Programmer Documentation

Phaser Configurator

The Phaser Programmer is a much simpler application, as all it does is take existing Bar Files, extract the pattern information from them, and saves that on physically connected Phasers. All of the tasks the Programmer performs fits into only a few files, and was made using WPF and the .NET 4.5.1 Framework.

# SharpDevelop

Due to the fact that the Programmer does not utilize Unity (and is in fact its own application) you will require a separate application called SharpDevelop. Installation files for it are available from the [Personal Folders](file:///U:\Personal_Folders\James\SharpDevelop).

In the linked folder there will be three files. Two of them (2013 Build Tools and dotNet DevPack) are dependencies for SharpDevelop to run and operate, while the third is the installer for SharpDevelop itself. I highly suggest you install the two dependencies before the installing the IDE itself.

# The Programmer

As far as modification goes, there should only be two files you will need to examine: MainWindow.xaml and MainWindow.xaml.cs. The former is considered the “design” of the window, while the latter is the “code-behind” of it. To get to the Code-Behind file, you will need to expand the former in the Projects pane on the left, as such:

## Design

The Design is written up using XAML – a language that’s based on XML used to design the view of the window. However, it’s not necessary to work with the source code for the Design, as SharpDevelop implements a Designer that lets you graphically modify the application’s appearance.

Any windows made using WPF utilize a core component called a Grid that divides up the window into a grid. As of writing, the window’s main Grid has only two rows – the upper one being the main area, the lower simply displaying the connection status.

The main area utilizes a TabControl that allows for two tabs that can be swapped between. The Designer will let you switch between both tabs, you need only click on the tab itself. Additionally, each tab also contains a Grid, but only out of necessity – both Grids are one column by one row.

Every compnent of the window has its own properties (like every Component on a GameObject has their own properties). Simply select an item and a list of modifiable properties will show up on the right in the Properties pane.

## Code-Behind

The Code-Behind file contains all of the processing code required for the application to operate. It checks in on the MCP2210, ascertains connections, and allows the user to read and write from a connected MCP2210, with the assumption that beyond the MCP2210 is a Phaser Bar.

Every pertinent action taken by the user – dropping a file into the window, clicking things, etc. – will create an event on the window, and the Code-Behind’s code handles those events.

The Code-Behind handles 7 different events as of writing:

* ReadBrowse\_Click – Called when the “Browse” button on the “Retrieve Program from Bar” tab is clicked.
* WriteBrowse\_Click – Called when the “Browse” button on the “Send Program to Bar” tab is clicked.
* FileDragEnter – Called when the user drags a file over either text box.
* FileDragDrop – Called when the user drops a file onto either text box.
* ReadBar – Called when the “Receive” image on the “Retrieve Program from Bar” tab is clicked.
* WriteBar – Called when the “Send” image on the “Send Program to Bar” tab is clicked.
* Window\_Closing – Called when the user has clicked the Close button on the window.

Additionally, the Code-Behind makes use of a DispatcherTimer, initialized in the constructor of the window (near the top of the file). While the 7 events above only run when the user does something, a DispatcherTimer calls a chunk of code on a clock – in the case of the Programmer specifically, this occurs every second.

Lastly, there are two additional classes near the bottom of the file – BarWriter and BarReader. These classes do nothing particularly fancy – they simply act as a handier means to perform actions on an array of bytes. BarWriter writes to said array (additionally keeping track of where it last wrote), while BarReader reads from said array (additionally keeping track of where it last read).

## The Other Files

For the sake of completeness, there are a few other important files that are a part of the Programmer. While I can’t imagine needing to modify them, I’ll still list them off here:

* App.config is a configuration file for the entire application, pre-generated by the IDE. I’ve never touched it, and I’d imagine you should see no need to either.
* App.xaml contains resources that the rest of the application might need. Which is none.
* Device.cs is the wrapper class for the MCP2210, used to make modifying various elements of the pieces of settings easier and cleaner. Everything the class exposes is documented, however there might be a few things that aren’t wrapped.
* fNbt\_v0.5.1.cs is the file that provides the whole of the fNbt source code. Care should be taken when modifying this file, as most of that code is not mine, I only know how to use it.
* The Assets folder contains all of the assets the Programmer uses (read: images). There might be a few that aren’t important in that folder, but they’re kept anyway just to be extra certain.
* The Plugins folder contains DLLs required for the Programmer to communicate with an MCP2210. Warning: these don’t get compiled into the project with the rest of the stuff listed; the MCP2210 DLL needs to be shipped along with the Programmer EXE, and exist in the same folder, and have the name “MCP2210DLL-M-dotNet4.dll” – otherwise, the project will not work.