SharpBlade, a guide

Making apps for the SBUI with SharpBlade 5.x

Adam Hellberg Brandon JC Scott

30th July 2014

Contents

List of Figures

List of Tables

Listings

1 Preface

This guide is open source! If you find anything wrong, missing or otherwise have suggestions for improvement, feel free to create an issue or fork the repository at: https://github.com/SharpBlade/SharpBlade-docs.

2 Introduction

This guide is meant for people just starting out with SwitchBlade User Interface (SBUI) development and who may be beginners to the C# language as well. Because of that, people with more experience in the field may find this guide slow paced, but we think it will still work good as an introduction to the SharpBlade library.

2.1 Terminology and formatting

Throughout this guide, we'll make references to both the SharpBlade Application Development Interface (API) and Razer Software Development Kit (SDK) API, which is characterized by how the text is typeset. SharpBlade components are set in a **`bolded green font`** and also link to the relevant documentation page at `http://sharpblade.net/docs` (only usable if you're viewing this online), while Razer SDK components are shown with simply **`bolded font`**, to match the style used in their own SDK documents. General code that isn't closely associated with either framework is shown in a `normal typewriter font`.

2.2 Automated SharpBlade documentation

The documentation found at `http://sharpblade.net/docs` is automatically generated each time SharpBlade is built by our TeamCity build server (every time new code is pushed to the GitHub repository). This guide was written with the intent that the automated documentation should be accessible while reading through the examples in this guide.

Of course, we try to explain all the API calls done in this guide and what they mean, but if you feel like you're missing something, make sure to look up the relevant function, method, class, struct or enum in the SharpBlade documentation, as it will have all the technical details and signature information. For things relating to the Razer API, you can refer to the SDK documentation that came with your Razer SDK installation.

3 Bugs and feedback

Found a bug or have a feature request? Please create a new issue on GitHub describing the situation and any steps to reproduce it in the case of a bug report): https://github.com/SharpBlade/SharpBlade/issues (please do a search first to see if your issue has already been reported by someone else).

4 Contact

Having problems getting SharpBlade to work? We will gladly help you get started! You can contact us personally, but try contacting the SBUI community via Razer's dev forum first. We frequent the forums, so it's very likely we'll get in touch with you over there, but with the added benefit that other members of the community can give their aid as well. Not to mention that other people having the same issue(s) as you can find the topic and get the help they need without having to contact us and wait for an answer!

You can find Razer's dev forums at: http://developer.razerzone.com/forums/ (login with your Razer ID).

If you'd rather get in touch us directly, you can find contact information on our websites: http://sharparam.com/ for Adam Hellberg (Sharparam) and http://brandonscott.co.uk/ for Brandon JC Scott (brandonscott).

5 Setting up your environment

5.1 Razer SBUI SDK

To develop apps for SBUI devices, you need to have the Razer SBUI SDK downloaded and installed on your system. The SDK can be downloaded from Razer's SBUI website: http://developer.razerzone.com/sbui/sbui-sdk/.

5.2 Visual Studio

The instructions and tutorials in this guide will assume that you are using Visual Studio Express 2013 for Windows Desktop, which can be obtained for free from Microsoft's website: http://www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop.

5.3 Installing the NuGet package manager

(If you intend to download SharpBlade from a source other than NuGet, you can skip this section)

SharpBlade is distributed as a NuGet package, and as such, you'll need the NuGet package manager to install it. You can get it from their project page at: http://docs.nuget.org/docs/start-here/installing-nuget

5.4 Obtaining SharpBlade

The easiest way to get SharpBlade is to install the NuGet package. Simply open the NuGet package manager console (Tools → Library Package Manager → Package Manager Console) and execute the following command: `Install-Package SharpBlade`. This will download the latest stable SharpBlade version and add it as a reference in your project.

6 App flow in SharpBlade

If you've read the documentation for Razer's SDK, you will have noticed the application flow that is listed there: `RzSBStart() -> RzSBQueryCapabilities () -> Register gestures/keys -> <app stuff> -> RzSBStop()`.

SharpBlade has a similar flow when creating applications, but in a way that should be familiar to C# developers.

6.1 The `RazerManager` class

`RazerManager` is the core of SharpBlade, it's what manages and provides all the other components like the `Touchpad` class and various `DynamicKey` instances. The class itself is a singleton, so you obtain a reference to it via the `RazerManager.Instance` property.

Like other singletons, this is done to enforce a single instance of `RazerManager` is used for an application. For your convenience, you may want to save a local reference to `RazerManager` like so:

```
var manager = SharpBlade.RazerManager.Instance;
```

Listing 6.1: Getting an instance of RazerManager

This (assuming it's the first access to `RazerManager.Instance`) will initialise the internal field with a new instance of `RazerManager`, which calls the necessary native functions to initialise the SDK. You can then use this manager instance to access the various parts of the SDK and perform method calls.

6.2 Life cycle

As mentioned, `RzSBStart` is called internally when the `RazerManager.Instance` property is first accessed, but when is `RzSBStop` called then? On dispose!

`RazerManager` implements the `IDisposable` interface, and calls `RzSBStop` when it disposes, either explicitly by code or when it's collected by the Garbage Collector (GC).

7 Creating your first app

The best way to introduce SharpBlade is to simply walk through the process of creating a new application from scratch.

7.1 Starting VS and creating a new WPF project

(First make sure NuGet is installed as described in chapter 5.)

Start your installation of Visual Studio (Windows Key (WinKey) + S[1] → "Visual Studio" → Enter) and create a new Windows Presentation Foundation (WPF) project.

1. File → New → Project (or Ctrl+Shift+N) (see figure 7.1)
2. Choose WPF as the project type (see figure 7.2)
3. Choose a name and location for your project (can be anything)
4. Press OK

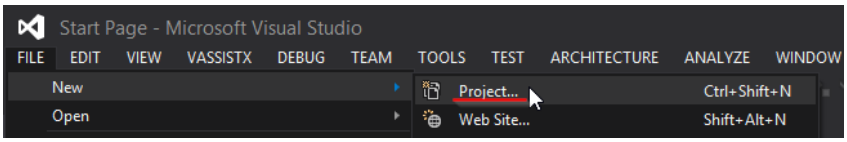

Figure 7.1: File → New → Project

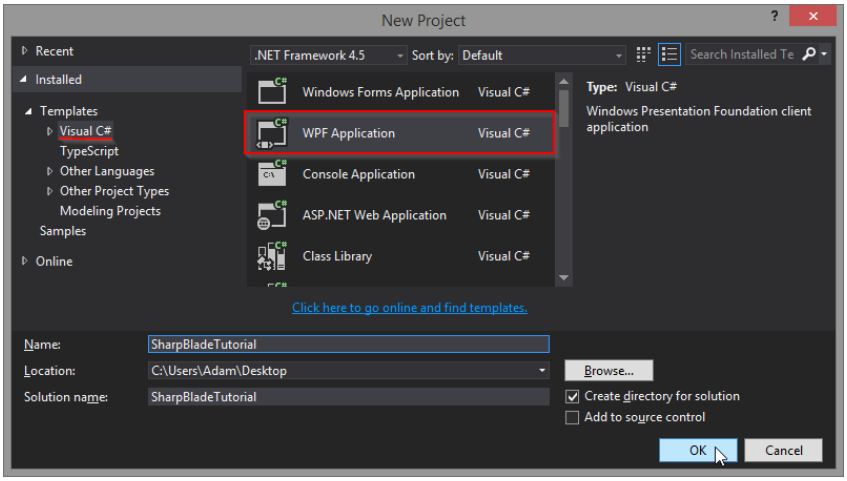

Figure 7.2: Creating a new WPF project

[1] Windows search panel or window

7.2 Installing SharpBlade from NuGet

(Skip if you're obtaining SharpBlade in a different way or you already did this step in chapter 5.)

Before we go further, you should install the SharpBlade library. In this guide we'll do it through NuGet, because it ensures you are getting the latest stable version of the library. Simply run the following command in the package manager console (accessible through the tools menu, see figures 7.3 and 7.4 for examples).

```
Install-Package SharpBlade
```

Listing 7.1: Running the Install-Package command

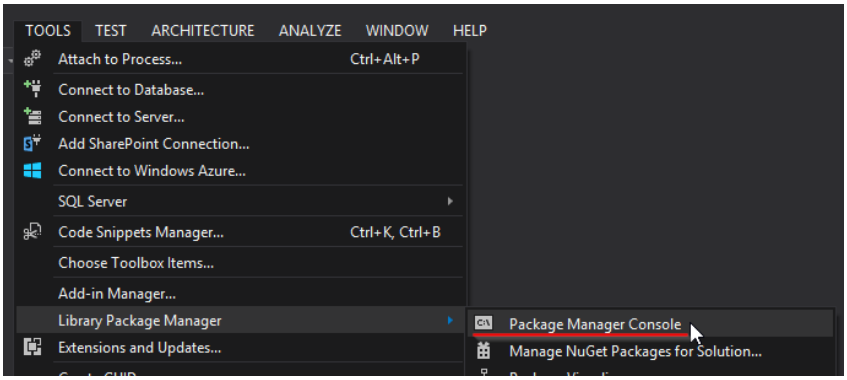

Figure 7.3: Accessing the package manager console

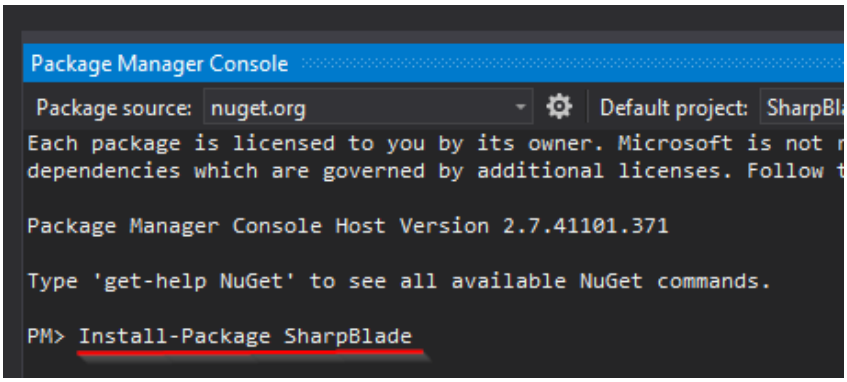

Figure 7.4: Running the Install-Package command

7.3 Setting up the window

You're now in your project with a newly created WPF window, make sure its dimensions are set to 800 pixels in width and 480 in height (the dimensions of the touchpad). Place something in the window in order to see if it renders properly to the touchpad (a label with some text or similar). Also make sure that the `WindowStyle` property of the window is set to `None`, to avoid the title bar and other such things to render to the touchpad (see figure 7.5).

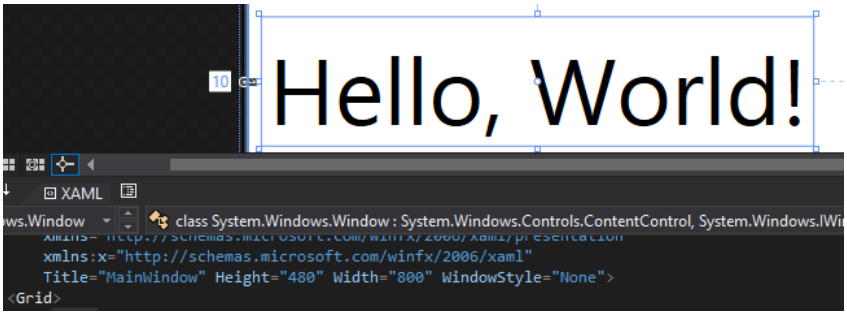

Figure 7.5: Customizing the WPF window

7.4 Initializing the manager and rendering

Now go into the code for the window (press F7 or right click the file for the WPF window and select "View code"). We'll want to add a private field to save a reference to `RazerManager` (for simplicity), so go ahead and do that (making sure to add a `using SharpBlade.Razer;` with the rest of the usings).

We'll have to assign an actual instance of `RazerManager` to this field, which we'll do in the constructor of the window, right after the call to `InitializeComponent()`.

A code example is provided in listing 7.2 (omitting some common `using` declarations that are inserted for you by the designer).

```
using SharpBlade.Razer;

namespace SharpBladeTutorial
{
    public partial class MainWindow : Window
    {
        private readonly RazerManager _razer;

        public MainWindow()
        {
            InitializeComponent();

            _razer = RazerManager.Instance;
        }
    }
}
```

Listing 7.2: Initializing RazerManager

This will initialize all Razer components and running the app at this stage would cause the SBUI to give control to our app and display nothing (since we haven't rendered anything yet!). As the next step, let's draw the window to the touchpad with the `Touchpad.SetWindow` method (`Touchpad` is accessible as a property on `RazerManager`).

```
public MainWindow()
{
    InitializeComponent();

    _razer = RazerManager.Instance;
```

```
    _razer.Touchpad.SetWindow(
        this,
        Touchpad.RenderMethod.Polling);
}
```

Listing 7.3: Drawing a window to the touchpad

Running the app now, you should see the window rendered to the touchpad at about 24 FPS (frequency can be modified by supplying a third argument to `Touchpad.SetWindow`).

Closing the app at this point (through either the window X button or pressing the *Razer Home key*) should work fine, as the finalize method on `RazerManager` will make sure the app is closed properly. That said, it doesn't hurt to subscribe to the `RazerManager.AppEvent` event and check for the `RazerAPI.AppEventType.Close` type event and do some clean-up there followed by exiting with `Application.Exit()`.

7.5 Enabling dynamic keys

Now that we have a window rendering on the touchpad we'll look into enabling some dynamic keys for additional app interaction. This isn't much more complicated than rendering windows, but we'll need a few more components, like a callback function to be called when a key is pressed.

Let's start with writing the callback method that will need to match the signature of a regular `EventHandler` delegate (the `sender` parameter is set to the dynamic key that was pressed).

```
private void MyDkHandler(object sender, EventArgs args)
{
    var dk = sender as DynamicKey;
    // A cast to DynamicKey should never fail in
    // SharpBlade 5.x, but it's possible future
    // versions could alter the behavior and it
    // could be worth using this method if you
    // rely on casting sender to DynamicKey
    if (dk == null)
        return; // Not a valid sender param
    Console.WriteLine("{0}␣was␣pressed!", dk.Type);
}
```

Listing 7.4: Creating the dynamic key callback handler

Now we will have to tell SharpBlade (which then tells the Razer SDK) to enable the dynamic key of our choosing. This is easily done with one of the `RazerManager.EnableDynamicKey` overloads. They are nearly identical with the exception that some of them take a callback method and/or a `pressedImage` string argument as parameters.

For this example, we'll use the overload taking an `EventHandler` since we have already written our handler method. This example is meant to complement the code in listing 7.3, so you should add the new code after the current code in the window's constructor, as we will be using the previously defined field `_razer` to access the instance of RazerManager.

Other than the callback, you will also need to tell the method what dynamic key (1-10) you want to enable, and the image file to display on

them. The last parameter tells it whether to overwrite the data associated with the specified dynamic key if it has previously been initialized. The last two parameters are optional and we'll discuss the method more thoroughly in the following paragraphs.

```
public MainWindow()
{
    DynamicKey dk = _razer.EnableDynamicKey(
        RazerAPI.DynamicKeyType.DK1,
        MyDkHandler,
        "image.png",
        "pressedImage.png",
        true);
}
```

Listing 7.5: Enabling a dynamic key

You'll notice we supplied `true` for the override parameter, this is to make sure that the dynamic key is enabled with the parameters we want. Had we passed `false` instead, and `DK1` had already been enabled by a previous call to `RazerManager.EnableDynamicKey`, this call would simply return the already configured dynamic key.

In the example we store the return value in a local variable, we can then use this o make further modifications to the dynamic key, or store it for future usage. If you choose to simply call the method without storing the return, which is convenient when you just want to enable a key and be done with it, you can obtain a reference to it later by calling the `RazerManager.GetDynamicKey` method.

We passed `"image.png"` and `"pressedImage.png"` as the third and fourth parameters, these are (as you probably already guessed) the images that will be displayed on the dynamic key depending on its state. `"image.png"` will be the image displayed when the key is in the up state or "idle". The framework will update the displayed image to `"pressedImage.png"` when the key is pressed down by the user (and then back to `"image.png"` as soon as the user releases the key again).

7.6 Compiling and testing your app

Congratulations! You should now have a working app, even if it doesn't do much yet. You can compile and run your application (by pressing F5 in Visual Studio) and after some brief compilation action, you should see your app running on the touchpad on your SBUI device!

If you encounter any errors during compilation, or get runtime exceptions, don't fear to contact us for help (or the rest of the SBUI developer community) through Razer's developer forums or our contact details listed in chapter 4 on page 8. Please do not make an issue on our GitHub repository for general help with the library, as that should be reserved for bugs within the library or feature requests.

Glossary

API : Application Development Interface. 6

GC : Garbage Collector. 10

SBUI : SwitchBlade User Interface. 6, 8, 9, 13, 15

SDK : Software Development Kit. 6, 9, 10, 14

WinKey : Windows Key. 11

WPF : Windows Presentation Foundation. 1, 2, 11–13

Index