Small Basic Programs on a Windows 8.1 Tablet Computer

By Philip Munts
Munts Technologies

Originally published here on the Microsoft TechNet Wiki on 22 September 2014

Introduction

I have awaited the arrival of the Windows 8.1 tablet computer with a great deal of anticipation. The tablet form factor is interesting for a number of user interface scenarios, especially in the world of embedded systems.

Unfortunately, I have found software development for iOS and Android applications unappealing. Xamarin for iOS and Android do allow using C# and .NET on those tablet platforms, but development is still cumbersome and expensive. The least expensive "Indie" packages from Xamarin are $300 a year for each platform. I use Xamarin Android with great success for developing Android apps, but it is neither easy nor convenient.

With a Windows 8.1 tablet, though, I can develop software with free development tools for Ada, C#, Java, Pascal, Python, (my favorites) and many other programming languages. Including Microsoft Small Basic, the subject of this article.

A Windows 8.1 computer, including a tablet, presents one of three views after the user has logged in. These views are the ***Start Area***, some ***Metro App***, or the ***Desktop Environment***. The Start Area (loved or reviled, depending on personal taste) presents an array of colored tiles, each representing a single Metro App, or desktop application program. It functions like the iOS and Android home screens. Metro Apps (aka Windows Store Apps) are the native applications for the new Windows Runtime environment. They take over the entire screen area, like apps on iOS and Android devices. The legacy Desktop Environment is the familiar Windows GUI (Graphical User Interface) environment we have known from Windows 3.1 onwards to today. The Desktop Environment is fully functional on a Windows 8.1 tablet, but the interface elements are very small and hard to see.

In order to run Small Basic programs on your tablet, you will need to make sure that the .NET Framework 3.5 is installed. On my tablet, after the many Windows 8 system updates that occur when a new computer is activated, .NET 3.5 was already installed. Otherwise, you will need to go to **`Control Panel`** → **`Programs and Features`** → **`Turn Windows features on or off`** and select ".NET Framework 3.5 (includes .NET 2.0 and 3.0)".

Strategy

Small Basic application programs run on the Desktop Environment view. It is possible to make them look and act very similar to native Metro Apps using the following basic strategy:

1. Maximize the application window when the program starts.

2. Enlarge all user interface elements to compensate for the small tablet screen.

3. Install the application such that a tile is created in the Start Area.

My strategy uses some services provided by the excellent ***LitDev*** extension (available at http://litdev.hostoi.com). I have found this extension so useful that I have repackaged it into a Windows installer (available at http://tech.munts.com/SmallBasic). My installer greatly simplifies installing the LitDev extension, which will be necessary on your development machine. (Neither Microsoft Small Basic nor the LitDev extension are necessary on the Windows 8.1 tablet computer, unless you plan to develop programs on it.)

Example Program

I have posted a sample Small Basic program, **tabletexample.sb**, that illustrates a GUI layout suitable for a Windows 8.1 tablet computer. It is available for download at http://tech.munts.com/SmallBasic/TabletExample.

It is also published as program **HJR688**, but since it uses the LitDev extension, it can't be imported into Small Basic and run unless you do some manual editing to uncomment some code that http://smallbasic.com/program seems to think is dangerous.

The following code fragments are extracted from **tabletexample.sb**.

Detecting a Tablet

It is entirely possible to develop Small Basic applications that will run on *either* a conventional Windows Desktop system or on a Windows 8.1 tablet. The question immediately arises, how to detect whether your application is running on a tablet in the first place? Various schemes for detecting the type of machine exist, but I settled on a simple and universal solution, the ***flag file***. On the Windows 8.1 tablet, I simply create the file:

```
C:\ProgramData\MuntsTechnologies\tablet.txt
```

Then, at the beginning of each Small Basic (or other programming language) application, you just check to see if **tablet.txt** exists. The LitDev extension provides the method **LDFile.Exists()**, which returns **"true"** or **"false"** depending on whether the specified file name exists. The following code fragment illustrates how to detect whether the program is running on a Windows 8.1 tablet:

```
If LDFile.Exists("C:\ProgramData\MuntsTechnologies\tablet.txt") Then
  ' Layout the GUI for Windows 8 tablet
Else
  ' Layout the GUI for Windows Desktop
Endif
```

GUI Layout

This first thing to do when laying out the GUI (Graphical User Interface) for a Windows 8.1 tablet computer is to maximize the application, using the LitDev extension property **LDUtilities.GWState**. Then you can determine the screen width and height from the **GraphicsWindow** properties **Width** and **Height**:

```
LDUtilities.GWState = 2
W = GraphicsWindow.Width
H = GraphicsWindow.Height
```

This works in either portrait (vertical) or landscape (horizontal) mode. A horizontally centered layout provides a pleasing visual effect, so we base all GUI element placement against the horizontal center point, which is **W/2**. We also set **GraphicsWindow.FontSize** to a larger size than we would normally use, to make text elements larger and easier to see, and controls easier to hit with a finger:

```
GraphicsWindow.FontSize = 36
GraphicsWindow.DrawText(W/2 - 200,  50, "Small Basic Windows 8")
GraphicsWindow.DrawText(W/2 - 150, 100, "Tablet Example")
GraphicsWindow.FontSize = 30
ButtonChange = Controls.AddButton("Change", W/2 - 200, 200)
ButtonQuit = Controls.AddButton("Quit", W/2 + 100, 200)
```

This GUI layout flows from the top of the screen, with horizontally centered rows of elements. It works well whether the tablet is oriented vertically or horizontally. It does not, however, recenter everything if the tablet is reoriented *after* the program has started.

Touch Input

The Windows 8.1 touch input services seem to work well with Small Basic user interface elements. "Mouse" elements (buttons, sliders, etc.) work fine with touch input. Text boxes, however require a little extra attention.

In native Metro Apps, if you touch inside a text box field, the on screen keyboard appears to let you begin typing text into the field. When the text field gives up keyboard focus, the on screen keyboard disappears. This behavior is not carried forward into the Desktop Environment.

It is possible, but awkward, to run **`osk.exe`**, the on screen keyboard program, whenever the application program detects text input. It is also possible for the user to touch the keyboard icon on the Desktop Environment task bar to manually bring up the on screen keyboard for text input. Neither of these options yield a satisfactory user experience.

Fortunately, somebody has already solved this problem. The commercial program ***Tabtip On-Demand*** (available for purchase for only USD $1.99 at http://chessware.ch/tabtipod) was been written for exactly this situation. It hooks into the Desktop Environment somehow to detect when a text field in an application program has gotten keyboard focus. It then automatically pops up the Windows on screen keyboard. If no text field retains keyboard focus, the keyboard is dismissed. The behavior is nearly indistinguishable from Metro Apps.

Building an Installer

Compiled Small Basic programs can be copied to and run on other computers that do not have Microsoft Small Basic installed. However, every compiled Small Basic program requires at least one library **`.DLL`** file, **`SmallBasic.dll`**. Every extension also contributes a library **`.DLL`** file. Therefore, deploying a Small Basic application to another computer requires copying the **`.EXE`** and one or more **`.DLL`** files from the development machine to the target machine. This gets old very fast. Furthermore, it is very handy to create a shortcut in the ***All Programs*** menu (and Windows 8 Start Area) when you install an application program. All of this can be accomplished more or less automatically by packaging your application program into an installer.

I use a free installer builder called ***Inno Setup*** (available at http://www.jrsoftware.org). Inno Setup provides an installer compiler program that reads a text file (**`.ISS`**) describing the installer package and generates an installer **`.EXE`** suitable for distribution. Here is the heart of the **`.ISS`** file I wrote to package my example:

```
[Setup]
AppName=Windows 8 Tablet Example
AppVersion={#APPVERSION}
AppPublisher=Munts Technologies
AppPublisherURL=http://tech.munts.com
DefaultDirName={pf}\Windows8-Tablet-Example
DirExistsWarning=no
Compression=lzma2
SolidCompression=yes
OutputBaseFileName=Windows8-Tablet-Example-{#APPVERSION}-setup
outputDir=.
UninstallFilesDir={app}/uninstall

[Files]
Source: "*.dll";        DestDir: "{app}"
Source: "*.exe";        DestDir: "{app}"

[Icons]
Name: "{commonprograms}\Tablet Example"; Filename: "{app}\tabletexample.exe"; WorkingDir:
"{app}"
```

The **[Setup]** section specifies the name of the application, who published it, where it comes from, where it will be installed, etc. The application version number **APPVERSION** will be supplied externally as a command line argument to the Inno Setup compiler program. The **[Files]** section specifies what goes into the installer package, in this example, just **.DLL** and **.EXE** files. The (somewhat misleading) **[Icons]** section specifies the shortcuts that will be created when the application is installed.

I have published a DOS batch file, **build.bat**, that automates creation of the installer package from the Small Basic source code. It runs the Microsoft Small Basic command line compiler **SmallBasicCompiler.exe** and the Inno Setup command line package compiler **iscc.exe** to produce the final deliverable installer package.