides of the widget—in iOS, **+** precedes **-**. I guess this is more the iOS style.

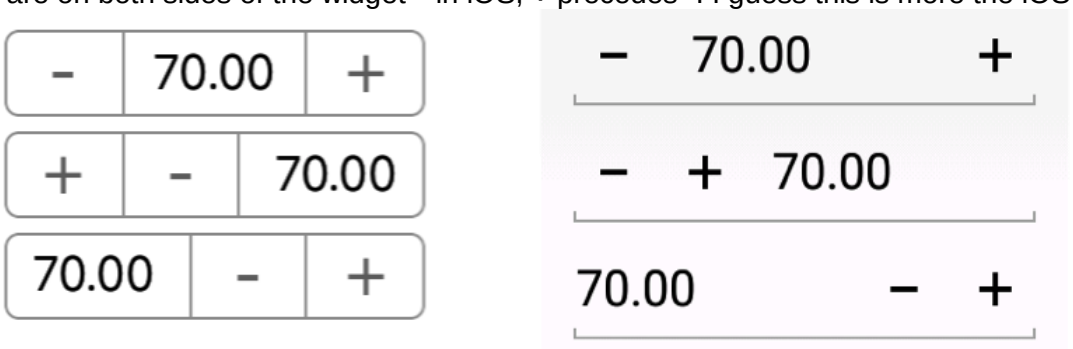


Figure 9: Syncfusion Stepper on iOS and Android

The controls in Figure 9 were created using the following CSCS script:

```
locStepper1 = GetLocation("ROOT", "CENTER", "ROOT", "CENTER");
AddSfStepper(locStepper1, "stepper1", "70:10:120:5", 300, 60);
SetValue(stepper1, "buttons", "both");

locStepper2 = GetLocation("ROOT", "CENTER", stepper1, "BOTTOM", 0, 10);
AddSfStepper(locStepper2, "stepper2", "70:10:120:5", 300, 60);
SetValue(stepper2, "buttons", "left");

locStepper3 = GetLocation("ROOT", "CENTER", stepper2, "BOTTOM", 0, 10);
AddSfStepper(locStepper3, "stepper3", "70:10:120:5", 300, 60);
SetValue(stepper3, "buttons", "right");
```

The **"70:10:120:5"** initialization means that 70 is the default value, the minimum is 10, the maximum is 120, and the step is 5. This is just a shortcut; you can do the same with these separate CSCS statements:

```
SetValue(stepper, "value",  70);  or SetValue(stepper, 70);
SetValue(stepper, "min",    10);
SetValue(stepper, "max",    120);
SetValue(stepper, "step",   5);
```

To get the current value of the stepper, use the following statement:

```
stepperValue = GetValue(stepper);
```

You can also subscribe to an event of the user by pressing a plus or minus button:

```
AddAction(stepper, "stepper_clicked");
function stepper_clicked(sender, arg)
{
// arg contains the current stepper value.
}
```

Now let's see the stepper's implementation. As follows from Code Listing 23, the **CreateStepper()** method is called in case of a stepper. It is shown in Code Listing 24.

Code Listing 24: The Implementation of the CreateStepper Method

```
void CreateStepper()
{
  m_stepper = new SFNumericUpDown();
  m_stepper.Frame = m_rect;

  double minValue = 0, maxValue = 1, currValue = 0, step = 1.0;
  Utils.Extract(m_data, ref currValue, ref minValue,
```


www.dbooks.org

```
                        ref maxValue, ref step);
  m_stepper.AllowNull = true;
  m_stepper.PercentDisplayMode = SFNumericUpDownPercentDisplayMode.Value;
  m_stepper.ValueChangeMode = SFNumericUpDownValueChangeMode.OnLostFocus;
  m_stepper.Value = (nfloat)currValue;
  m_stepper.StepValue = (nfloat)step;
  m_stepper.Minimum = (nfloat)minValue;
  m_stepper.Maximum = (nfloat)maxValue;
  m_stepper.Culture = NSLocale.CurrentLocale;

  m_stepper.ValueChanged += (sender, e) =>  {
    ActionDelegate?.Invoke(WidgetName, e.Value.ToString());
  };

  ViewX = m_stepper;
}
```

That's it—as you can see, the C# implementation is pretty compact, since the rest was done by the good people at Syncfusion.

To be able to use the generic **`SetValue()`** CSCS function, the **`iOSVariable.SetValue()`** function for iOS, and the **`DroidVariable.SetValue()`** function for Android, must be overridden. A fragment of the **`SfWidgetSetValue()`** function for the stepper is shown in Code Listing 25.

Code Listing 25: A Fragment of the SfWidget.SetValue Method

```
public override bool SetValue(string value1, string value2 = "")
{
  double valueNum = Utils.ConvertToDouble(value2);
  if (m_stepper != null) {
    switch (value1) {
      case "min":
        m_stepper.Minimum   = (nfloat)valueNum;
        break;
      case "max":
        m_stepper.Maximum   = (nfloat)valueNum;
        break;
      case "step":
        m_stepper.StepValue = (nfloat)valueNum;
        break;
      case "value":
        m_stepper.Value     = (nfloat)valueNum;
        break;
      case "buttons":
        switch(value2) {
          case "left": m_stepper.SpinButtonAlignment =
```

```
                        SFNumericUpDownSpinButtonAlignment.Left;
            break;
          case "right":
            m_stepper.SpinButtonAlignment =
                        SFNumericUpDownSpinButtonAlignment.Right;
            break;
          default: m_stepper.SpinButtonAlignment =
                        SFNumericUpDownSpinButtonAlignment.Both;
            break;
        }
        break;
      }
    }
  }
 // Other Syncfusion Widgets.
}
```

You can also set up custom icons instead of the **+** and **-** buttons, but this is out of our scope.

QR barcode

A QR (“Quick Response”) barcode is a form of a two-dimensional machine-readable matrix. It is often used to code website addresses. There is also a Syncfusion control that creates these codes. Check out an example of applying the QR code to the Google website in Figure 10.



Figure 10: Syncfusion QR Barcode for www.google.com

The QR barcode in Figure 10 was created using the following CSCS code:

www.dbooks.org

```
locSFWidget = GetLocation("ROOT", "CENTER", "ROOT", "CENTER");
AddSfQRBarcode(locSFWidget,  "QRBarcode", "http://www.google.com", 450, 450);
```

You can also set the QR code in a separate CSCS statement, for example:

```
SetText(QRBarcode, "http://www.iLanguage.ch");
```

The C# implementation is very compact as well; see Code Listing 26.

Code Listing 26: The Implementation of the CreateQRBarcode Method

```
void CreateQRBarcode()
{
  m_barcode = new SFBarcode();
  m_barcode.Frame = m_rect;

  m_barcode.BackgroundColor = UIColor.Clear;
  m_barcode.Text = (NSString)m_data;
  m_barcode.Symbology = SFBarcodeSymbolType.SFBarcodeSymbolTypeQRCode;
  SFQRBarcodeSettings settings = new SFQRBarcodeSettings();
  settings.XDimension = 5;
  m_barcode.SymbologySettings = settings;

  ViewX = m_barcode;
}
```

Barcode 39

Code 39 is a one-dimensional barcode used to code uppercase words, numbers, and some special characters. It can be used to code prices for subsequent scanning. Figure 11 shows an example of coding a $9.99 price.



Figure 11: Syncfusion Barcode 39

The barcode in Figure 11 was created using the following CSCS code:

```
AddSfCode39Barcode(locSFWidget, "Code39", "$9.99", 600, 400);
```

Note that we can reuse the same location for **`locSFWidget`** that we used before, placing into it a different widget and hiding the widget that was there before. The C# code is very similar to the Code Listing 26, so we will skip it.

Picker

We already saw the Syncfusion Picker in action in Chapter 3. The cool part about this control is that it's very easy to configure it to have multiple columns. However, here we are going to look at a harder part to implement—a picker with an image and a text together. See how it looks on iOS and Android in Figure 12.



Figure 12: Syncfusion Picker Control on iOS and on Android

The CSCS code to produce the picker in Figure 12 is the following:

```
AddSfPicker(locSFWidget, "cbCurrency", "75", 200, 380);
currencies = {"EUR",   "USD",   "GBP",   "CHF",   "BTC", "JPY",  "CNY",
              "MXN",   "RUB",   "BRL",   "SAR"};
flags      = {"eu_EU", "en_US", "en_GB", "de_CH", "BTC", "ja_JP", "zh_CN",
              "es_MX", "ru_RU", "pt_BR", "ar_SA"};
AddWidgetData(cbCurrency, currencies);
AddWidgetImages(cbCurrency, flags);
SetSize(cbCurrency, 80, 40);
SetText(cbCurrency, "BTC");
```

We are going to see how to implement it on Android because it requires more effort—we have to manually create a view for each picker entry.

We also have to manually create different views for selected and unselected entries. On top of that, we have to reload all views on each selection. See Code Listing 27 for details.

www.dbooks.org

Code Listing 27: A Fragment of the CreatePicker Method for Android

```
void CreatePicker()
{
  m_picker = new SfPicker(m_context);

  string str1 = "", str2 = "", str3 = "", str4 = "";
  Utils.Extract(m_data, ref str1, ref str2, ref str3, ref str4);
  if (!string.IsNullOrEmpty(str1)) {
    double rowHeight = Utils.ConvertToDouble(str1);
    if (rowHeight > 0) {
      rowHeight = AutoScaleFunction.TransformSize((int)rowHeight,
                                  UtilsDroid.GetScreenSize().Width);
      m_picker.ItemHeight = rowHeight;
    }
  }
  m_picker.ShowHeader = !string.IsNullOrEmpty(str2);
  if (m_picker.ShowHeader) {
    m_picker.HeaderHeight = 60;
    m_picker.HeaderText = str2;
    m_picker.HeaderBackgroundColor = Color.Transparent;
    m_picker.HeaderTextColor = Color.Black;
  }
  m_picker.ShowColumnHeader = !string.IsNullOrEmpty(str3);
  if (m_picker.ShowColumnHeader) {
    m_picker.ColumnHeaderHeight = 60;
    m_picker.ColumnHeaderText = str3;
    m_picker.ColumnHeaderBackgroundColor = Color.Transparent;
    m_picker.ColumnHeaderTextColor = Color.Black;
  }

  m_picker.ShowFooter    = false;
  m_picker.SelectedIndex = 0;
  m_picker.PickerMode    = PickerMode.Default;
  m_picker.BorderColor   = Color.Black;
  m_picker.PickerWidth   = m_width;
  m_picker.PickerHeight  = m_height;

  m_picker.OnSelectionChanged += (sender, e) => {
    ActionDelegate?.Invoke(WidgetName, e.NewValue.ToString());
  };

  var linePic = UtilsDroid.String2Pic("grayline");
  m_picker.OnPickerItemLoaded += (sender, e) => {
    if (m_pics == null) {
      return;
    }
    m_picker.HeaderBackgroundColor = Color.Transparent;
    m_picker.HeaderTextColor = Color.Black;
```

```
    m_picker.UnSelectedItemTextColor = Color.LightGray;
    m_picker.SelectedItemTextcolor = Color.Black;

    int row = e.Row;
    bool isSelected = row == (int)m_picker.SelectedIndex;
    LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams(
        ViewGroup.LayoutParams.WrapContent,
        ViewGroup.LayoutParams.WrapContent
    );
    layoutParams.Height  = (int)m_picker.ItemHeight;
    layoutParams.Width   = (int)m_picker.Width;
    LinearLayout rowView = new LinearLayout(m_context) {
                           Orientation = Orientation.Vertical };

    LinearLayout rowData = new LinearLayout(m_context);
    rowData.Orientation = Orientation.Horizontal;

    if (m_pics.Count > row) {
      ImageView rowImage = new ImageView(m_context);
      rowImage.SetImageResource(m_pics[row]);
      if (!isSelected) {
        rowImage.SetColorFilter(Color.Argb(200, 128, 128, 128));
      } else {
        rowImage.SetColorFilter(null);
      }
      rowData.AddView(rowImage, ViewGroup.LayoutParams.WrapContent,
                                ViewGroup.LayoutParams.WrapContent);
    }
    if (m_strings != null && m_strings.Count > row) {
      TextView rowText = new TextView(m_context);
      rowText.Text = " " + m_strings[row];
      rowText.TextAlignment = m_alignment.Item2;
      rowText.SetBackgroundColor(m_bgColor);
      rowText.SetTextColor(isSelected ? m_fontColor : Color.Gray);
      rowText.Gravity = GravityFlags.CenterVertical;
      rowData.AddView(rowText, ViewGroup.LayoutParams.WrapContent,
                               ViewGroup.LayoutParams.WrapContent);
    }

    ImageView line = new ImageView(m_context);
    line.SetImageResource(linePic);
    line.LayoutParameters = new ViewGroup.LayoutParams(
                                ViewGroup.LayoutParams.MatchParent,
                                ViewGroup.LayoutParams.WrapContent);
    rowView.AddView(rowData, ViewGroup.LayoutParams.MatchParent,
                             ViewGroup.LayoutParams.WrapContent);
    rowView.AddView(line,    ViewGroup.LayoutParams.MatchParent,
                             ViewGroup.LayoutParams.WrapContent);
    rowView.LayoutParameters = layoutParams;
```


www.dbooks.org

```
    e.CustomView = rowView;
  };

  ViewX = m_picker;
}
```

Digital Gauge

The Digital Gauge control looks especially cool when showing the current time; see Figure 13.

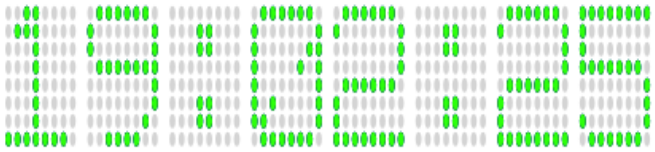


Figure 13: Syncfusion Digital Gauge

The CSCS code producing what you see in Figure 13 is the following:

```
AddSfDigitalGauge(locSFWidget,  "DigitalGauge", "", 500, 200);
SetValue(DigitalGauge, "value", Now("HH:mm:ss"));
SetFontColor(DigitalGauge,  "green");

Schedule(1000, "timer_timeout", "", "timerId", 1);
function timer_timeout(sender, arg)
{
  SetValue(DigitalGauge, "value", Now("HH:mm:ss"));
}
```

Note that the time shown in the Digital Gauge control is being updated every second. The C# wrapper over the Syncfusion Digital Gauge is shown in Code Listing 28.

Code Listing 28: The Implementation of the CreateDigitalGauge Method

```
void CreateDigitalGauge()
{
  m_digitalGauge = new SFDigitalGauge();
  m_digitalGauge.Frame = m_rect;

  m_digitalGauge.CharacterHeight = 36;
  m_digitalGauge.CharacterWidth = 17;
  m_digitalGauge.VerticalPadding = 10;
  m_digitalGauge.SegmentWidth = 3;

  m_digitalGauge.CharacterType = SFDigitalGaugeCharacterType.
```

```
            SFDigitalGaugeCharacterTypeEightCrossEightDotMatrix;
  m_digitalGauge.StrokeType = SFDigitalGaugeStrokeType.
            SFDigitalGaugeStrokeTypeTriangleEdge;
  m_digitalGauge.Value = (NSString)NSDate.Now.ToString();
  m_digitalGauge.DimmedSegmentAlpha = 0.11f;
  m_digitalGauge.BackgroundColor = UIColor.White;
  m_digitalGauge.CharacterColor = UIColor.Blue;
  m_digitalGauge.DimmedSegmentColor = UIColor.LightGray;

  ViewX = m_digitalGauge;
}
```

Circular Gauge

The Circular Gauge control is probably one of the most complex, regarding the configuration. There are many different options for it. An example with two scales is shown in Figure 14.

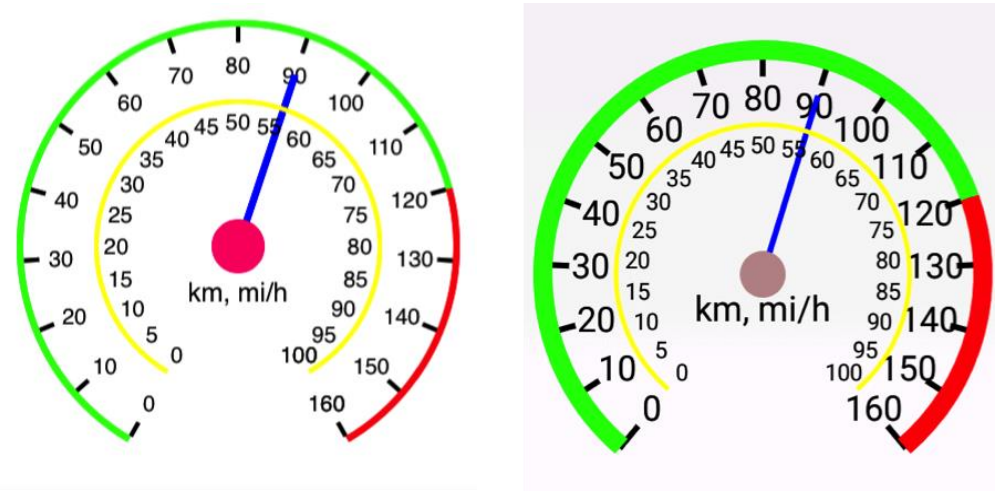


Figure 14: Syncfusion Circular Gauge on iOS and on Android

The example in Figure 14 was created using the CSCS code in Code Listing 29:

Code Listing 29: The CSCS Code to Create a Syncfusion Circular Gauge

```
AddSfCircularGauge(locSFWidget, "CircularGauge", "", 400, 400);
SetValue(CircularGauge, "from",                0);
SetValue(CircularGauge, "to",                  160);
SetValue(CircularGauge, "step",                10);
SetValue(CircularGauge, "value",               90);
SetValue(CircularGauge, "start_angle",         30);
SetValue(CircularGauge, "sweep_angle",         330);
SetValue(CircularGauge, "start_range2",        120);
SetValue(CircularGauge, "radius_knob",         12);
```


www.dbooks.org

```
SetValue(CircularGauge, "color_needle",       "blue");
SetValue(CircularGauge, "color_knob",         "rose");
SetValue(CircularGauge, "color_labels",       "black");
SetValue(CircularGauge, "color_range1",       "green");
SetValue(CircularGauge, "color_range2",       "red");
SetValue(CircularGauge, "color_minorticks",   "gray");
SetValue(CircularGauge, "color_majorticks",   "black");
SetValue(CircularGauge, "scale2_from",        0);
SetValue(CircularGauge, "scale2_to",          100);
SetValue(CircularGauge, "scale2_interval",    5);
SetValue(CircularGauge, "scale2_rimcolor",    "yellow");
SetValue(CircularGauge, "scale2_labelcolor",  "black");

SetText(CircularGauge,  "km, mi/h");
```

Calendar

The Syncfusion Calendar is a nice widget displaying either a month or a year (depending on the configuration). You can get the previous or next years or months by swiping it left and right. You can also easily add appointments to the calendar. Check out Figure 15 for an example.

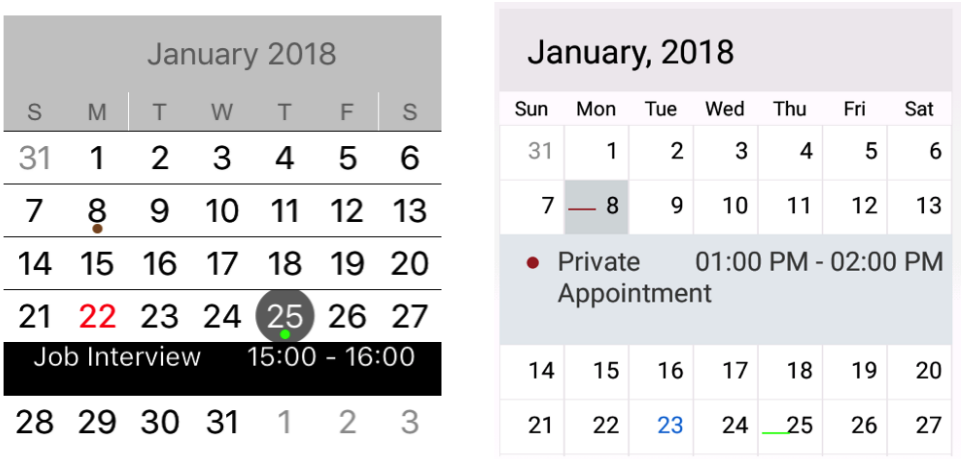


Figure 15: Syncfusion Calendar on iOS and on Android

Here is the CSCS code that created the controls in Figure 15:

```
AddSfCalendar(locSFWidget, "Calendar", "", 450, 450);
AddSfAppointment(Calendar, "Private Appointment",
                 "2018/01/08 13:00", "2018/01/08 14:00", "brown");
AddSfAppointment(Calendar, "Job Interview",
                "2018/01/25 15:00", "2018/01/25 16:00", "green");
```

Image Editor

This is probably one of the coolest Syncfusion widgets. You can actually use it to edit images! See Figure 16 for an example.

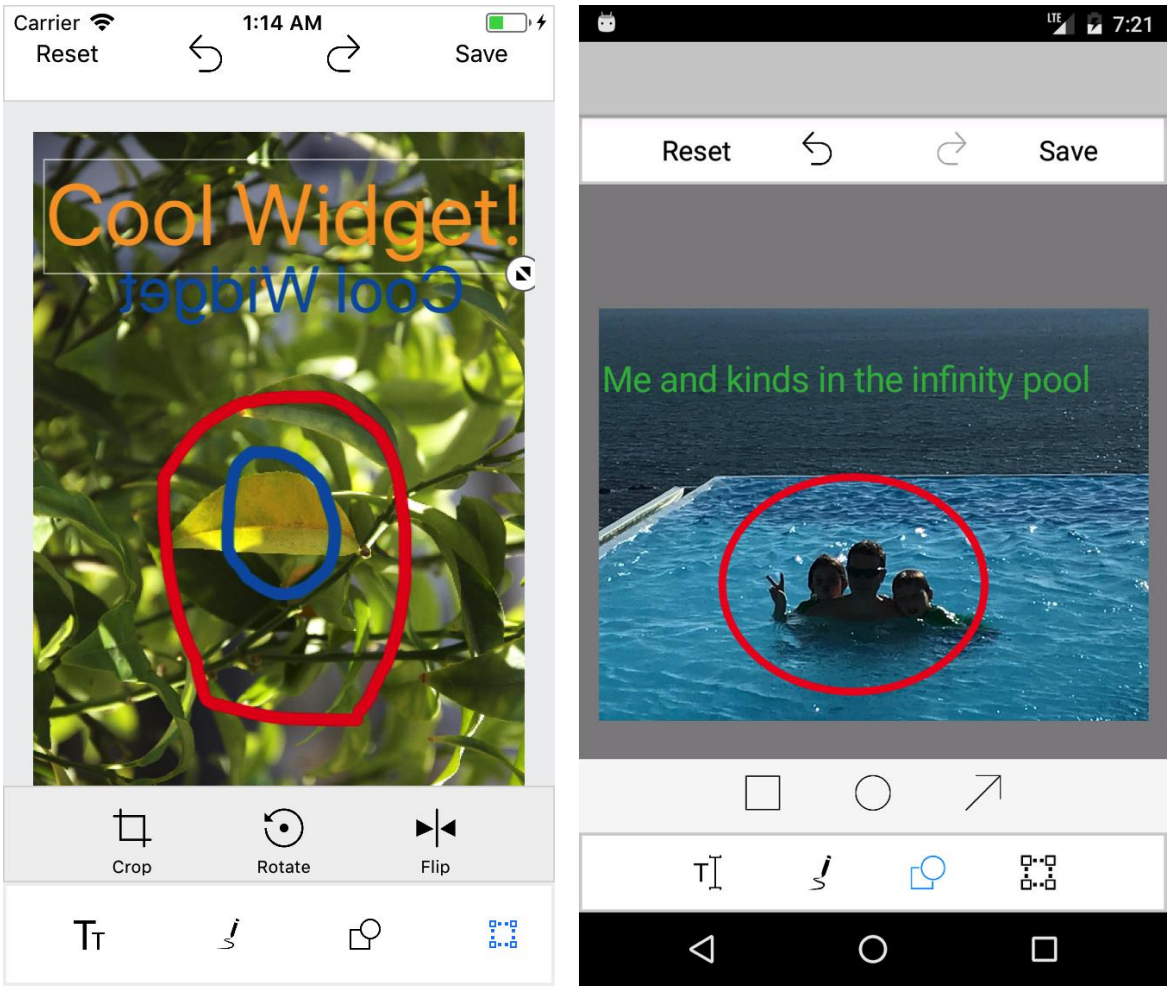


Figure 16: Syncfusion Image Editor on iOS and on Android

The Syncfusion Image Editor can be created from the following CSCS code:

```
AddSfImageEditor(locSFWidget, "ImageEditor", "library",
                 DisplayWidth, DisplayHeight);
```

Yes, just a one-liner! Note that **`DisplayWidth`** and **`DisplayHeight`** are special CSCS variables, indicating that the widget should take over the whole screen.

The **`library`** initialization parameter indicates that the picture will be selected by the user from a device library file. Another option is **`camera`**, meaning that the image will be taken from a device camera.

www.dbooks.org

Busy Indicator

The Busy Indicator control currently contains 15 different animations that can be shown to the user while the app is doing some other work. Figure 17 contains all these 15 animations. It is a static picture, but in reality, all the images are animated.

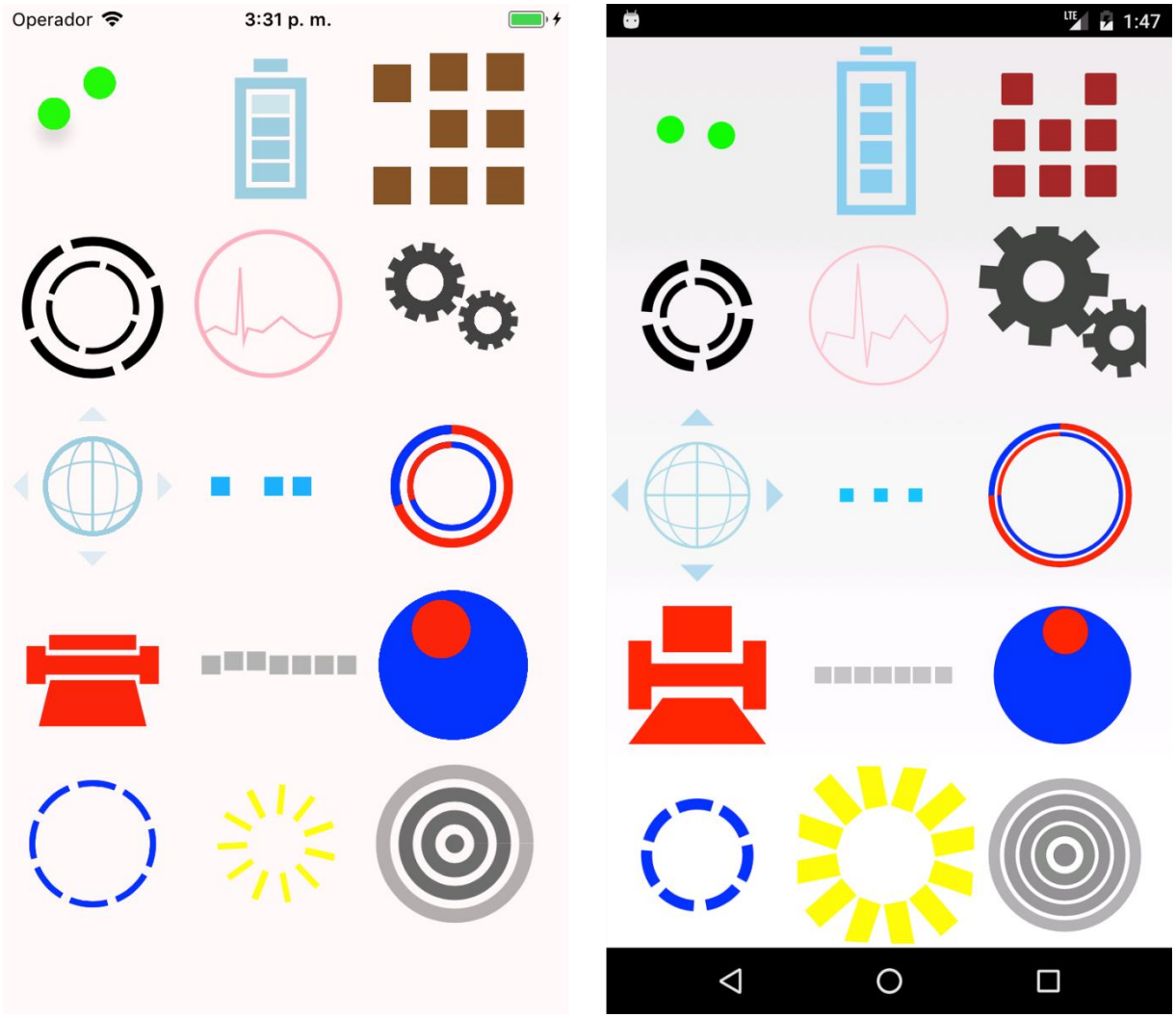


Figure 17: Syncfusion BusyIndicator on iOS and Android

The indicators in Figure 17 were set up in CSCS by providing different parameters during the initialization. Check out the CSCS code in Code Listing 30.

Code Listing 30: The CSCS Code to Create Syncfusion BusyIndicators

```
SetBackgroundColor("snow");
h = 200;
w = 200;
delta = 3;
```

```
locSFWidget1 = GetLocation("ROOT", "LEFT", "ROOT", "TOP", delta, 12);
AddSfBusyIndicator(locSFWidget1, "BusyIndicator1", "ball", h, w);

locSFWidget2 = GetLocation(BusyIndicator1, "RIGHT",
                           BusyIndicator1, "CENTER", delta);
AddSfBusyIndicator(locSFWidget2, "BusyIndicator2", "battery", h, w);

locSFWidget3 = GetLocation(BusyIndicator2, "RIGHT",
                           BusyIndicator2, "CENTER", delta);
AddSfBusyIndicator(locSFWidget3, "BusyIndicator3", "box", h, w);

locSFWidget4 = GetLocation(BusyIndicator1, "CENTER",
                           BusyIndicator1, "BOTTOM",  0, delta);
AddSfBusyIndicator(locSFWidget4, "BusyIndicator4", "doublecircle", h, w);

locSFWidget5 = GetLocation(BusyIndicator2, "CENTER",
                           BusyIndicator2, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget5, "BusyIndicator5", "ecg", h, w);

locSFWidget6 = GetLocation(BusyIndicator3, "CENTER",
                           BusyIndicator3, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget6, "BusyIndicator6", "gear", h, w);

locSFWidget7 = GetLocation(BusyIndicator4, "CENTER",
                           BusyIndicator4, "BOTTOM",  0, delta);
AddSfBusyIndicator(locSFWidget7, "BusyIndicator7", "globe", h, w);

locSFWidget8 = GetLocation(BusyIndicator5, "CENTER",
                           BusyIndicator5, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget8, "BusyIndicator8", "pulsing", h, w);

locSFWidget9 = GetLocation(BusyIndicator6, "CENTER",
                           BusyIndicator6, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget9, "BusyIndicator9", "movie", h, w);

locSFWidget10 = GetLocation(BusyIndicator7, "CENTER",
                            BusyIndicator7, "BOTTOM",  0, delta);
AddSfBusyIndicator(locSFWidget10, "BusyIndicator10", "print", h, w);

locSFWidget11 = GetLocation(BusyIndicator8, "CENTER",
                            BusyIndicator8, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget11, "BusyIndicator11", "rectangle", h, w);

locSFWidget12 = GetLocation(BusyIndicator9, "CENTER",
                            BusyIndicator9, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget12, "BusyIndicator12", "rollingball", h, w);

locSFWidget13 = GetLocation(BusyIndicator10, "CENTER",
```


www.dbooks.org

```
                              BusyIndicator10, "BOTTOM",  0, delta);
AddSfBusyIndicator(locSFWidget13, "BusyIndicator13", "singlecircle", h, w);

locSFWidget14 = GetLocation(BusyIndicator11, "CENTER",
                              BusyIndicator11, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget14, "BusyIndicator14", "slicedcircle", h, w);

locSFWidget15 = GetLocation(BusyIndicator12, "CENTER",
                              BusyIndicator12, "BOTTOM", delta, delta);
AddSfBusyIndicator(locSFWidget15, "BusyIndicator15", "zooming", h, w);

SetValue(BusyIndicator1,  "color", "green");
SetValue(BusyIndicator2,  "color", "sky_blue");
SetValue(BusyIndicator3,  "color", "brown");
SetValue(BusyIndicator4,  "color", "black");
SetValue(BusyIndicator5,  "color", "pink");
SetValue(BusyIndicator6,  "color", "dark_gray");
SetValue(BusyIndicator7,  "color", "light_blue");
SetValue(BusyIndicator8,  "color", "deep_sky_blue");
SetValue(BusyIndicator9,  "color", "blue");
SetValue(BusyIndicator10, "color", "red");
SetValue(BusyIndicator11, "color", "silver");
SetValue(BusyIndicator12, "color", "blue");
SetValue(BusyIndicator13, "color", "blue");
SetValue(BusyIndicator14, "color", "yellow");
SetValue(BusyIndicator15, "color", "gray");
SetValue(BusyIndicator9,  "secondary_color", "red");
SetValue(BusyIndicator12, "secondary_color", "red");
```

Data Grid

The Data Grid control shows a list of data entries; see Figure 18. This list has different configuration options. When the user clicks on the **Edit** button, an extra window is shown, where the user can edit the row's entries.

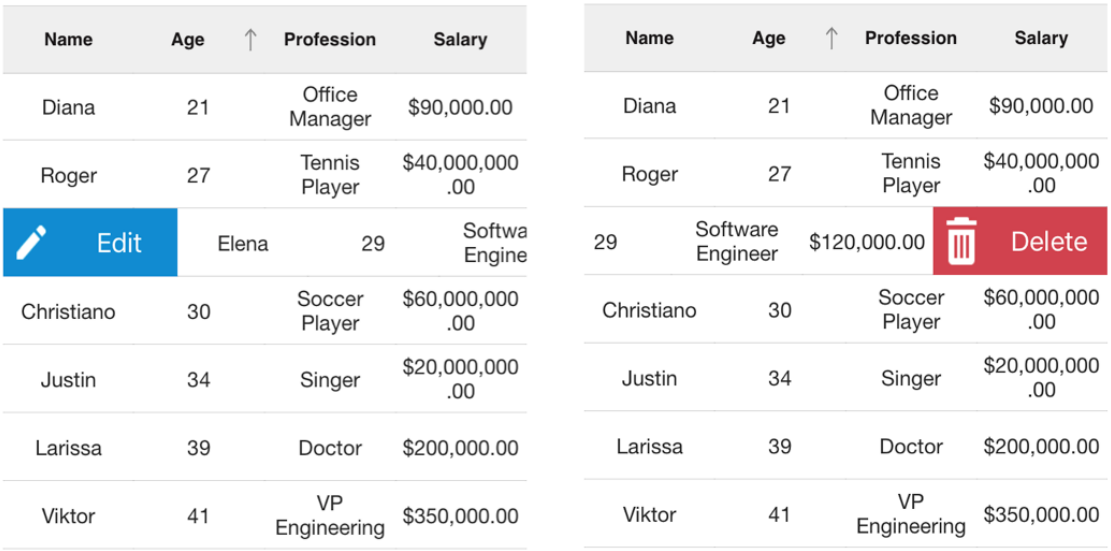


Figure 18: Syncfusion DataGrid After Swiping a Row to the Right and to the Left on iOS

The CSCS code that created the Data Grid in Figure 18 is shown in Code Listing 31.

Code Listing 31: The CSCS Code to Create Syncfusion Data Grid

```
AddSfDataGrid(locSFWidget,      "DataGrid", "", 600, 625);

listCols    = {"Name", "string", "Age", "number",
               "Profession", "string", "Salary", "currency"};
person1     = {"Christiano", "30", "Soccer Player",     "60000000"};
person2     = {"Roger",      "27", "Tennis Player",     "40000000"};
person3     = {"Sharon",     "47", "Actress",           "30000000"};
person4     = {"Justin",     "34", "Singer",            "20000000"};
person5     = {"Sergio",     "52", "CEO TheBank",       "5000000"};
person6     = {"Viktor",     "41", "VP Engineering",    "350000"};
person7     = {"Beat",       "44", "Psychiatrist",      "280000"};
person8     = {"Larissa",    "39", "Doctor",            "200000"};
person9     = {"Elena",      "29", "Software Engineer", "120000"};
person10    = {"Pedro",      "41", "Quality Engineer",  "115000"};
person11    = {"John",       "48", "Nuclear Physicist", "95000"};
person12    = {"Diana",      "21", "Office Manager",    "90000"};
person13    = {"Freddy",     "51", "Unemployed",        "20000"};
AddWidgetData(DataGrid, listCols, "columns");
AddWidgetData(DataGrid, person1,  "item");
AddWidgetData(DataGrid, person2,  "item");
AddWidgetData(DataGrid, person3,  "item");
AddWidgetData(DataGrid, person4,  "item");
AddWidgetData(DataGrid, person5,  "item");
AddWidgetData(DataGrid, person6,  "item");
AddWidgetData(DataGrid, person7,  "item");
AddWidgetData(DataGrid, person8,  "item");
AddWidgetData(DataGrid, person9,  "item");
```


www.dbooks.org

```
AddWidgetData(DataGrid, person10, "item");
AddWidgetData(DataGrid, person11, "item");
AddWidgetData(DataGrid, person12, "item");
AddWidgetData(DataGrid, person13, "item");

SetValue(DataGrid, "swipe_edit",   true);
SetValue(DataGrid, "swipe_delete", true);
SetValue(DataGrid, "allow_drag",   true);
SetValue(DataGrid, "allow_swap",   true);
SetValue(DataGrid, "allow_resize", true);
SetValue(DataGrid, "allow_sort",   true);
```

The column name configuration is taking place in:

```
AddWidgetData(DataGrid, listCols, "columns");
```

The format of the `listCols` list is the following:

```
{ columnName1, columnType1, columnName2, columnType2, …}
```

Note that we explicitly allow the user to sort by the column values, resize column widths, drag and drop rows in different places, and so on, as defined in the **`SetValue()`** functions at the end of Code Listing 31.

Spline Chart

The Spline Chart creates a continuous, piecewise polynomic function from some data points: see Figure 19. When you touch one of the black data points on the graph, you will see the exact value it represents.

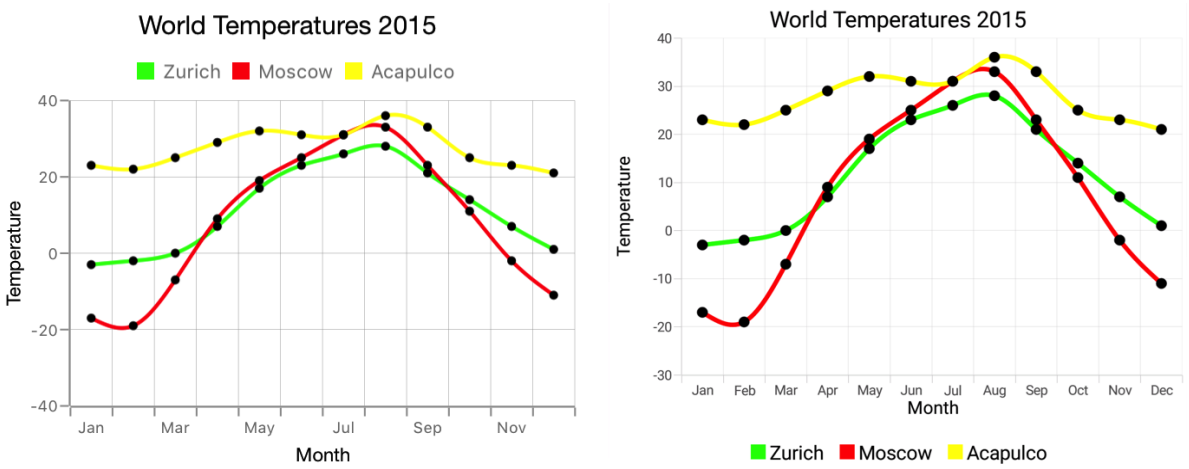


Figure 19: Syncfusion Spline Chart on iOS and on Android

The CSCS code that created the Spline charts in Figure 19 is shown in Code Listing 32.

Code Listing 32: The CSCS Code to Create Syncfusion Spline Chart

```
AddSfSplineGraph(locSFWidget, "SplineGraph", "", 600, 500);
chartData1 = {"Jan",-3, "Feb",-2, "Mar",0, "Apr",7, "May",17,
  "Jun",23, "Jul",26, "Aug", 28, "Sep",21, "Oct",14, "Nov",7, "Dec",1};
chartData2 = {"Jan",-17, "Feb",-19, "Mar",-7, "Apr",9, "May",19,
  "Jun",25, "Jul",31, "Aug",33, "Sep",23, "Oct",11, "Nov",-2, "Dec",-11};
chartData3 = {"Jan",23, "Feb",22, "Mar",25, "Apr",29, "May",32,
  "Jun",31, "Jul",31, "Aug",36, "Sep",33, "Oct",25, "Nov",23, "Dec",21};

AddWidgetData(SplineGraph, chartData1,  "Zurich",   "green:black");
AddWidgetData(SplineGraph, chartData2,  "Moscow",   "red:black");
AddWidgetData(SplineGraph, chartData3,  "Acapulco", "yellow:black");

SetText(SplineGraph,  "World Temperatures 2015");
SetValue(SplineGraph, "primary_axis",   "Month");
SetValue(SplineGraph, "secondary_axis", "Temperature");
```

Doughnut and Semi-Doughnut Charts

The Syncfusion Doughnut and Semi-Doughnut graphs are shown in Figure 20.

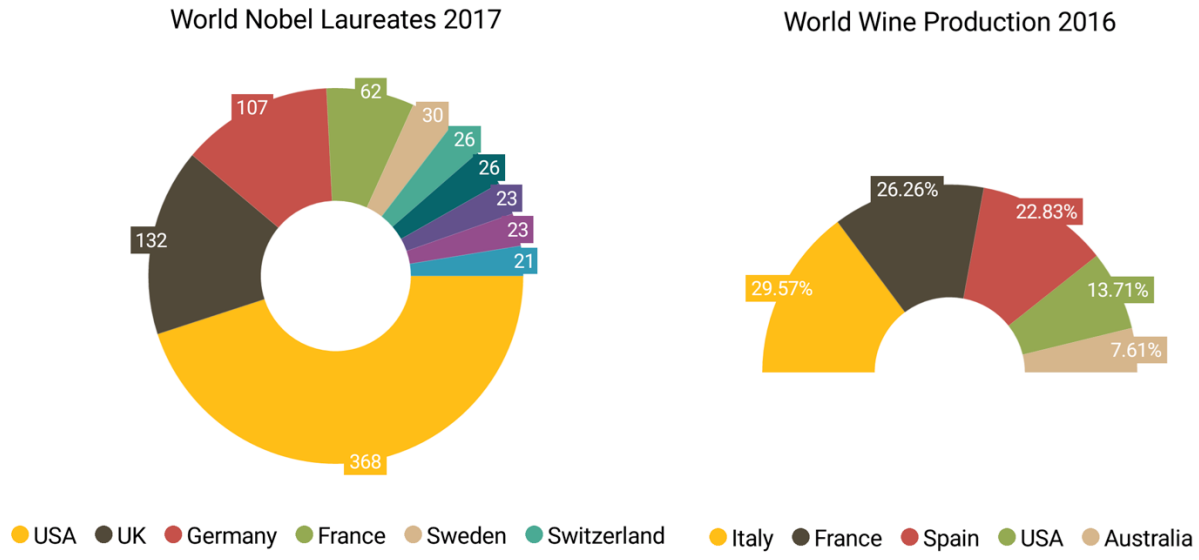


Figure 20: Syncfusion Doughnut and Semi-Doughnut Charts on Android

The CSCS code that created the charts in Figure 20 is shown in Code Listing 33. Note that the charts can have either a percentage value, or an absolute one. This is set up by an initialization parameter in the **`AddWidgetData()`** call (it can be either **`values`** or **`percent`**).

www.dbooks.org

We also define there from which angle to which angle the graph is going to be shown. **`0:360`** means the full circle, and **`180:360`** means the upper half. All other values between 0 and 360 are possible as well.

Code Listing 33: The CSCS Code to Create Syncfusion Doughnut and Semi-Doughnut Charts

```
AddSfDoughnutGraph(locSFWidget, "DoughnutGraph", "", 600, 500);
chartDataF = {"USA", 368, "UK", 132, "Germany", 107, "France", 62,
              "Sweden", 30, "Switzerland", 26, "Japan", 26,
              "Canada", 23, "Russia", 23, "Austria", 21};
AddWidgetData(DoughnutGraph, chartDataF,  "values", "0:360");
SetText(DoughnutGraph,     "World Nobel Laureates 2017");


AddSfDoughnutGraph(locSFWidget, "SemiDoughnutGraph", "", 600, 500);
chartDataD = {"Italy", 50900, "France", 45200, "Spain", 39300,
              "USA", 23600, "Australia", 13100};
AddWidgetData(SemiDoughnutGraph, chartDataD,  "percent", "180:360");
SetText(SemiDoughnutGraph, "World Wine Production 2016");
```

Summary

In this chapter, we saw a few of the Syncfusion controls. There are also many controls that we did not have time to see here, for instance, a calculator, an image rotator, different list views, a diagram creator, and many others.

This chapter is also important because it can give you an idea of how to add other existing widget frameworks to CSCS.

In the next chapter, we are going to dig deeper into Syncfusion controls, looking into how to create known file types.

Chapter 6 Adding PDF, Word, and Excel Functionality from Syncfusion to CSCS

"A guy who builds a nice chair doesn't owe money to everyone who ever has built a chair."
Mark Zuckerberg

In this chapter, we are going to see how to create different types of documents using the Syncfusion framework. Namely, we are going to see how to create PDF, Microsoft Word, and Excel files.

One of the differences with the controls that we developed in the previous chapter is that the Syncfusion implementation for iOS and Android is almost the same (including the class and the method names). Since the platform-specific code is minimal, this allows us to use the same code for both platforms, putting the Syncfusion wrappers in the shared project.

Adding PDF file creation to CSCS

To add a Syncfusion widget for PDF (Portable Document Format) file creation, we extend the **`iOSVariable`** class on iOS, and the **`DroidVariable`** class on Android. Check out the details in Code Listing 34.

Code Listing 34: A Fragment of the SfPdf Class

```
#if __ANDROID__
using scripting.Droid;
using BASE_VARIABLE = scripting.Droid.DroidVariable;
using UTILS = scripting.Droid.UtilsDroid;
#elif __IOS__
using scripting.iOS;
using BASE_VARIABLE = scripting.iOS.iOSVariable;
using UTILS = scripting.iOS.UtilsiOS;
#endif

namespace scripting
{
  public class SfPdf : BASE_VARIABLE
  {
    PdfDocument m_document;
    PdfLoadedDocument m_loadedDoc;
    PdfPage m_page;
    PdfGraphics m_grapics;
    PdfStandardFont m_font = new PdfStandardFont(
             PdfFontFamily.Helvetica, 12);
```


www.dbooks.org

```
public void Init()
{
  if (m_document != null) {
    return;
  }
  m_document = new PdfDocument();
  m_page = m_document.Pages.Add();
  m_grapics = m_page.Graphics;
}

public void AddText(string text, int x, int y, PdfBrush color)
{
  m_grapics.DrawString(text, m_font, color, x, y);
}

public void AddImage(string imagePath, int x, int y, int w, int h)
{
  Stream pngImageStream = UTILS.ImageToStream(imagePath);
  PdfImage pngImage = new PdfBitmap(pngImageStream);
  m_grapics.DrawImage(pngImage, x, y, w, h);
}

public void AddLine(int x1, int y1, int x2, int y2,
                    PdfBrush color, float width)
{
  PdfPen pen = new PdfPen(color, width);
  m_grapics.DrawLine(pen, x1, y1, x2, y2);
}

public void AddRectangle(int x, int y, int w, int h, PdfBrush color)
{
  m_grapics.DrawRectangle(color, x, y, w, h);
}

public void AddPie(int x, int y, int w, int h, float startAngle,
                   float sweepAngle, PdfBrush color, float width)
{
  PdfPen pen = new PdfPen(color, width);
  m_grapics.DrawPie(pen, x, y, w, h, startAngle, sweepAngle);
}

public void Save(string filename)
{
  MemoryStream stream = new MemoryStream();
  if (m_loadedDoc != null) {
    m_loadedDoc.Save(stream);
  } else {
    Init();
    m_document.Save(stream);
```

```
        m_document.Close(true);
      }
      stream.Position = 0;
#if __ANDROID__
      SaveAndroid.Save(filename, "application/pdf", stream);
#elif __IOS__
      PreviewController.Save(filename, "application/pdf", stream);
#endif
      m_document = null;
      Utils.SaveFile(filename, stream);
    }
  }
}
```

To register all the Syncfusion PDF-related functions with the parser, we use the following statements:

```
ParserFunction.RegisterFunction("SfPdfNew", new CreatePdf());
ParserFunction.RegisterFunction("SfPdfOpen", new OpenPdf());
ParserFunction.RegisterFunction("SfSetPdfText", new SetPdfText());
ParserFunction.RegisterFunction("SfSetPdfImage", new SetPdfImage());
ParserFunction.RegisterFunction("SfSetPdfLine", new SetPdfLine());
ParserFunction.RegisterFunction("SfSetPdfRectangle", new SetPdfRectangle());
ParserFunction.RegisterFunction("SfSetPdfPie", new SetPdfPie());
ParserFunction.RegisterFunction("SfSetPdfFont", new SetPdfFont());
ParserFunction.RegisterFunction("SfSavePdf", new SavePdf());
```



Tip: All the classes deriving from the `ParserFunction` class are the glue between the CSCS parser and our C# wrapper over the Syncfusion classes.

See fragments of some of them in Code Listing 35.

Code Listing 35: Fragments of the CreatePdf and SetPdfText Classes

```
public class CreatePdf : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 0, m_name);

    SfPdf pdf = new SfPdf(true);
    return pdf;
  }
}

public class SetPdfText : ParserFunction
{
```


www.dbooks.org

```
  protected override Variable Evaluate(ParsingScript script)
  {
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 3, m_name);

    SfPdf pdf = args[0] as SfPdf;
    Utils.CheckNotNull(pdf, m_name);

    string text = Utils.GetSafeString(args, 1);
    int x = Utils.GetSafeInt(args, 2);
    int y = Utils.GetSafeInt(args, 3);

    string colorStr = Utils.GetSafeString(args, 4, "black");
    PdfBrush color = SfUtils.String2PdfColor(colorStr);

    pdf.Init();
    pdf.AddText(text, x, y, color);

    ParserFunction.UpdateFunction(pdf);
    return pdf;
  }
}
```

Now you can create PDF files with a few lines of CSCS code: check out Code Listing 36.

Code Listing 36: The CSCS Code to Create a PDF File

```
pdf = SfPdfNew();

SfSetPdfFont(pdf, "Helvetica", 24, true);
SfSetPdfText(pdf, "Take a look at this dog:", 0, 10, "blue");

SfSetPdfLine(pdf, 0, 50, 640, 50, "green", 4);
SfSetPdfImage(pdf, "funnyDog", 0, 70, 640, 420);
SfSetPdfLine(pdf, 0, 520, 640, 520, "green", 4);

SfSavePdf(pdf, "hello.pdf");
```

The result of running Code Listing 36 is shown in Figure 21.

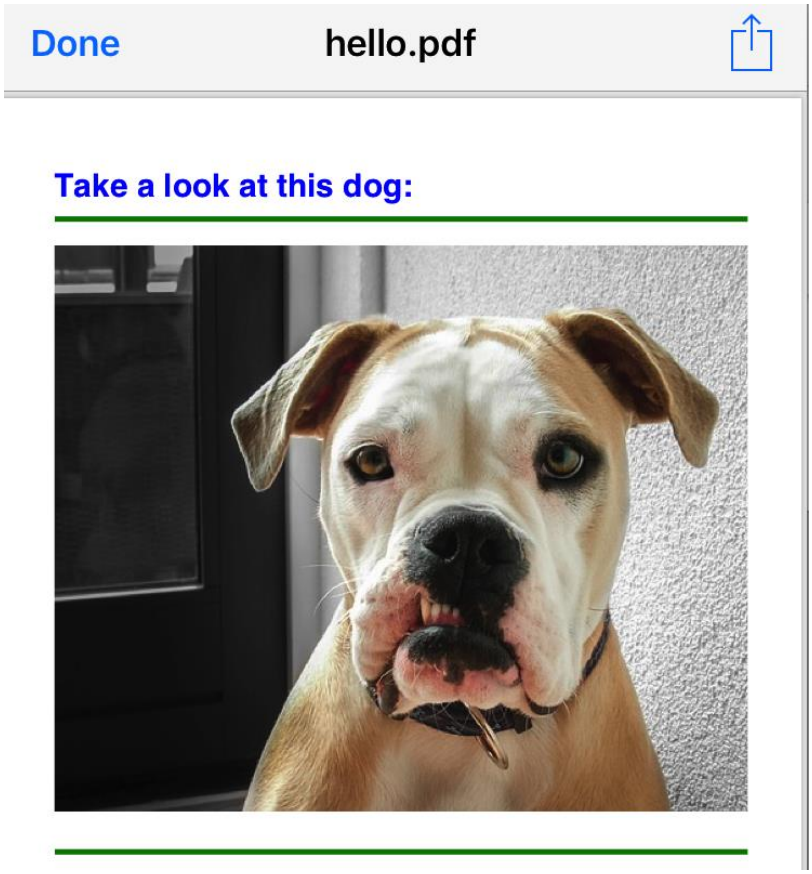


Figure 21: A Sample PDF File Created Using Syncfusion

Adding Microsoft Word file creation to CSCS

Adding the Syncfusion widget for Microsoft Word is very similar to the PDF widget. See Code Listing 37 for a fragment of the **`SfWord`** class. This class is a wrapper over the Syncfusion Word widget.

Code Listing 37: A Fragment of the SfWord Class

```
public class SfWord : BASE_VARIABLE
{
  WordDocument m_document;
  WSection m_section;
  IWParagraph m_paragraph;

  IWTable m_table;
  WTableRow m_row;
  float m_margins;

  void Init()
  {
```


www.dbooks.org

```
    if (m_document != null) {
      return;
    }
    m_document = new WordDocument();
    m_section = m_document.AddSection() as WSection;
    m_section.PageSetup.Margins.All = m_margins;
    AddStyles();
  }
  void CheckParagraph()
  {
    if (m_paragraph == null) {
      m_paragraph = m_section.AddParagraph();
    }
  }
  public void AddTable(int rows, int cols, string styleStr)
  {
    m_table = m_section.AddTable();
    m_table.ResetCells(rows, cols);
    m_table.TableFormat.Borders.BorderType =
            SfUtils.String2BorderStyle(styleStr);
    m_table.TableFormat.IsAutoResized = true;
  }
  public void AddText(string text)
  {
    CheckParagraph();
    m_paragraph.AppendText(text);
  }
}
```

To register the Word-specific functions with the parser, we use the statements in Code Listing 38.

Code Listing 38: Registration of the Syncfusion Word Classes with the Parser

```
ParserFunction.RegisterFunction("SfWordNew", new CreateWord());
ParserFunction.RegisterFunction("SfWordOpen", new OpenWord());
ParserFunction.RegisterFunction("SfAddWordText", new AddWordText());
ParserFunction.RegisterFunction("SfAddWordTextRange",
                                new AddWordTextRange());
ParserFunction.RegisterFunction("SfAddWordImage", new AddWordImage());
ParserFunction.RegisterFunction("SfAddWordTable", new AddWordTable());
ParserFunction.RegisterFunction("SfAddWordParagraph",
                                new AddWordParagraph());
ParserFunction.RegisterFunction("SfApplyWordStyle", new ApplyWordStyle());
ParserFunction.RegisterFunction("SfSaveWord", new SaveWord());
```

We won't show the implementation of the **`CreateWord()`**, **`OpenWord()`**, and other methods, since they are very similar to Code Listing 35 for PDF documents. Using the functions defined in Code Listing 38, it is easy to create non-trivial Word documents with just a few lines of CSCS

code. Code Listing 39 contains some CSCS code to create a Word document containing images and a table using different colors, fonts, and styles.

Code Listing 39: CSCS Code to Create a Microsoft Word Document

```
word = SfWordNew();

SfAddWordParagraph(word, "header");
SfApplyWordStyle(word, "Normal2", "left");
SfAddWordTextRange(word, "Adventure Cycles", "Calibri", 12, "red");

SfAddWordImage(word, "AdventureCycle", 243.5, -24, "InFrontOfText",
                     "Column", "Milesargin", 20, 15);
SfAddWordParagraph(word);
SfAddWordParagraph(word);
SfApplyWordStyle(word, "Heading 1", "center");
SfAddWordTextRange(word, "Adventure Works Cycles", "Calibri", 18);

SfAddWordParagraph(word, "normal", 36);
SfAddWordTextRange(word, "In 2000, Adventure Works Cycles bought a small ma
nufacturing plant, Importadores Neptuno...", "Calibri", 12);

SfAddWordParagraph(word);
SfAddWordParagraph(word);
SfApplyWordStyle(word, "Heading 1", "left");
SfAddWordTextRange(word, "Product Overview", "Calibri", 16);
SfAddWordParagraph(word);

SfAddWordTable(word, 3, 2, "None");
SfAddWordParagraph(word, "table", 0, 0, 0);
SfAddWordImage(word, "Mountain200", -5, 0, "TopAndBottom",
                     "Column", "Paragraph", 75, 75);
SfAddWordParagraph(word);
SfAddWordParagraph(word, "table", 0, 0, 1);
SfApplyWordStyle(word, "Heading 1");
SfAddWordText(word, "Mountain-200");
SfAddWordParagraph(word, "table", 0, 0, 1);
SfAddWordTextRange(word, "Product No: BK-M68B-38\n",
                         "Times New Roman", 12);
SfAddWordTextRange(word, "Size: 38\n", "Times New Roman", 12);
SfAddWordTextRange(word, "Weight: 25\n", "Times New Roman", 12);
SfAddWordTextRange(word, "Price: $2,294.99\n", "Times New Roman", 12);
SfAddWordParagraph(word, "table", 0, 0, 1);

SfAddWordParagraph(word, "table", 0, 1, 0);
SfAddWordParagraph(word, "table", 0, 1, 0);
SfApplyWordStyle(word, "Heading 1");
SfAddWordText(word, "Mountain-300");
```


www.dbooks.org

```
SfAddWordParagraph(word, "table", 0, 1, 0);
SfAddWordTextRange(word, "Product No: BK-M47B-38\n",
                         "Times New Roman", 12);
SfAddWordTextRange(word, "Size: 35\n", "Times New Roman", 12);
SfAddWordTextRange(word, "Weight: 22\n", "Times New Roman", 12);
SfAddWordTextRange(word, "Price: $1,089.99\n", "Times New Roman", 12);
SfAddWordParagraph(word, "table", 0, 1, 0);

SfAddWordParagraph(word, "table", 0, 1, 1);
SfAddWordParagraph(word, "table", 0, 1, 1);
SfAddWordImage(word, "Mountain300", -15, 8, "TopAndBottom",
                     "Column", "Paragraph", 75, 75);
SfAddWordParagraph(word, "table", 0, 2, 0);
SfAddWordParagraph(word, "table", 0, 2, 0);
SfApplyWordStyle(word, "Heading 1");
SfAddWordImage(word, "Road550W", -5, 0, "TopAndBottom",
                     "Column", "Paragraph", 75, 75);
SfAddWordParagraph(word, "table", 0, 2, 1);
SfApplyWordStyle(word, "Heading 1");
SfAddWordText(word, "Road-150");
SfAddWordParagraph(word, "table", 0, 2, 1);
SfAddWordTextRange(word, "Product No: BK-R93R-44\n",
                         "Times New Roman", 12);
SfAddWordTextRange(word, "Size: 44\n", "Times New Roman", 12);
SfAddWordTextRange(word, "Weight: 14\n", "Times New Roman", 12);
SfAddWordTextRange(word, "Price: $3,599.99\n", "Times New Roman", 12);

SfAddWordParagraph(word, "table", 0, 2, 1);
SfApplyWordStyle(word, "Heading 1");

SfSaveWord(word, "hello.docx");
```

The result of running Code Listing 39 is shown in Figure 22.

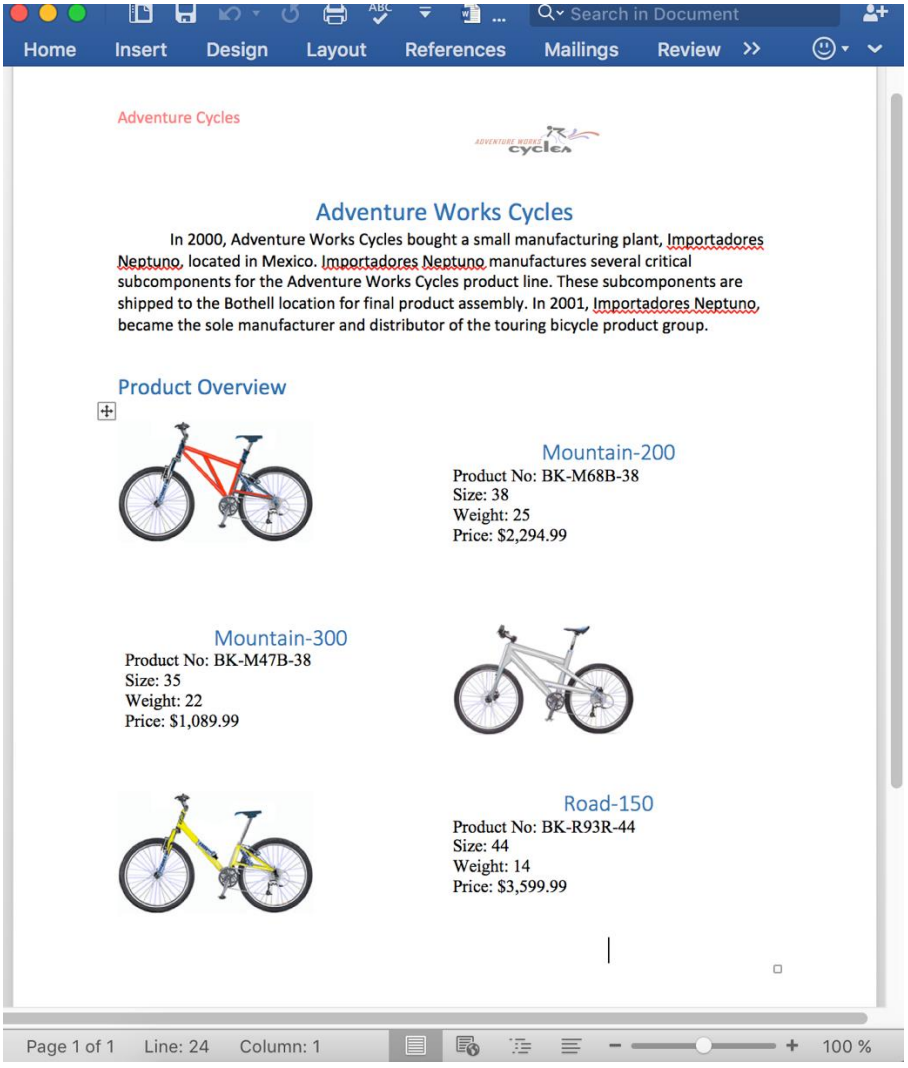


Figure 22: A Word Document Created by CSCS Using Syncfusion

Adding Microsoft Excel file creation to CSCS

Adding the Syncfusion widget for the Microsoft Excel file creation is very similar to the PDF and Word file creation as well. See Code Listing 40 for a fragment of the **`SfExcel`** class. This class is a wrapper over the Syncfusion Excel widget.

Code Listing 40: A Fragment of the SfExcel Class

```
public class SfExcel : BASE_VARIABLE
{
  ExcelEngine m_excelEngine;
  IApplication m_application;
  IWorkbook m_workbook;
  IWorksheet m_sheet;
  int m_numberSheets;
```


www.dbooks.org

```
  public void Init()
  {
    if (m_excelEngine != null) {
      return;
    }
    m_excelEngine = new ExcelEngine();
    m_application = m_excelEngine.Excel;
    m_application.DefaultVersion = ExcelVersion.Excel2013;

    m_workbook = m_application.Workbooks.Create(m_numberSheets);
    m_workbook.Version = ExcelVersion.Excel2013;

    m_sheet = m_workbook.Worksheets[0];
    m_sheet.EnableSheetCalculations();
  }
  public void AddChart(string range, string title, int top, int bottom,
                       int left, int right)
  {
    IChartShape chart = m_sheet.Charts.Add();
    chart.DataRange = m_sheet[range];
    chart.ChartTitle = title;
    chart.HasLegend = false;
    chart.TopRow = top;
    chart.LeftColumn = left;
    chart.RightColumn = right;
    chart.BottomRow = bottom;
  }
  public void AddWorksheet(string title)
  {
    m_sheet = m_workbook.Worksheets.Create(title);
  }
  public void SetWorksheetName(string title)
  {
    m_sheet.Name = title;
  }
  public void ActivateWorksheet(string title)
  {
    m_sheet = m_workbook.Worksheets[title];
  }
}
```

Code Listing 41 shows the statements to register the Excel-specific functions with the parser.

Code Listing 41: Registration of the Syncfusion Excel Classes with the Parser

```
ParserFunction.RegisterFunction("SfExcelNew", new CreateExcel());
ParserFunction.RegisterFunction("SfExcelOpen", new OpenExcel());
```

```
ParserFunction.RegisterFunction("SfExcelSet", new SetExcelOption());
ParserFunction.RegisterFunction("SfAddExcelWorksheet",
                                new AddExcelWorksheet());
ParserFunction.RegisterFunction("SfRenameExcelWorksheet",
                                new RenameExcelWorksheet());
ParserFunction.RegisterFunction("SfActivateExcelWorksheet",
                                new ActivateExcelWorksheet());
ParserFunction.RegisterFunction("SfAddExcelChart", new AddExcelChart());
ParserFunction.RegisterFunction("SfSaveExcel", new SaveExcel());
```

We won't show the implementation of the **`CreateExcel()`**, **`OpenExcel()`**, and other methods, since they are very similar to Code Listing 35 for PDF.

Using the functions defined in Code Listing 41, it is easy to create non-trivial Excel documents with just a few lines of CSCS code.

Code Listing 42 contains some CSCS code to create an Excel document with charts and formulas using different colors, fonts, and styles.

Code Listing 42: CSCS Code to Create a Microsoft Word Document

```
excel = SfExcelNew();

SfRenameExcelWorksheet(excel, "Expenses");
SfAddExcelWorksheet(excel, "Extras");

SfExcelSet(excel, "A2", "col_width", 18);
SfExcelSet(excel, "B2", "col_width", 12);
SfExcelSet(excel, "C2", "col_width", 12);
SfExcelSet(excel, "D2", "col_width", 12);

SfExcelSet(excel, "A2:D2", "merge",  true);
SfExcelSet(excel, "A2", "text",       "Expense Report");
SfExcelSet(excel, "A2", "font_name",  "Verdana");
SfExcelSet(excel, "A2", "font_color", "sky_blue");
SfExcelSet(excel, "A2", "font_size",  28);
SfExcelSet(excel, "A2", "bold", true);
SfExcelSet(excel, "A2", "horizontal_alignment", "center");
SfExcelSet(excel, "A2", "row_height",     34);

SfExcelSet(excel, "A4", "text",           "Employee");
SfExcelSet(excel, "B4", "text",           "Roger Federer");
SfExcelSet(excel, "A4:B7", "font_name",   "Verdana");
SfExcelSet(excel, "A4:B7", "font_size",   11);
SfExcelSet(excel, "A4:B7", "bold",        true);
SfExcelSet(excel, "A4:A7", "horizontal_alignment", "left");
SfExcelSet(excel, "B4:B7", "horizontal_alignment", "right");

SfExcelSet(excel, "B4:D4", "merge",       true);
```


www.dbooks.org

```
SfExcelSet(excel, "A9:D20", "font_name",  "Verdana");
SfExcelSet(excel, "A9:D20", "font_size",  11);

SfExcelSet(excel, "A5", "text",           "Department");
SfExcelSet(excel, "B5", "text",           "Administration");
SfExcelSet(excel, "B5:D5", "merge",       true);

SfExcelSet(excel, "A6", "text",           "Week Ending");
SfExcelSet(excel, "B6", "number_format",  "m/d/yyyy");
SfExcelSet(excel, "B6", "date_time",      "12/12/2012");
SfExcelSet(excel, "B6:D6", "merge",       true);

SfExcelSet(excel, "A7", "text",           "Mileage Rate");
SfExcelSet(excel, "B6", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B7", "number",         0.7);
SfExcelSet(excel, "B7:D7", "merge",       true);

SfExcelSet(excel, "A10", "text",           "Miles Driven");
SfExcelSet(excel, "A11", "text",           "Reimbursement");
SfExcelSet(excel, "A12", "text",           "Parking/Tolls");
SfExcelSet(excel, "A13", "text",           "Auto Rental");
SfExcelSet(excel, "A14", "text",           "Lodging");
SfExcelSet(excel, "A15", "text",           "Breakfast");
SfExcelSet(excel, "A16", "text",           "Lunch");
SfExcelSet(excel, "A17", "text",           "Dinner");
SfExcelSet(excel, "A18", "text",           "Snacks");
SfExcelSet(excel, "A19", "text",           "Others");
SfExcelSet(excel, "A20", "text",           "Total");
SfExcelSet(excel, "A20:D20", "color",      "yellow");
SfExcelSet(excel, "A20:D20", "font_color", "black");
SfExcelSet(excel, "A20:D20", "bold",       true);

SfExcelSet(excel, "B9:D9", "horizontal_alignment",     "right");
SfExcelSet(excel, "B9:D9", "vertical_alignment",       "center");
SfExcelSet(excel, "B9:D9", "color",                    "light_green");
SfExcelSet(excel, "B9:D9", "bold",                     true);
SfExcelSet(excel, "B9:D9", "font_color",               "white");

SfExcelSet(excel, "A9", "text",           "Expenses");
SfExcelSet(excel, "A9", "color",          "pink");
SfExcelSet(excel, "A9", "font_color",     "white");
SfExcelSet(excel, "A9", "bold",           true);

SfExcelSet(excel, "B9", "text",           "Day 1");
SfExcelSet(excel, "B10", "number",         80);
SfExcelSet(excel, "B11", "number_format", "$#,##0.00");
SfExcelSet(excel, "B11", "formula",       "=(B7*B10)");
SfExcelSet(excel, "B12", "number_format",  "$#,##0.00");
```

```
SfExcelSet(excel, "B12", "number",         "11");
SfExcelSet(excel, "B13", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B13", "number",         "9");
SfExcelSet(excel, "B14", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B14", "number",         "13");
SfExcelSet(excel, "B15", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B15", "number",         "7");
SfExcelSet(excel, "B16", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B16", "number",         "21");
SfExcelSet(excel, "B17", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B17", "number",         "22");
SfExcelSet(excel, "B18", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B18", "number",         "11");
SfExcelSet(excel, "B19", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B19", "number",         "5");
SfExcelSet(excel, "B20", "number_format",  "$#,##0.00");
SfExcelSet(excel, "B20", "formula",        "=SUM(B11:B19)");

SfExcelSet(excel, "C9", "text",           "Day 2");
SfExcelSet(excel, "C10", "number",         120);
SfExcelSet(excel, "C11", "number_format", "$#,##0.00");
SfExcelSet(excel, "C11", "formula",       "=(B7*B10)");
SfExcelSet(excel, "C12", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C12", "number",         "21");
SfExcelSet(excel, "C13", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C13", "number",         "17");
SfExcelSet(excel, "C14", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C14", "number",         "13");
SfExcelSet(excel, "C15", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C15", "number",         "7");
SfExcelSet(excel, "C16", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C16", "number",         "24");
SfExcelSet(excel, "C17", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C17", "number",         "22");
SfExcelSet(excel, "C18", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C18", "number",         "18");
SfExcelSet(excel, "C19", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C19", "number",         "15");
SfExcelSet(excel, "C20", "number_format",  "$#,##0.00");
SfExcelSet(excel, "C20", "formula",        "=SUM(C11:C19)");

SfExcelSet(excel, "D9", "text",            "Day 3");
SfExcelSet(excel, "D10", "number",         80);
SfExcelSet(excel, "D11", "number_format", "$#,##0.00");
SfExcelSet(excel, "D11", "formula",       "=(B7*B10)");
SfExcelSet(excel, "D12", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D12", "number",         "13");
SfExcelSet(excel, "D13", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D13", "number",         "9");
```


www.dbooks.org

```
SfExcelSet(excel, "D14", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D14", "number",         "13");
SfExcelSet(excel, "D15", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D15", "number",         "17");
SfExcelSet(excel, "D16", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D16", "number",         "21");
SfExcelSet(excel, "D17", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D17", "number",         "22");
SfExcelSet(excel, "D18", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D18", "number",         "14");
SfExcelSet(excel, "D19", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D19", "number",         "15");
SfExcelSet(excel, "D20", "number_format",  "$#,##0.00");
SfExcelSet(excel, "D20", "formula",        "=SUM(D11:D19)");

SfExcelSet(excel, "A10:D10", "font_color", "dark_blue");

SfAddExcelChart(excel, "A9:D20", "", 23, 40, 1, 8);
SfActivateExcelWorksheet(excel, "Expenses");

SfSaveExcel(excel, "hello.xlsx");
```

The result of running Code Listing 42 is shown in Figure 23.

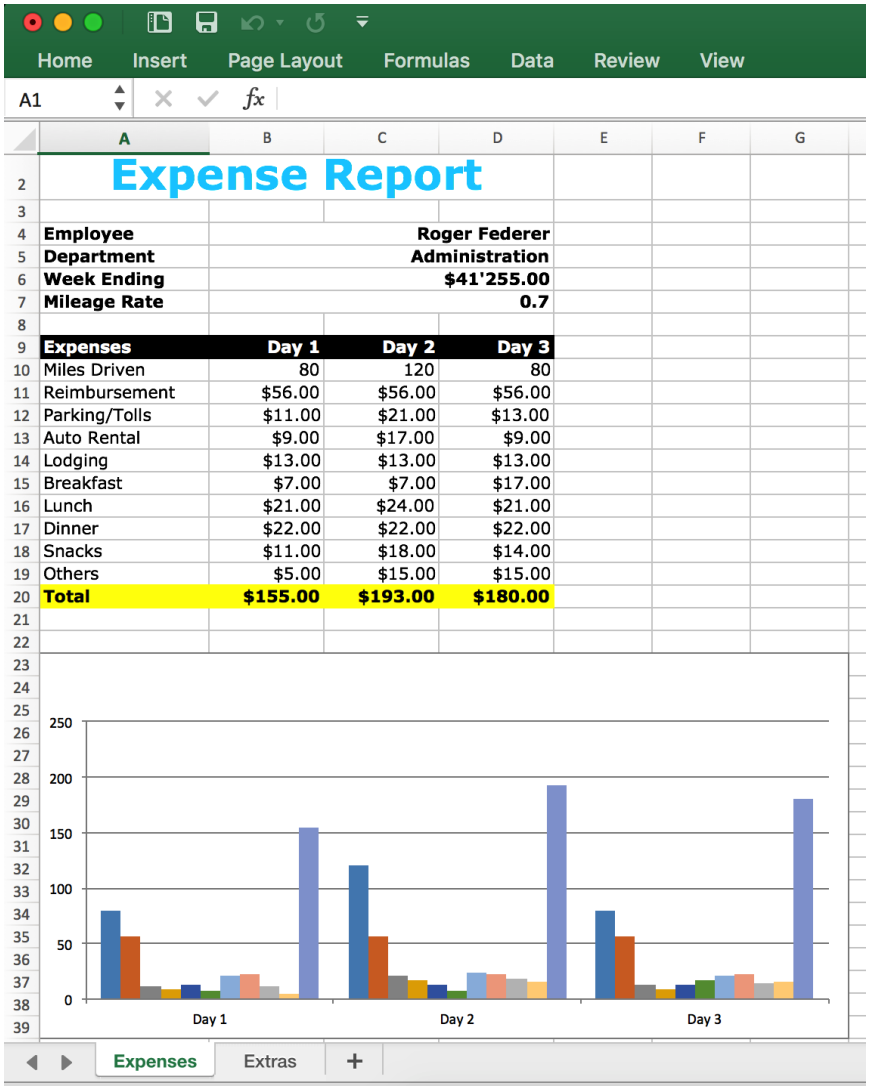


Figure 23: An Excel Document Created by CSCS using Syncfusion

Summary

In this chapter, we saw how to create PDF, Microsoft Word, and Excel documents using the Syncfusion framework. You can create Microsoft PowerPoint presentations with it as well. If you want to add PowerPoint to CSCS, just mimic what we did in this chapter.

In the next chapter, we are going to see how to add non-GUI related functionality to CSCS, in particular, text-to-speech and speech recognition.

www.dbooks.org

Chapter 7 Text-To-Speech and Speech Recognition in CSCS

"Think before you speak. Read before you think."
Fran Lebowitz

In this chapter, we are going to look at some non-GUI related features, namely, text-to-speech and speech recognition for iOS and Android. We will see that it is very easy to add these features to CSCS as well.

Text-to-speech in CSCS

Code Listing 43 contains a fragment of the text-to-speech implementation for Android. Some non-essential details are not shown.

Code Listing 43: A Fragment of the TTS Implementation for Android

```
public class TTS : Java.Lang.Object, TextToSpeech.IOnInitListener,
                                     IDisposable
{
  public static  bool   Sound { set; get; }           = true;
  public static  float  SpeechRate { set; get; }      = 1.0f;
  public static  float  Volume { set; get; }          = 0.7f;
  public static  float  PitchMultiplier { set; get; } = 1.0f;
  public static  string Voice { set; get; }           = "en-US";

  static Context   m_context;
  TextToSpeech     m_textToSpeech;
  Java.Util.Locale m_language;

  TTS(string voice)
  {
    m_language = UtilsDroid.LocaleFromString(voice, false);
    m_textToSpeech = new TextToSpeech(Application.Context,
                         this, "com.google.android.tts");
    m_textToSpeech.SetLanguage(m_language);
  }

  public void Speak(string text, bool force = false)
  {
    if (!Sound && !force) {
      return;
    }
    m_textToSpeech.Stop();
```

```
    m_textToSpeech.SetLanguage(m_language);
    m_textToSpeech.SetSpeechRate(SpeechRate);
    m_textToSpeech.SetPitch(PitchMultiplier);

    if (Build.VERSION.SdkInt >= BuildVersionCodes.Lollipop) { // API 21
      string utteranceId = text.GetHashCode() + "";
      m_textToSpeech.Speak(text, QueueMode.Flush, null, utteranceId);
    } else {
      m_textToSpeech.Speak(text, QueueMode.Flush, null);
    }
  }

  void TextToSpeech.IOnInitListener.OnInit(OperationResult status)
  {
    if (!Sound) {
      return;
    }
    if (status == OperationResult.Error) {
      m_textToSpeech.SetLanguage(Java.Util.Locale.Default);
      return;
    }
    m_textToSpeech.SetLanguage(m_language);
  }

  public static void OnTTSResult(int req, Result res, Intent data)
  {
    if (req == TTS_CHECK_DATA || req == TTS_INSTALLED_DATA) {
      IList<string> availableLanguages = data.GetStringArrayListExtra(
                    TextToSpeech.Engine.ExtraAvailableVoices);
      IList<string> unavailableLanguages = data.GetStringArrayListExtra(
                    TextToSpeech.Engine.ExtraUnavailableVoices);
      if (availableLanguages.Count <= 1 && m_context != null) {
        var installTTS = new Intent();
        installTTS.SetAction(TextToSpeech.Engine.ActionInstallTtsData);
        m_context.StartActivity(installTTS);
      }
    }
  }
}
```

To register the text-to-speech functionality with the parser, we use the following calls:

```
ParserFunction.RegisterFunction("InitTTS", new InitTTSFunction());
ParserFunction.RegisterFunction("Speak",   new SpeakFunction());
```

The implementation of the gluing class **`SpeakFunction`** is shown in Code Listing 44.

www.dbooks.org

Code Listing 44: The Implementation of the SpeakFunction Class

```
public class SpeakFunction : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 1, m_name);

    string phrase = args[0].AsString();
    TTS.Voice     = Utils.GetSafeString(args, 1, TTS.Voice);
    bool force    = Utils.GetSafeInt(args, 2) != 0;

    TTS tts = new TTS(TTS.Voice);
    tts.Speak(phrase, force);

    return Variable.EmptyInstance;
  }
}
```

Speech recognition in CSCS

For the speech recognition functionality, the app requires some permissions. On iOS, you have to add NSSpeechRecognitionUsageDescription and NSMicrophoneUsageDescription keys to your **Info.plist** file. Figure 24 shows how it will look in Visual Studio.

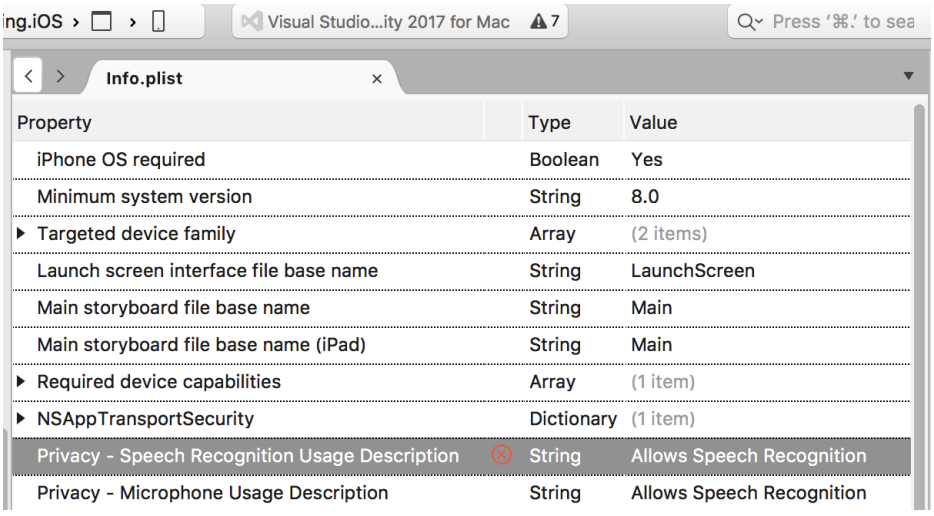


Figure 24: Speech Recognition in Info.plist on iOS

On Android, I use the SpeechRecognizer class for speech recognition. The following property must be added to your **AndroidManifest.xml** file:

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

The essential details of the speech recognition on iOS are shown in Code Listing 45.

Code Listing 45: A Fragment of the Speech Recognition Implementation for iOS

```
public class STT
{
  public Action<string> OnSpeechOK;
  public Action<string> OnSpeechError;

  static public string Voice { set; get; } = "en-US";

  static UIViewController                       m_controller;
  static AVAudioEngine                          AudioEngine;
  static SFSpeechRecognizer                     SpeechRecognizer;
  static SFSpeechAudioBufferRecognitionRequest LiveSpeechRequest;
  static SFSpeechRecognitionTask                RecognitionTask;
  static System.Timers.Timer                    m_speechTimer;

  static int m_timeout;
  static DateTime m_lastSpeech;
  static DateTime m_startSpeech;

  public static bool IsCancelled   { get; set; }
  public static bool IsRecording   { get; set; }
  public static bool SpeechEnabled { get; set; }
  public string      LastResult    { get; set; }

  public static void RequestAuthorization()
  {
    SFSpeechRecognizer.RequestAuthorization(
       (SFSpeechRecognizerAuthorizationStatus status) => {
      switch (status) {
        case SFSpeechRecognizerAuthorizationStatus.Authorized:
          SpeechEnabled = true;
          break;
        case SFSpeechRecognizerAuthorizationStatus.Denied:
          SpeechEnabled = false;
          break;
        case SFSpeechRecognizerAuthorizationStatus.NotDetermined:
          SpeechEnabled = false;
          break;
        case SFSpeechRecognizerAuthorizationStatus.Restricted:
          SpeechEnabled = true;
          break;
      }
    });
  }
  public void StartRecording(string voice, bool longTimeout = false)
```


www.dbooks.org

```
{
  if (!SpeechEnabled) {
    return;
  }
  AudioEngine = new AVAudioEngine();
  NSLocale voiceLocale = NSLocale.FromLocaleIdentifier(voice);
  SpeechRecognizer = new SFSpeechRecognizer(voiceLocale);
  LiveSpeechRequest = new SFSpeechAudioBufferRecognitionRequest();

  NSError error;
  var audioSession = AVAudioSession.SharedInstance();
  audioSession.SetCategory(AVAudioSessionCategory.Record);
  audioSession.SetMode(AVAudioSession.ModeMeasurement, out error);
  if (error != null) {
    OnSpeechError?.Invoke("Audio session error: " + error.ToString());
    return;
  }
  audioSession.SetActive(true,
      AVAudioSessionSetActiveOptions.NotifyOthersOnDeactivation);
  LiveSpeechRequest.ShouldReportPartialResults = true;

  var node = AudioEngine.InputNode;
  if (node == null) {
    OnSpeechError?.Invoke("Couldn't initialize Speech Input");
    return;
  }

  RecognitionTask = SpeechRecognizer.GetRecognitionTask(
                    LiveSpeechRequest,
                   (SFSpeechRecognitionResult result, NSError err) => {
    if (IsCancelled) {
      node.RemoveTapOnBus(0);
      return;
    }
    if (err != null) {
      OnSpeechError?.Invoke(err.ToString());
    } else if (result != null) {
      LastResult = result.BestTranscription.FormattedString;
      m_lastSpeech = DateTime.Now;
      if (result.Final) {
        OnSpeechOK?.Invoke(LastResult);
      }
    }
    if ((result != null && result.Final) || err != null) {
      IsRecording = false;
      AudioEngine.Stop();
      m_speechTimer.Close();
    }
  });
```

```
    var recordingFormat = node.GetBusOutputFormat(0);
    node.InstallTapOnBus(0, 1024, recordingFormat,
      (AVAudioPcmBuffer buffer, AVAudioTime when) => {
      LiveSpeechRequest.Append(buffer);
    });

    // Start recording
    AudioEngine.Prepare();
    AudioEngine.StartAndReturnError(out error);

    if (error != null) {
      OnSpeechError?.Invoke("Speech init error: " + error.ToString());
      IsRecording = false;
      return;
    }
    IsRecording = true;
    IsCancelled = false;
    LastResult = "";
    m_lastSpeech = DateTime.MaxValue;
    m_startSpeech = DateTime.Now;
    m_timeout = longTimeout ? m_phraseTimeout : m_wordTimeout;

    m_speechTimer = new System.Timers.Timer(250);
    m_speechTimer.AutoReset = true;
    m_speechTimer.Elapsed += (sender, e) => {
      CheckRecording();
    };
    m_speechTimer.Start();
  }
}
```

The user is explicitly asked for the speech recognition authorization; this is dealt in the **`RequestAuthorization()`** method.

We hook everything up with the parser by using the following statement:

```
ParserFunction.RegisterFunction("VoiceRecognition", new VoiceFunction());
```

The implementation of the **`VoiceFunction`** class is in Code Listing 46. Note that the speech recognition functionality is available starting from iOS 10.0, so we have a corresponding check there. This check can be added to the CSCS code as well.

Code Listing 46: The Implementation of the VoiceFunction Class

```
public class VoiceFunction : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
```


www.dbooks.org

```
  {
    List<Variable> args = script.GetFunctionArgs();
    Utils.CheckArgs(args.Count, 1, m_name);

    string strAction = args[0].AsString();
    STT.Voice = Utils.GetSafeString(args, 1, STT.Voice).Replace('_', '-');

    bool speechEnabled = UIDevice.CurrentDevice.CheckSystemVersion(10, 0);
    if (!speechEnabled) {
      UIVariable.GetAction(strAction, "\"" +
       string.Format(
"Speech recognition requires iOS 10.0 or higher. You have iOS {0}",
                    UIDevice.CurrentDevice.SystemVersion) + "\"", "");
      return Variable.EmptyInstance;
    }
    UIViewController controller = AppDelegate.GetCurrentController();
    m_speech = new STT(controller);
    m_speech.OnSpeechError += (errorStr) => {
      controller.InvokeOnMainThread(() => {
        UIVariable.GetAction(strAction, "\"" + errorStr + "\"", "");
      });
    };
    m_speech.OnSpeechOK += (recognized) => {
      controller.InvokeOnMainThread(() => {
        UIVariable.GetAction(strAction, "", "\"" + recognized + "\"");
      });
    };

    m_speech.StartRecording(STT.Voice);
    return Variable.EmptyInstance;
  }
}
```

An example of text-to-speech and speech recognition

Code Listing 47 contains a CSCS example for both text-to-speech and speech recognition.

Code Listing 47: Text-to-Speech and Speech Recognition in CSCS

```
AutoScale();
SetBackground("scream");
voice = "en-US";

locButtonTalk = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, -10);
AddButton(locButtonTalk, "buttonTalk", "", 200, 200);
AddAction(buttonTalk, "talk_click");
SetImage(buttonTalk, "microphone");
```

```
function talk_click(sender, arg)
{
  ShowToast("Start talking...", 5, "white", "black");
  VoiceRecognition("voice_recog", voice);
}
function voice_recog(errorStatus, recognized)
{
  if (errorStatus != "") {
    AlertDialog("CSCS", "Error: " + errorStatus);
  } else {
    ShowToast("Recognized: " + recognized, 5, "blue", "green");
    Speak("You said: " + recognized, voice);
  }
}
```

The **`ShowToast()`** CSCS function shows a native Toast on Android, and a custom made Toast on iOS (very similar to the Android Toast). The **`AlertDialog()`** CSCS function calls corresponding native dialogs on iOS and Android.

The result of running the CSCS code of Code Listing 47 is shown in Figure 25. On iOS, we have a successful result of recognizing the “Hello, World!” phrase, so that a Toast is shown and the recognized phrase is spoken by the text-to-speech feature.

On Android, the speech recognition fails—it’s not so straightforward to set up the microphone on Android simulators. On the real devices, everything works fine.

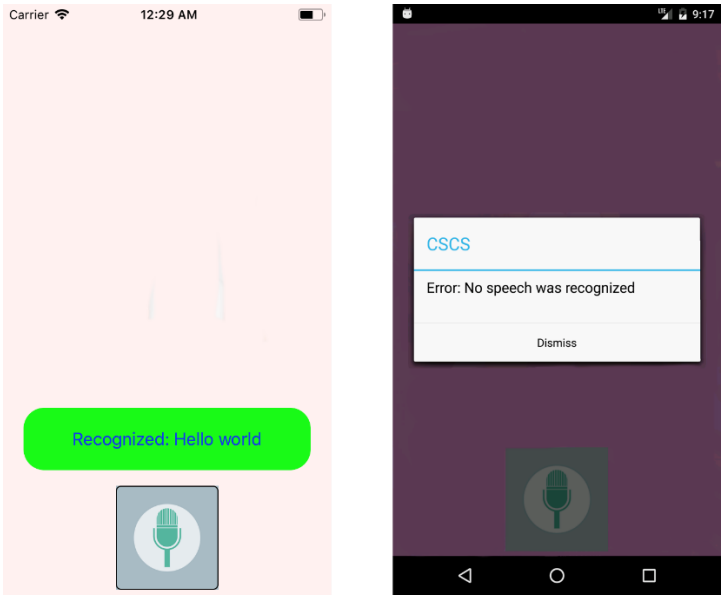


Figure 25: Running Speech Example on iOS and Android

www.dbooks.org

Summary

In this chapter, we saw how to add a non-GUI related functionality to CSCS. The principle remains the same: implement a class deriving from the **`ParserFunction`** class, and override its **`Evaluate()`** method.

This method serves as a wrapper method for all of the functionality that you want to add to CSCS. Don't forget to register the newly created class with the CSCS parser.

In the next chapter, we are going to learn how to add yet another useful feature to your app: In-App Purchases on iOS, and In-App Billing on Android.

Chapter 8 In-App Purchases in CSCS

"A study of economics usually reveals that the best time to buy anything is last year."
Marty Allen

In this chapter, we are going to see how to get our happy app users to pay us (in previous chapters, we saw how to make them happy). This concept is called In-App Purchase in the iOS world, and In-App Billing in the Android world.

In comparison to other mobile topics, setup and testing are less straightforward, and this is where the most time is spent. We are going to start with the setup.

Setting up In-App Purchases for iOS

First, you need to create your app in iTunes Connect. Then, you create your app purchases in iTunes Connect by choosing **Select Your App** > **Features** > **In-App Purchases**.

One of the common uses of an in-app purchase is to create a free app with some advertisements in it, and make the user pay to remove those advertisements.

In the next chapter, we are going to see how to add advertisements to an app. See Figure 26 for an example of adding an ad removal item in iTunes Connect.

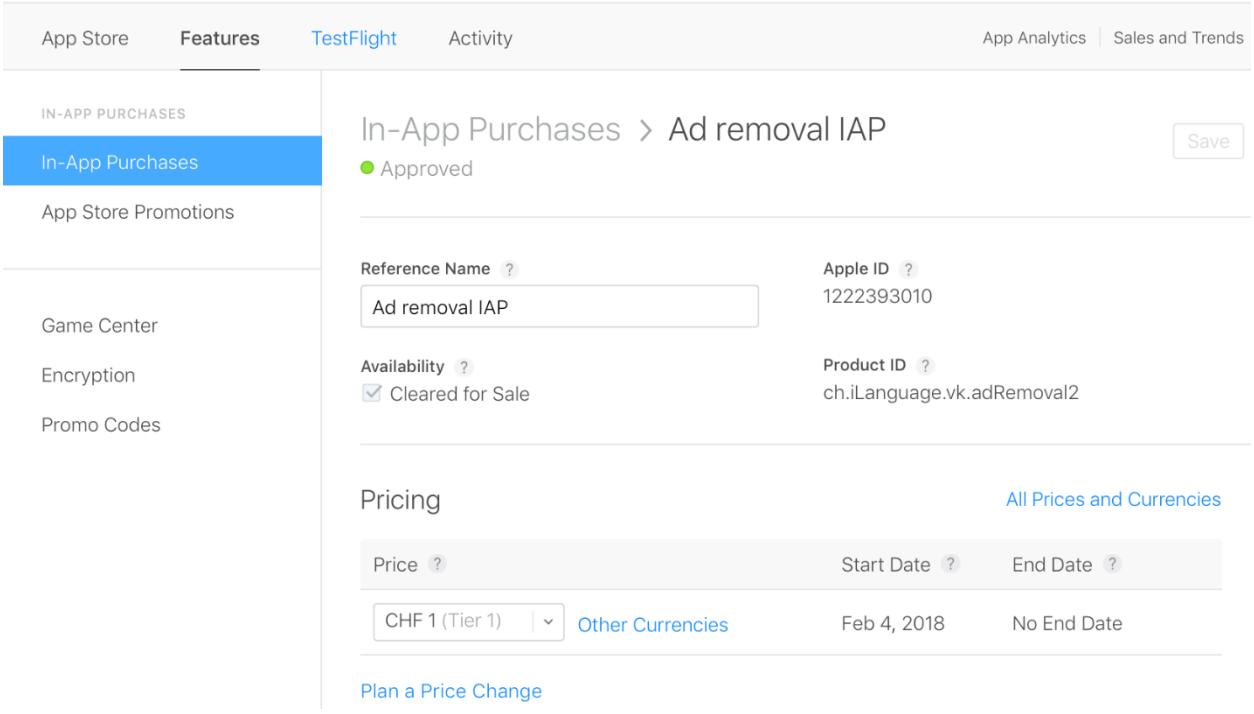


Figure 26: Setting Up In-App Purchases for iOS

www.dbooks.org

Next, you need to create test users for testing the In-App Purchase functionality. This is done in iTunes Connect by selecting **Users and Roles** > **Sandbox Testers**.

You will use one of the Sandbox Test accounts when testing the In-App Purchase functionality. Note that you must sign off in **Settings** > **iTunes & App Store** on your iOS device (do not sign-in any test account there; you will do this later on directly from the app).

Last, you need to add the In-App Purchase entitlement to your project. Open the **Entitlement.plist** file in Visual Studio, and make sure that the **Enable In-App Purchases** option is enabled.

If everything is set up correctly, you should see the enabled **In-App Purchase** check here.

Setting up In-App Billing for Android

To add different in-app products, go to **Google Play Console** > **Select Your App** > **Store presence** > **In-app products**. Check out an example of adding an ad removal item in Figure 27.

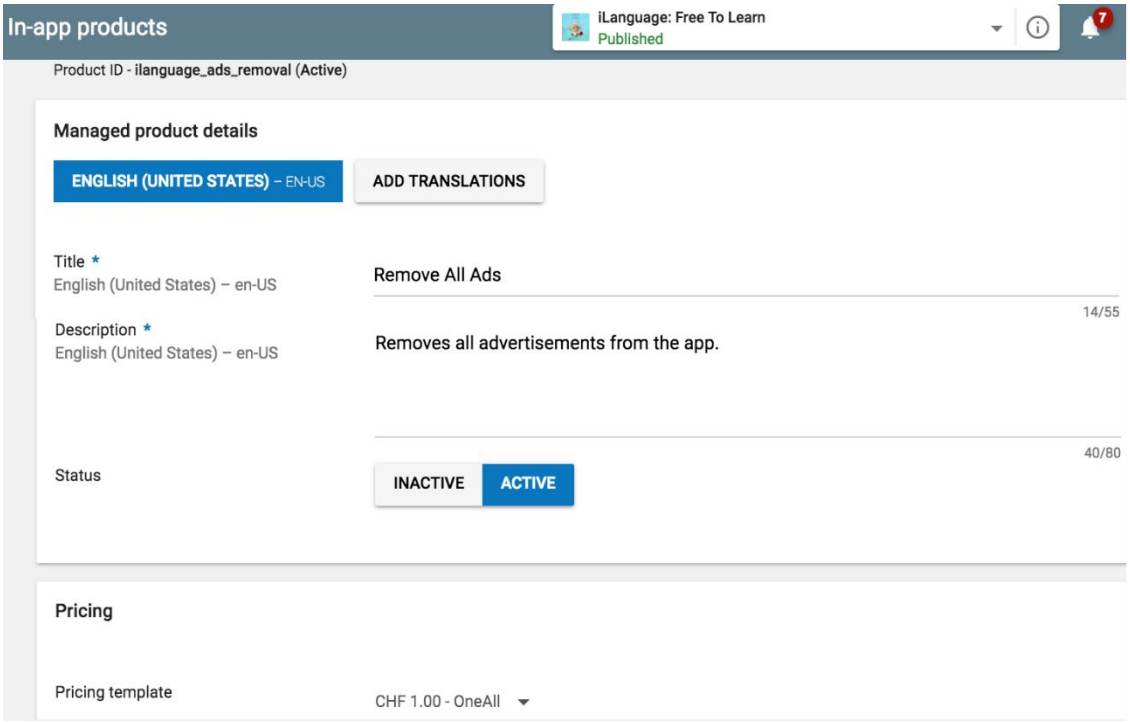


Figure 27: Setting Up in-App Billing for Android

Note that before testing the In-App Billing functionality, you have to upload an Alpha or a Beta APK to **Google Play Console** > **Select Your App** > **Release management** > **App releases**.

You also need to add test accounts for your app in **Google Play Console** > **Settings** > **Manage testers**, where you create a list of testers for your app.

For example, you can add testers to the Beta release in **Google Play Console** > **Select Your App** > **Release management** > **App releases** > **Manage Beta** > **Manage testers**. Make sure that you sign in on your Android device as one of these testers.

Also note that the following property must be added to your **AndroidManifest.xml** file:

```
<uses-permission android:name="com.android.vending.BILLING"/>
```

It is not possible to test In-App Billing using an emulator; you need a real device.



Tip: Enabling an Android device for development and USB debugging is tricky: go to** Setttings > System > About phone > Build number**, and tap seven times on the Build number. Then, Developer options become visible. But you are not done yet: now you can enable USB debugging in the Developer options—it is still disabled by default.

Implementation

For the implementation, I decided to use the Plugin.InAppBilling NuGet by James Montemagno from Xamarin. It hides quite a few of the tedious implementation details, making the C# code short and concise.

Using CSCS as a wrapper, the code is even more short and concise—we are going to see it in the next section.

There is good documentation on this plug-in, so I am going to give just an example of its usage in Code Listing 48, which extracts all available In-App Purchase items.

Code Listing 48: Extracting All the In-App Purchase Items Using the InAppBilling Plug-in

```
public async static Task GetInventory(string productId = "")
{
  if (!string.IsNullOrEmpty(productId) &&
      m_id2Description.ContainsKey(productId)) {
    return;
  }
  var billing = CrossInAppBilling.Current;
  try {
    var connected = await billing.ConnectAsync();
    if (!connected) {
      ConnectionFailed();
      return;
    }
    var items = await billing.GetProductInfoAsync(ItemType.InAppPurchase,
                                m_availableProducts.ToArray());
```


www.dbooks.org

```
    if (items == null) {
      return;
    }
    foreach (var item in items) {
      m_id2Description[item.ProductId] = item.Name;
    }
  } catch (Exception ex) {
    OnIAPError?.Invoke(ex.Message);
  } finally {
    await billing.DisconnectAsync();
  }
}
```

The usage in CSCS is even more straightforward: there are just three functions, which are registered with the parser as follows:

```
ParserFunction.RegisterFunction("Purchase", new PurchaseFunction());
ParserFunction.RegisterFunction("Restore", new RestoreFunction());
ParserFunction.RegisterFunction("ProductIdDescription",
                                new ProductIdDescriptionFunction());
```

PurchaseFunction, **RestoreFunction**, and **ProductIdDescriptionFunction** are just thin wrappers over the **InAppBilling** plug-in.

The **ProductIdDescription** returns the user-friendly In-App Purchase name—it is cached into **the m_id2Description dictionary** in the **GetInventory()** method shown in Code Listing 48.

The signatures of the three CSCS functions are:

```
Restore("restore_callback", productId1, productId2, ...);
Purchase("purchase_callback", productId);
ProductIdDescription(productId);
```

We are going to see how to use them in the next section.

CSCS example

Code Listing 49 contains the CSCS code for both restoring previously made purchases, and making a new purchase.

Note: The product IDs in Code Listing 49 must exactly match the product IDs in Figure 26 and Figure 27.

Code Listing 49: Restoring and Purchasing in CSCS

```
if (_IOS_) {
  productIdAds = "ch.iLanguage.vk.adRemoval2";
} elif (_ANDROID_) {
  productIdAds = "ilanguage_ads_removal";
}

locRestore = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", -80,-2);
AddButton(locRestore, "buttonRestore", "Restore", 130,100);
AddAction(buttonRestore, "restore");

locPurchase = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 80,-2);
AddButton(locPurchase,"buttonPurchase", "Purchase", 130,100);
AddAction(buttonPurchase, "purchase_ads");

function restore(sender, arg)
{
  ShowToast(Localize("Please wait..."), 10, "blue");
  Restore("on_restore", productIdAds);
}
function purchase_ads(sender, arg)
{
  ShowToast(Localize("Please wait..."), 10, "blue");
  Purchase("on_purchase", productIdAds);
}

function on_restore(error, productIds)
{
  if (error != "") {
    ShowToast(Localize("Error Restoring:") + " " + error, 8, "red");
    return;
  }
  if (productIds == "") {
    ShowToast(Localize("Nothing to restore"), 8, "red");
    return;
  }

  restoredIds = tokenize(productIds, ",");
  restoredStr = " ";
  for (productId : restoredIds) {
    description = ProductIdDescription(productId);
```


www.dbooks.org

```
    if (restoredStr != " ") {
      restoredStr += ", ";
    }
    restoredStr += description;
  }
  ShowToast(Localize("Restored:") + restoredStr, 6, "blue");
}

function on_purchase(error, productId)
{
  if (error != "") {
    ShowToast(Localize("Error Purchasing:") + " " + error, 6, "red");
    return;
  }

  description = ProductIdDescription(productId);
  ShowToast(Localize("Purchased:") + " " + description, 6, "blue");
}
```



Note: The "_IOS_" and "_ANDROID_" CSCS functions return `true` for their corresponding platforms.

The results of running the purchasing part are shown in Figure 28 for real iOS and Android devices. Note that the purchases on both devices are test purchases because of the way we set up the test users in the beginning of this chapter.

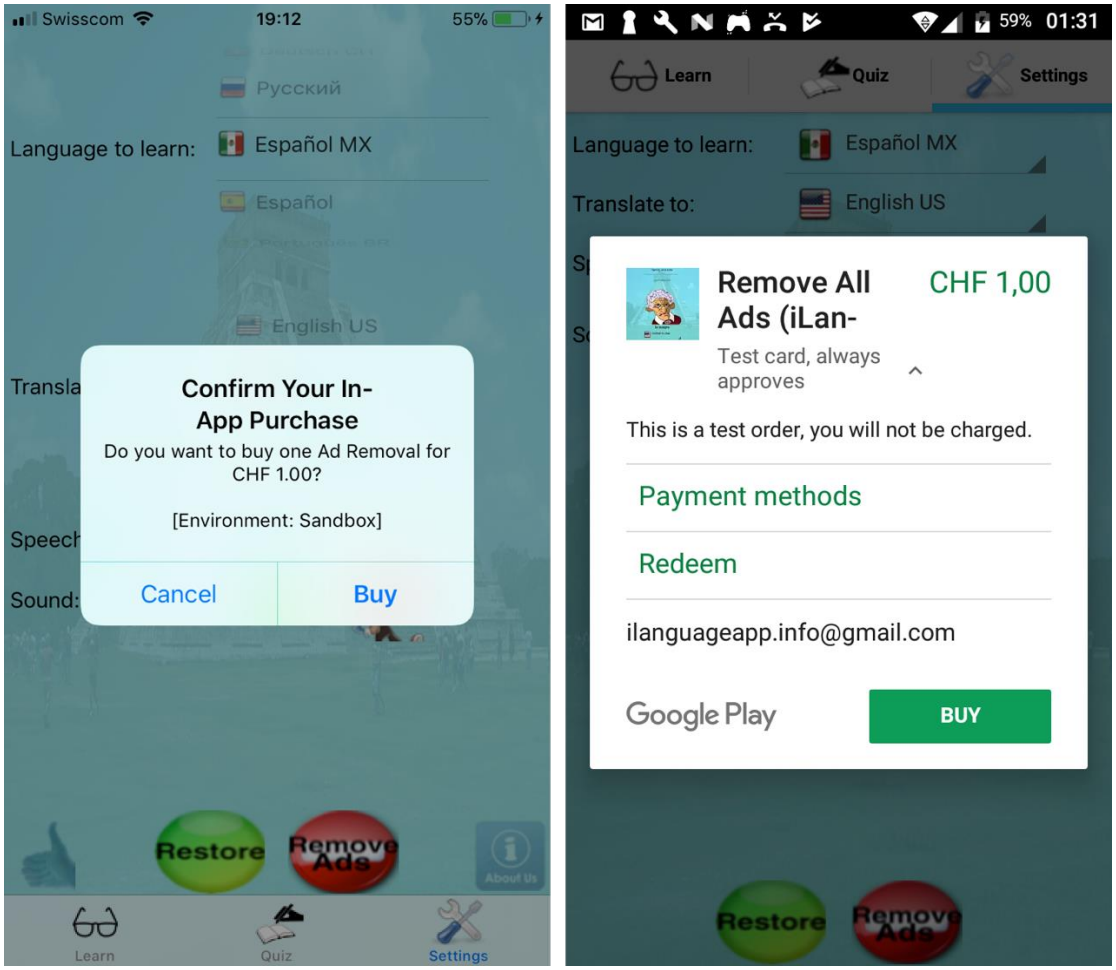


Figure 28: A Test Purchase on iOS and Android

Summary

Implementing the In-App Purchase and In-App Billing functionality is very time consuming because there are many things that can go wrong.

Using CSCS, there are just three functions. However, you still need to give a lot of attention to the exact product IDs, app names, permissions, and the test users being used. On Android, it's also important to upload an Alpha or Beta APK before testing.

In the next chapter, we are going to see how to add advertisements to an app and look at debugging, localization, user defaults and settings, scheduling events, and some other topics.

www.dbooks.org

Chapter 9 Adding Mobile Advertising and Advanced Topics

"Stopping advertising to save money is like stopping your watch to save time."
Henry Ford

Adding Google AdMob to CSCS

Adding an advertisement to be displayed in your app is a good source of extra income—not only from the ads themselves, but also from the In-App Purchase items set up to remove ads if they annoy users too much. We saw how to do that in the previous chapter.

There are a few providers of the advertisement content for mobile devices. I chose Google AdMob because it is one of the most popular ones, and it also has good documentation and a support base.

To plug the AdMob Framework into the project, on iOS I used the Xamarin.Firebase.iOS.AdMob NuGet, and on Android, I used Xamarin.GooglePlayServices.Ads.Lite NuGet. You can check out the implementation details in the accompanying source code.

Code Listing 50 contains the CSCS code showing some of the AdMob banners. It also creates a button that when clicked, causes an interstitial ad to be displayed. Interstitial ads are the full-screen ads monopolizing the whole device screen (and are by far the most irritating ones).

Code Listing 50: Adding AdMob Banners in CSCS

```
AutoScale();
SetBackgroundColor("gainsboro");

if (_IOS_) {
  appId          = "ca-app-pub-5365456490909700~2128563695";
  interstitialId = "ca-app-pub-5365456490909700/7192496494";
  bannerId       = "ca-app-pub-5365456490909700/3326095292";
} elif (_ANDROID_) {
  appId          = "ca-app-pub-5365456490909700~8197798895";
  interstitialId = "ca-app-pub-5365456490909700/2151265291";
  bannerId       = "ca-app-pub-5365456490909700/8058198097";
}

InitAds(appId, interstitialId, bannerId);

locClickme = GetLocation("ROOT", "CENTER", "ROOT", "CENTER", 0, 20);
AddButton(locClickme, "buttonClickme", "Interstitial", 200, 80);
AddAction(buttonClickme,  "clickme_click");
```

```
function clickme_click(sender, arg)
{
  ShowInterstitial();
}

locAd1 = GetLocation("ROOT", "LEFT", "ROOT", "TOP", 0, 0);
AddAdMobBanner(locAd1, "adMob1", "MediumRectangle", DisplayWidth, 500);

locAd2 = GetLocation("ROOT", "LEFT", buttonClickme, "BOTTOM", 0, 20);
AddAdMobBanner(locAd2, "adMob2", "FullBanner", 1024, 160);

locAd3 = GetLocation("ROOT", "LEFT", adMob2, "BOTTOM", 0, 10);
AddAdMobBanner(locAd3, "adMob3", "LargeBanner", DisplayWidth, 200);

locAd4 = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, 0);
AddAdMobBanner(locAd4, "adMob4", "Banner", DisplayWidth, 100);
```

Code Listing 50 contains the production banner and interstitial IDs. You get these IDs when you register your app with Google AdMob. For testing, you can use the following IDs; they are the same for iOS and Android:

```
bannerId          = "ca-app-pub-3940256099942544/6300978111";
interstitialId    = "ca-app-pub-3940256099942544/1033173712";
```

The result of running Code Listing 50 with the testing IDs is shown in Figure 29.

www.dbooks.org

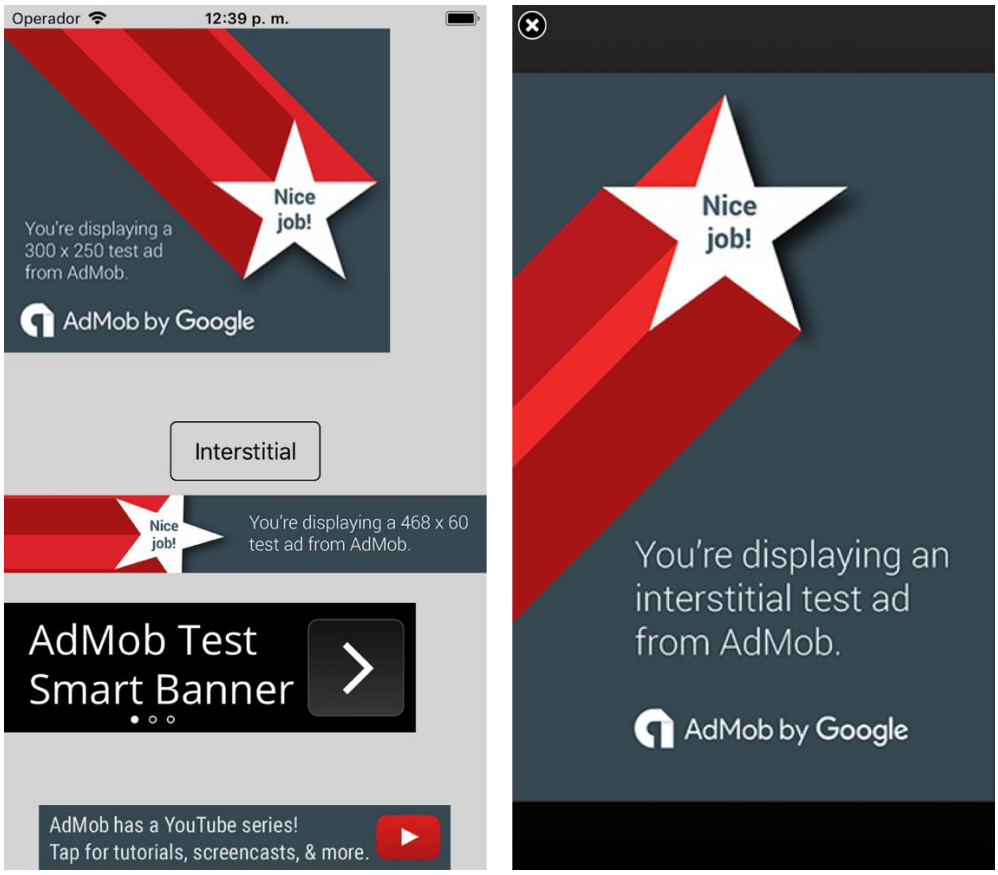


Figure 29: Google AdMob with Different Banners (left) and an Interstitial Ad (right)

Debugging and unit testing

To test and debug the CSCS code, you can use the Visual Studio Debugger for all of the CSCS functions implemented in C#.

For functions implemented in CSCS, the easiest is to use the **`WriteConsole()`** CSCS method to see where the problem occurs and print out the current values of the variables.

I have also implemented unit testing. The code is in the **unitTest.cscs** file. As soon as I add a new feature to CSCS, or change an existing feature, I add or modify a unit test in **unitTest.cscs**. To run unit tests, I just enable the following line in **start.cscs**:

```
ImportFile("unitTest.cscs");
```

The output from the unit tests is shown in a scrollable TextView. I create it as follows:

```
locView = GetLocation("ROOT", "CENTER", "ROOT", "TOP");
AddTextView(locView, "textView", "", DisplayWidth - 30, DisplayHeight - 60);
```

The main CSCS function for unit testing is **`test()`**. Its implementation is shown in Code Listing 51.

Code Listing 51: Implementation of the CSCS Test Function

```
function test(x, expected)
{
  if (x == expected) {
    AddText(textView, string(x) + " as expected. OK", "green");
    return;
  }
  if (type(expected) != "NUMBER") {
    AddText(textView, "[" + x + "] but expected [" + expected +
                      "]. ERROR", "red");
    return;
  }

  epsilon = 0.000001;
  if ((expected == 0 && abs(x) <= epsilon) ||
       abs((x - expected) / expected) <= epsilon) {
    AddText(textView, "[" + x + "] within epsilon to [" + expected +
                      "]. almost OK", "gray");
  } else {
    diff = expected - x;
    AddText(textView, "[" + x + "] but expected [" + expected +
                      "]. diff=" + diff + ". ERROR", "red");
  }
}
```

Note that the third argument to the **`AddText()`** function is the text color. Using the **`test()`** function, it's easy to write various CSCS unit tests.

Code Listing 52 shows some of the unit tests, and the Figure 30 shows the result of running them on an iPhone simulator.

Code Listing 52: A Fragment of the CSCS Unit Tests

```
AddText(textView, "Testing strings...");
txt = "lu";
txt += txt + Substring(txt, 0, 1) + "_" + 1;
test(txt, "lulul_1");
bb = "abc_blah;";
c = Substring(bb, 4, 3);
test(c, "bla");
ind = StrIndexOf(bb, "bla");
test(ind, 4);
between = StrBetween(bb, "_", ";");
test(between, "bla");
between = StrUpper(StrReplace(between, "a", "aa"));
test(between, "BLAAH");

AddText(textView, "Testing numbers...");
```


www.dbooks.org

```
a=(-3+2*9)-(10-15);
test(a, 20);
test((a++)-(--a)-a--, -20);
test(a, 19);
test(((16-3)-3)+15/2*5, 47.5);
test(1-2-3-(4-(5-(6-7)))-pow(2,3*exp(14-7*2)), -10);
test(sin(pi/2), 1);

x = 2.0E+15 + 3e+15 - 1.0e15;
test(x, 4e+15);
a=1; c=0; b=5;
test(a||c, 1);
test(c&&b, 0);

AddText(textView, "Testing arrays and maps...");
a[1][2]=22;
a[5][3]=15;
a[1][2]-=100;
a[5][3]+=100;
test(a[1][2], -78);
test(a[5][3], 115);

arr[2] = 10; arr[1] = "str";
test(type(arr),    "ARRAY");
test(type(arr[0]), "NONE");
test(type(arr[1]), "STRING");
test(type(arr[2]), "NUMBER");

x["bla"]["blu"]=113;
test(contains (x["bla"], "blu"), 1);
test(contains (x["bla"], "bla"), 0);
x["blabla"]["blablu"]=125;
test(--x["bla"]["blu"] + x["blabla"]["blablu"]--, 237);

b[5][3][5][3]=15;
b[5][3][5][3]*=1000;
test(b[5][3][5][3], 15000);
test(size(b), 6);
test(size(b[5]), 4);
test(size(b[5][3]), 6);
test(size(b[5][3][5]), 4);
test(size(b[5][3][5][3]), 5);
```

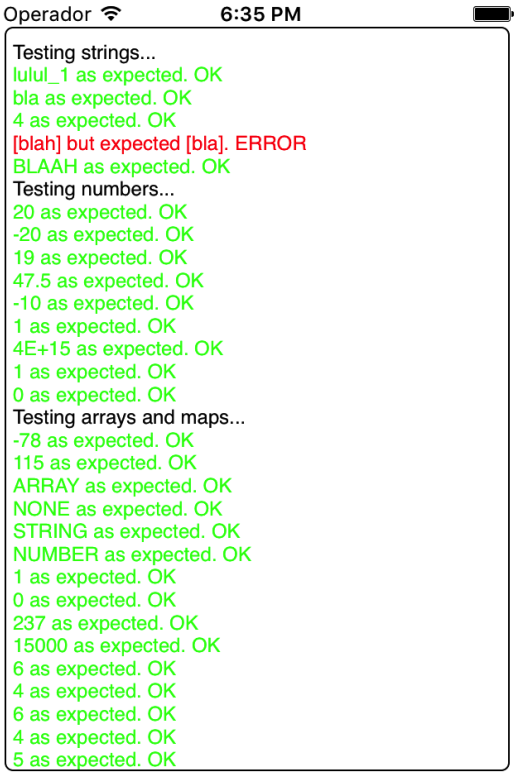


Figure 30: Unit Testing TextView in iOS Simulator

Localization

We already saw an example of the CSCS **`Localize()`** function in Code Listing 49. It returns a string, localized in the device locale. A general signature of this function is:

```
Localize(text, languageCode = AppLocale);
```

You can specify any other language in the second **`Localize()`** argument as well.

Other localization-specific CSCS functions are:

```
GetDeviceLocale();
SetAppLocale(languageCode);
```

`GetDeviceLocale()` returns the device language, and **`SetAppLocale()`** sets the language. It does not change the global device language, just the language that is used by the app.

To make it work, the translations must be manually added to the resource files.
On iOS, the translations are added to the **languageCode.lproj/Localizable.strings** files (the folder names are, **de.lproj**, **fr.lproj**, etc.), and on Android, the translations are added to the **values-languageCode/Strings.xml** files.

www.dbooks.org

There is no need to write files for both projects: there are a few tools that convert from one localization format to another. On the other hand, Xamarin.Forms uses the RESX resource files for multiple platforms.

Settings or user defaults

The interface to access and modify user preferences is **NSUserDefaults** on iOS, and **ISharedPreferences** on Android. You use it to cache the user preference data, keep best user scores, and so on, so that they are persistent between different program runs.

The CSCS functions shield the iOS and Android differences:

```
SetSetting(aKey, aVariable, type = "string");
GetSetting(aKey, type = "string", defaultValue = "" or 0);
```

The type can be any of the following: **string**, **int**, **long**, **bool**, **float**, or **double**. If no type is specified, a string is assumed. On Android, **double** is treated as **float**, and on iOS, **long** is treated as **int**.

Here's an example of using settings in CSCS:

```
SetSetting("myKey", a, "int");
a = GetSetting("myKey", "int");
```

Swipe, long click, and drag and drop

We saw earlier that the **AddAction(widget, callback)** CSCS function connects a widget to a function that is called when the user clicks on a widget.

There are also other types of user events that you can connect with some actions in CSCS. You can connect an event of a user doing a swipe with an action as follows:

```
AddSwipe(widget, type, callback);
```

The type parameter can be any of the following: **Left**, **Right**, **Up**, or **Down**. For example:

```
AddSwipe(widget, "Left",  "swipe");
AddSwipe(widget, "Right", "swipe");
```

The **swipe** function has the following signature:

```
function swipe(sender, eventName)
```

The **eventName** variable is a string equal to the type parameter that triggered the event. It can be **Left**, **Right**, **Up**, or **Down** so that you know what type of swipe occurred.

There are also other events that you can connect to an action. For example, the following events are used when a widget has been clicked and held or dragged and dropped:

```
AddLongClick(widget, callback);
AddDragAndDrop(widget, callback);
```

The callback of the **AddDragAndDrop** function has the following signature:

```
function callback(sender, receiving_widget);
```

The **receiving_widget** parameter specifies a widget (or a comma-separated list of widgets), on which the original widget was dropped. It will be empty if there is no such widget.

How is it implemented? Every time a widget is added (or moved), we store its location. We also have a mapping from every tab and a parent view to the list of widgets that were placed there. Then we just look for the widgets that are present in the location where the original widget was dropped.

Scheduling events

One handy feature is the ability to schedule an event to be executed in the future. The CSCS functions used to schedule and to cancel an event are the following:

```
Schedule(timeoutMs, callback, sender = "", timerId = "", autoreset = false);
CancelSchedule(timerId);
```

If the **autoreset** parameter is set to **false**, then the **callback** function will be invoked only once; otherwise it will be invoked periodically, unless the **CancelSchedule()** function is called.

The event will be executed on the main GUI thread. This is done by invoking **UIViewController().InvokeOnMainThread()** on iOS, and **MainActivity.RunOnUiThread()** on Android.

For instance, I used scheduling when implementing an example of a Syncfusion Digital Gauge:

```
Schedule(1000, "timer_timeout", "", "timerId", true);
function timer_timeout(sender, arg)
{
  SetValue(DigitalGauge, "value", Now("HH:mm:ss"));
}
```

The **timer_timeout** function is invoked every second (every 1,000 milliseconds). It updates the time shown by **DigitalGauge**.

www.dbooks.org

Asynchronous programming in CSCS

We saw some examples of asynchronous programming in CSCS in the previous chapter, when we asynchronously requested purchasing and restoring In-App Purchase items. The C# implementation there used the **async** and **await** keywords.

In CSCS, asynchronous programming is done by supplying a callback function. Note that the callback function will be scheduled on the main GUI thread (see what this means in the previous section). So the callback will not be triggered in the "middle" of something, but only after the current operation on the main GUI thread is complete.

Calling native code from CSCS

By "native," I mean C#. You might have a lot of existing C# code that you do not want to rewrite in CSCS. There is no need—it is possible to call C# code from CSCS and get the results back.

Code Listing 53 contains code that is used to compile and cache native calls for subsequent access. Only the first call will take time—because of reflection, compilation, and caching. The time spent on the subsequent calls will be roughly the same as when calling C# code directly.

Code Listing 53: Implementation of Calling C# Code from CSCS

```
static Dictionary<string, Func<string, string>> m_compiledCode =
   new Dictionary<string, Func<string, string>>();

public static Variable InvokeCall(Type type, string methodName,
              string paramName, string paramValue, object master = null)
{
  string key = type + "_" + methodName + "_" + paramName;
  Func<string, string> func = null;

  // Cache compiled function:
  if (!m_compiledCode.TryGetValue(key, out func)) {
    MethodInfo methodInfo = type.GetMethod(methodName,
                                new Type[] { typeof(string) });
    ParameterExpression param = Expression.Parameter(typeof(string),
                                                     paramName);
    MethodCallExpression methodCall = master == null ?
          Expression.Call(methodInfo, param) :
          Expression.Call(Expression.Constant(master), methodInfo, param);
    Expression<Func<string, string>> lambda =
        Expression.Lambda<Func<string, string>>(methodCall,
                  new ParameterExpression[] { param });
    func = lambda.Compile();
    m_compiledCode[key] = func;
  }

  string result = func(paramValue);
```

```
  return new Variable(result);
}
```

To register a CSCS function to call the native code, we use the following statement:

```
ParserFunction.RegisterFunction("CallNative", new InvokeNativeFunction());
```

Here, **InvokeNativeFunction** is a thin wrapper over Code Listing 53. This is the main call it makes:

```
var result = Utils.InvokeCall(typeof(Statics),
                              methodName, paramName, paramValue);
```

The CSCS function signature is the following:

```
CallNative(methodName, argumentName, argumentValue);
```

Here is an example of calling a native function from CSCS and getting a result back:

```
clicks++;
title = CallNative("ProcessClick", "arg", clicks);
```

All the native functions are implemented in the **Statics** class, and here is the implementation of the **ProcessClick** method, which returns back to CSCS a string, created in C# code:

```
public static string ProcessClick(string arg)
{
  var now = DateTime.Now.ToString("T");
  return "Clicks: " + arg + "\n" + now;
}
```

Using this feature, you can keep using any existing C# code, for instance, for database access.

Conclusion

CSCS is an evolving language. I am constantly adding new features to it, so check periodically the GitHub repository, where the source code lives.

As we have seen, you can add to CSCS anything that can be implemented in Xamarin.iOS and Xamarin.Android. Since they can both implement any iOS or Android feature, you can add any native platform feature to CSCS as well.

I am looking forward to your feedback and seeing how you have modified CSCS, what you have added, and what apps you have created.

www.dbooks.org

Appendix

"I am a brain, Watson. The rest of me is a mere appendix."
Arthur Conan Doyle

Currently implemented CSCS functions

Table 1: CSCS Functions for Adding Widgets

CSCS Function	Description
GetLocation(referenceX, relationX, referenceY, relationY, marginX=0, marginY=0, parentView=null)	Creates a location relative to the other widgets' **referenceX** horizontally, and **referenceY** vertically.
AddWidget(type, widgetName, location, initString, width, height)	A generic function to add a widget at a given location with the initialization string, width, and height. Specializations for different widgets follow.
AddView(widgetName, location, initString, width, height)	Adds a **UIView** on iOS and a **RelativeLayout** on Android to place additional widgets inside of it.
AddButton(widgetName, location, initString, width, height)	Adds a **UIButton** on iOS and a **Button** on Android with the title string from the **initString**.
AddLabel(widgetName, location, initString, width, height)	Adds a **UILabel** on iOS and a **TextView** on Android with a given text string from the **initString**.
AddTextView(widgetName, location, initString, width, height)	Adds a **UITextView** on iOS and a **TextEdit** on Android with a given text string from the **initString**.
AddTextEdit(widgetName, location, initString, width, height)	Adds a **UITextField** on iOS and a **TextEdit** on Android with a given text hint (or a placeholder in iOS terms) from the **initString**.
AddImageView(widgetName, location, initString, width, height)	Adds a **UIImageView** on iOS and an **ImageView** on Android with image filename from the **initString**.
AddSwitch(widgetName, location, initString, width, height)	Adds a **UISwitch** on iOS and a **Switch** on Android. The **initString** argument must be either a 0 or a 1.

CSCS Function	Description
AddSlider(widgetName, location, initString, width, height)	Adds a **UISlider** on iOS and a **SeekBar** on Android. The **initString** argument must have format **fromNumber:toNumber** (for example, **0:100**).
AddStepper(widgetName, location, initString, width, height)	Adds a **UIStepper** on iOS and two buttons on Android. The **initString** argument must have format **fromNumber:toNumber** (for example, **0:100**).
AddStepperLeft(widgetName, location, initString, width, height)	Adds a stepper as in **AddStepper**, but also with the label with the stepper current value on the left.
AddStepperRight(widgetName, location, initString, width, height)	Adds a stepper as in **AddStepper**, but also with the label with the stepper current value on the right.
AddCombobox(widgetName, location, initString, width, height)	Adds a custom **Combobox** on iOS and a **Spinner** on Android. Used together with **AddWidgetData** and **AddWidgetImages** functions.
AddTypePickerView(widgetName, location, initString, width, height)	Adds a **UITypePicker** on iOS and a **NumberPicker** on Android. The **initString** argument is not used.
AddPickerView(widgetName, location, initString, width, height)	Adds a **UITypePicker** on iOS and a **Spinner** on Android. The **initString** argument is not used.
AddSegmentedControl(widgetName, location, initString, width, height)	Adds a **UISegmentedControl** on iOS and either a Switch (for two segments) or buttons on Android. **InitString** specifies number of segments.
AddListView (widgetName, location, initString, width, height)	Adds a **UITableView** on iOS and a **ListView** on Android. It can be populated with **AddWidgetData** and **AddWidgetImages** functions.

Table 2: CSCS UI Related Functions

CSCS Function	Description
AddWidgetData(widgetName, data, selectAction)	Adds a list of strings to the widget. Also registers the **selectAction** CSCS function to be called when an entry is selected.

www.dbooks.org

CSCS Function	Description
AddWidgetImages(widgetName, images)	Adds a list of images (specified as image filenames) to the widget.
AddAction(widgetName, callback)	Adds a widget action on clicking event. If the **widgetName** is **ROOT**, then adds a global action (**OnTabSelected** for a Tab app, for example).
AutoScale(scale=1.0)	Automatically scales all the widgets according to the display resolution. The widgets can be scaled individually as well.
SetBaseWidth(width)	Sets the base device width to be used by the **AutoScale** function. The default value is 640.
Relative(size)	Returns passed parameter adjusted to the device width, depending
AddLongClick(widgetName, callback)	Adds a widget action when the user presses and holds for a few seconds.
AddSwipe(widgetName, callback)	Adds detection of swiping left, right, up, and down. Calls **callback**, passing as an argument which type of swipe event happened.
AddDragAndDrop(widgetName, callback)	Adds drag and drop functionality. Moves widget till the finger is lifted. The callback function gets a widget (or an empty string) where it was dropped.
AddBorder(widgetName, width, corner, color="black")	Adds a border around a widget. If the width is 0, removes existing border.
AddTab(tabName, activeImage, inactiveImage)	Adds a tab to the app with corresponding active and inactive images.
AddOrSelectTab(tabName, activeImage, inactiveImage)	Adds a tab to the app like **AddTab()** if it doesn't exist; otherwise, just selects it.
GetSelectedTab()	Returns index of the active tab.
SelectTab(index)	Programmatically activates a tab.
RemoveTabViews(index)	Removes all the views in a tab with a given index.
OnTabSelected(callback)	Calls a **callback** when a new tab is selected. Passes the numeric index of the new tab.

CSCS Function	Description
AlignText(widgetName, alignType)	Aligns text according to the **alignType**, which can be **left**, **right**, **top**, **bottom**, **center**, **justified**, **fill**, or **natural**.
GetText(widgetName)	Returns widget's text.
SetText(widgetName, text)	Sets text on the widget.
AddText(widgetName, text, color="black")	Appends text to the widget's text field. The color is used for the **TextViews**.
GetValue(widgetName)	Returns widget's value.
SetValue(widgetName, value)	Sets value to the widget.
SetImage(widgetName, imageFile)	Sets image on a widget.
SetBackground(imageFile)	Sets background image on the root view.
SetBackgroundColor(widgetName, color)	Sets background color on a widget. If the widget is **ROOT**, sets background color on the root view.
SetSize(widgetName, width, height)	Sets widget's height and width.
SetFont(widgetName, fontName, fontSize=0)	Sets widget's font on iOS or Typeface on Android.
SetFontSize(widgetName, fontSize)	Sets widget's font size.
SetFontColor(widgetName, fontColor)	Sets widget's font color.
SetBold(widgetName, fontSize=0)	Sets widget's text as bold.
SetItalic(widgetName, fontSize=0)	Sets widget's text as italic.
SetNormalFont(widgetName, fontSize=0)	Removes any widget's specifically set font properties, like **bold** or **italic**.
SetVisible(widgetName, visible=true)	Shows or hides a widget (or a view / layout).
MoveView(widgetName, x, y)	Moves a widget **x** pixels right and **y** pixels down (for negative values, moves widget left and up).
MoveViewTo(widgetName, x, y)	Moves a widget to a new position, having **x** and **y** as absolute coordinates on the screen.

www.dbooks.org

CSCS Function	Description
RemoveView(widgetName)	Removes a view or a widget from the layout.
RemoveAllViews()	Removes all views from the layout. It can be used on orientation change, when rebuilding the layout.
GetX(widgetName)	Returns widget's absolute X coordinate.
GetY(widgetName)	Returns widget's absolute Y coordinate.
SetStyle(style, type, orientation)	Sets app style (can be **tabs**, **navi**, or **page**).
ShowHideKeyboard(widgetName, show=true)	Shows or hides a software keyboard for a particular widget.
IsKeyboard(widgetName)	Returns **true** if the keyboard is visible for a particular widget, and **false** otherwise. The widget argument is not used on Android.

Table 3: CSCS Non-GUI Functions

CSCS Function	Description
CallNative(methodName, argName, argValue)	Calls a native C# method, having given parameters. The C# method has one string argument and returns a string.
GetRandom(limit,numberOfValues)	A pseudo-random number generator returning a list of generated numbers between 0 and limit (exclusive limit).
ShowToast(message, duration, bg_color, fg_color)	Shows a Toast on Android and a custom Toast implementation on iOS. The last two parameters are optional.
AlertDialog(type, title, message, button1, action1, button2, action2)	Shows an alert dialog to the user. The last three parameters are optional; the dialog is dismissed if there is no action.
SetupSpeech(option, value)	Sets one of speech options **(voice**, **speechRate**, **pitch**, **volume**, or **sound**).
Speak(phrase, voice, rate, pitch)	Text-To-Speech. The default voice is **en-US**. Speech rate and pitch are between 0 and 1.

CSCS Function	Description
VoiceRecognition(callback, voice)	Starts voice recognition, calling the callback on completion, passing the recognized phrase as an argument. The default voice is **en-US**.
StopVoiceRecognition()	Stops and cancels current voice recognition operation.
ReadFile(filename)	Reads file from the device assets directory. Returns a list of lines from that file.
Schedule(timeout, callback, timerId)	Schedules execution of the **callback** function after the timeout milliseconds on the main GUI thread.
CancelSchedule(timerId)	Cancels a job previously scheduled with the **timerId** by the **Schedule()** function.
GetSetting(settingName, type, defaultValue)	Gets value from the device settings. Type can be **float**, **double**, **int**, **long**, **string** or **bool**.
SetSetting(settingName, value, type)	Saves setting on the device so that later it can be retrieved by the **GetSetting()** function.
GetDeviceLocale()	Returns the language locale of the device.
SetAppLocale(localeName)	Sets the locale of the app. The expected format of the **localeName** is **en-US**, **es-MX**, etc.
Localize(text, language=AppLocale)	Localizes passed text to the current app locale (by default it's the device locale). If **language** is set, localizes to that language.
TranslateTabBar()	Translates the tab bar to the language set by **SetAppLocale()** or to the device language.
Restore(callback , productId1, productId2, …)	Checks if products have been purchased before in Apple App Store or Google Play Store.
Purchase(callback, productId)	Purchases a product, connecting to the Apple App Store or Google Play Store.
ProductIdDescription(productId)	Returns a product description, extracted from the Apple App Store or Google Play Store.
GetTrie(id, words)	Builds a trie form the passed list of words.

www.dbooks.org

CSCS Function	Description
SearchTrie(trie, text, maxWords=10)	Searches a trie returned from **GetTrie()** for a text.
ImportFile(filename)	Imports another CSCS source file.
OnOrientationChange(callback)	Calls the **callback** function when there's a widget orientation change, passing as a parameter the new orientation (**portrait** or **landscape**).
RegisterOrientationChange(on_portrait, on_landscape)	Registers the callback functions to be triggered when the device orientation changes.
Orientation	Current device orientation (**landscape** or **portrait**).
OnEnterBackground(callback)	A function to be called when the app is entering the background.
OpenURL(url)	Opens a URL in a browser.
WebRequest(url)	Submits a web request to a URL and returns the complete response from the server.
DisplayWidth	Returns the width of the display in pixels.
DisplayHeight	Returns the height of the display in pixels.
ANDROID	Returns **true** if and only if the current code is being executed on Android.
IOS	Returns **true** if and only if the current code is being executed on iOS.
_DEVICE_INFO_	Returns the smartphone information.
_VERSION_INFO_	Returns the smartphone OS info.
_VERSION_NUMBER_	Returns the smartphone OS number.
CompareVersions(version1, version2)	Compares **version1** and **version2**, returning **-1** if **version1** is earlier, **0** if same, and **1** otherwise.

Table 4: CSCS Extras: Google AdMob and Syncfusion

CSCS Function	Description
InitAds(appId, interstitialId, bannerId)	Inits the Google AdMob advertisement framework with the initialization parameters, previously requested at https://www.google.com/admob.
AddAdMobBanner(widgetName, location, bannerType)	Adds a **BannerView** on iOS and an **AdView** on Android for the AdMob. The **bannerType** can be either **SmartBanner**, **MediumRectangle**, **Banner**, **LargeBanner**, **FullBannner**, or **Leadeboard**.
ShowInterstitial()	Shows an AdMob interstitial ad.
AddSfQRBarcode(widgetName, location, initString, width, height)	Adds a Syncfusion QR Barcode to a given location.
AddSfCode39Barcode(widgetName, location, initString, width, height)	Adds a Syncfusion Barcode 39 to a given location.
AddSfCircularGauge(widgetName, location, initString, width, height)	Adds a Syncfusion Circular Gauge to a given location.
AddSfDigitalGauge(widgetName, location, initString, width, height)	Adds a Syncfusion Digital Gauge to a given location.
AddSfPicker(widgetName, location, initString, width, height)	Adds a Syncfusion Picker to a given location.
AddSfStepper(widgetName, location, initString, width, height	Adds a Syncfusion Circular Gauge to a given location.
AddSfBusyIndicator(widgetName, location, initString, width, height)	Adds a Syncfusion Busy Indicator to a given location.
AddSfSplineGraph(widgetName, location, initString, width, height)	Adds a Syncfusion Spline Graph to a given location.
AddSfDoughnutGraph(widgetName, location, initString, width, height)	Adds a Syncfusion Doughnut Graph to a given location.

www.dbooks.org

CSCS Function	Description
AddSfCalendar(widgetName, location, initString, width, height)	Adds a Syncfusion Calendar to a given location.
AddSfAppointment(calendar, subject, start, end, color, format)	Adds a Syncfusion Appointment to the calendar created in **AddSfCalendar()**. Default format: **"yyyy/MM/dd HH:mm"**.
AddSfImageEditor(widgetName, location, initString, width, height)	Adds a Syncfusion Image Editor to a given location.
StartSfImageEditor(imageEditor, init)	Starts the Image Editor created in **AddSfImageEdito**r. The **init** parameter is either **camera** or **library**.
SfExcelNew()	Creates and returns a new Excel document.
SfExcelOpen(filename)	Opens an existing Excel document.
SfExcelSet(excel, cell, option, value)	Sets an Excel option for a cell to the passed excel document.
SfAddExcelWorksheet(excel, sheet)	Adds a worksheet to the Excel document.
SfRenameExcelWorksheet(excel, sheet)	Renames a worksheet in the Excel document.
SfActivateExcelWorksheet(excel, sheet)	Activates a worksheet in the Excel document.
SfAddExcelChart(excel, range, title, top, bottom, left, right)	Adds an Excel chart using passed cells as chart data.
SfSaveExcel (excel, filename)	Saves Excel document to a file.
SfWordNew()	Creates and returns a new Word document.
SfWordOpen(filename)	Opens an existing Word document.
SfAddWordText(word, text)	Adds text to the current Word paragraph.
SfAddWordTextRange(word, text, font, fontSize, fontColor, bgColor)	Adds a text range to the Word document with different options.

CSCS Function	Description
SfAddWordImage(word, path, x, y, wrap, origH, origV, widthScale, heightScale)	Adds an image to the Word document.
SfAddWordTable(word, rows, cols, style)	Adds a table to the Word document.
SfAddWordParagraph(word, type, indent, row, col)	Adds a paragraph to the Word document.
SfApplyWordStyle(word, style, alignment)	Applies a style to the current Word document paragraph.
SfSaveWord(excel, filename)	Saves Word document to a file.
SfPdfNew()	Creates and returns a new Pdf document.
SfPdfOpen(filename)	Opens an existing PDF document.
SfSetPdfText(pdf, text, x, y, color)	Adds text starting at **x**,**y** in the PDF document.
SfSetPdfImage(pdf, image, x, y, w, h)	Adds an image to the PDF document.
SfSetPdfLine(pdf, x1, y1, x2, y2, color, width)	Adds a line between **(x1, y1)** and **(x2, y2)** of given width and color.
SfSetPdfRectangle(pdf, x, y, w, h, color)	Adds a rectangle to the PDF document.
SfSetPdfPie(pdf, x, y, w, h, color, startAngle, sweepAngle, width)	Adds a pie figure to the PDF document.
SfSetPdfFont(pdf, name, size, bold=false, italic=false)	Sets a font to the PDFdocument. All text added after this call will be using this font.
SfSavePdf(pdf, filename)	Saves PDF document to a file.

www.dbooks.org