**Introduction to ApsimNG – the GtkSharp user interface for ApsimX**

**The Build environment**

Before you can do any development work on ApsimNG, you will need to set up your development environment appropriately. You should be able to edit and compile this application using either MonoDevelop (aka Xamarin Studio) or Visual Studio. I’ve done the initial development using Visual Studio 2015, but other environments should work just as well.

**Installing GtkSharp.** Whatever IDE you choose, you will first need to install the GtkSharp modules on your system. Although there are GtkSharp packages available on NuGet, I strongly recommend that you do **NOT** install them. They are set up for Gtk 3, which doesn’t play particularly well on Windows at this stage. I recommend you get the Gtk# for .NET installer from the Mono project. ~~However, don’t use the link at~~ [~~http://www.mono-project.com/download/#download-win~~](http://www.mono-project.com/download/#download-win) ~~– this will give you the installer for GtkSharp 2.12.30, and that version has problems with in-built icons. This problem results in the file open/save dialogs not displaying folder icons for directories. Instead, I recommend using the installer for GtkSharp 2.12.26.~~ **~~You can get this at~~** [**~~http://download.xamarin.com/GTKforWindows/Windows/gtk-sharp-2.12.26.msi~~**](http://download.xamarin.com/GTKforWindows/Windows/gtk-sharp-2.12.26.msi)**~~.~~ Update:** a newer installer for Mono on Windows (Mono version 4.4.0) includes an updated GtkSharp (version 2.12.38) which handles the icons correctly. It’s now OK to use the installers at <http://www.mono-project.com/download/>. Run this installer, and it will install a lot of stuff in (by default) \Program Files (x86)\GtkSharp\2.12. The native binaries needed for Gtk (basically the Gtk+ layer) reside in the bin sub-folder. You should add \Program Files (x86)\GtkSharp\2.12\bin to your PATH environment variable if you want to be able to make use of these easily. (Well, it’s easier than copying them to your development bin directory. Be careful though: it is possible that you may already have a Gtk folder on your path. If you do, and it’s a different version, it’s up to you to sort out the resulting DLL hell.) The .Net assemblies that make up the GtkSharp layer are in sub-folders of the lib sub-folder. You can manually add these as references to the ApsimNG project (actually, they should already be there).

You will need three other Gtk Sharp .Net assemblies that the installer won’t have provided for you: Mono.TextEditor.dll, Xwt.dll and webkit-sharp.dll. I tried to put copies of these into an Assemblies sub-folder of the ApsimNG folder, but git is set up to ignore binary files. **Instead, you can download them from** [**http://bob.apsim.info/Files/Assemblies.7z**](http://bob.apsim.info/Files/Assemblies.7z)**.** The Mono.TextEditor.dll is taken from the Xamarin Studio distribution, and provides a nice language-aware text editor. It uses Xwt.dll, which must lie on the run-time path, but isn’t needed during compilation. The webkit-sharp.dll is a thin layer around the WebKit browser, and isn’t really of much use on Windows (where IE is being used as the embedded browser), but you need it so you can compile code that will support the WebKit browser on Linux and MacOS. It’s hard to find a copy of webkit-sharp.dll for Windows; I’m actually using one taken from an Ubuntu system. I think all the other referenced assemblies should get pulled in automatically by NuGet (use OxyPlot.GtkSharp, not OxyPlot.GtkSharp3).

**Glade.** There are several options for designing forms in GtkSharp. It’s actually not all that hard to do it all in code, but sometimes it’s nice to have GUI designer to work with. There is no GtkSharp designer for Visual Studio; Xamarin Studio has an in-built designed named “stetic”, but I decided not to use that because not everyone will necessarily want to use Xamarin Studio. Instead, I used the Glade GUI designer. I’m currently using Glade 3.8.5, available at <http://ftp.gnome.org/pub/GNOME/binaries/win32/glade/3.8/glade-3-8-5-installer.exe>. Windows (well, Windows 7 at least) is quirky about allowing you to run glade. If a binary is named “glade-3.exe” you get an annoying message box each time you run it. Change the name to just “glade.exe” and the message goes away. I can’t explain why this is so; ask Microsoft. Anyway, Glade allows forms to be saved in two slightly different formats: either GtkBuilder or Libglade. GtkBuilder is the Gtk 3 way of doing things, but isn’t supported fully in Gtk 2, so I have used the Libglade format. I recommend that you stick with that, for now. It shouldn’t be hard to convert later on.

**Tutorials and documentation**

GtkSharp has some notable differences from Windows.Forms, so there is a bit of a learning curve as one gets used to using it. Documentation is a bit sparse in some areas, so it can take a while to really get on top of things. I’ve found several resources to be particularly useful:

1. The best detailed resource in the Mono documentation at <http://docs.go-mono.com/index.aspx>. The Gtk stuff is at <http://docs.go-mono.com/?link=N%3aGtk>. Here you will find documentation on the various classes and their members.
2. If you need a tutorial to get started in the GtkSharp world, <http://zetcode.com/gui/gtksharp/> gives a simple, but reasonable thorough, run through most of the GtkSharp interface elements. A few other tutorials can be accessed from <http://www.mono-project.com/docs/gui/gtksharp/tutorials/>
3. If you’re trying to make sense of the inner workings of Gtk, there are some very good course notes available at <http://www.compsci.hunter.cuny.edu/~sweiss/course_materials/csci493.70/lecture_notes/>. Although written for the Gtk+ layer rather than the Gtk# layer, they do a good job of explaining some of the concepts of the Gtk world.

**The current state of play**

As of this writing (20 April 2016) ApsimNG is reasonably usable, but not yet quite complete. Some of the areas that clearly need refinement are the use of “accelerators” (tying keystrokes to events) and a more sophisticated implementation of the GridView. Also, the “Upgrade” form hasn’t yet been implemented.

Development to this stage was based on the existing UserInterface code. I started with that, commented out pretty much everything that relied on System.Windows.Forms, then gradually started re-implementing stuff in GtkSharp. Most of the commented-out code has the acronym “TBI” (for “to be implemented”) associated with the comment, so you can readily search for this string to locate the gaps in implementation.

Parts of my approach to coding GtkSharp have changed as I’ve gotten a better understanding of how GtkSharp works. My first few forms used signal (event) handling definitions in glade, but I’ve gotten away from that, and now just use “+=” operators in the code to add handlers for events. I personally find this easier, especially since the Glade GUI is not integrated with Visual Studio. I should mention that the glade files all reside in the Resources\Glade sub-folder, and are added to the Project as embedded resources.

I’m using HTML for the Summary file output, but the HTML is currently really ugly, and has almost no nice (e.g., table) formatting. In the longer run, though, it should be a lot easier to work with than RTF.

I haven’t really consider all the issue regarding writing installers for this thing, but it can run on Linux and MacOS, with a bit of care in setup.

**Cross-platform things that just don’t work (yet)**

There are a few Windows-specific bits of UserInterface that can’t be directly ported to ApsimNG. The first is the recently added MapView. It makes use of GMap.NET.Windows.Forms, and there doesn’t appear to be a GtkSharp counterpart in existence. It may be possible to get the same basic functionality be using Google Maps in a browser window.

Another problem is with the generation of model documentation, which is almost, but not quite, platform-independent. The problem here is with PdfSharp-gdi, which is tailored for use with Windows GDI+. Again, there is no GtkSharp counterpart. If fact, it almost works, but it tries to get and use a Windows DC to obtain font information. There may be a workaround (and it may be worth seeing what the WPF and Silverlight versions do).