Program Structure

The structure of this program is completely different to the old one, (as I now know how to use the object orientated-ness of VB properly) and it because this program needs to do more and needs to be expandable.

The first concept the program uses a lot is 'interfaces.' An interface as far as the program is concerned is a way of talking to an external program and exchanging data, while all different interfaces can talk appear to the main body of the program as the same, as far as sending ad receiving data is concerned. Certain interfaces will be set up as 'readonly,' or what protocol is used by the interface, for example, so the main body of the program can work out what should be sent to received by the interface.

As there is a lot to the program that may need configuring, each interface will have an array of settings, all settings are contained within a class, to keep the code tidy, and so a set of settings can be passed within the program easily.

How the interface system works

The InterfaceParent.vb class is the generic interface that the main program uses to send its data. It has functions such as 'Send,' and events for when data is received. It will also have functions to allow it to pass packets back as one string rather than a trickle. It has properties such CanWrite/CanRead so the main program knows what to send to a particular instance of the class. When an instance of the class is created, the settings are passed to the class, which tells it what type of interface it will be, so it then knows what specific class to use to actually send the data. For example, when a new FLDigi interface is created using the InterfaceParent class, it will create a new generic TCP interface, and whenever its asked to send data it then forwards, perhaps processing he data in some way before hand, and passes it to the generic TCP interface class.

Class Descriptions

GlobalSettings.vb

A class which essentially contains all the settings within it. Information about the interfaces are stored with classes with this class (in the interfaces_ list).

Usage in the program:

One instance of this class is stored in the main form. It is passed to the settings form (for example) so that form may populate itself with the existing settings. The form then has its own copy of the class, and when it closes It updates its copy with the settings entered by the user. The form which originally opened that form, the main form in this case, then copies the instance of the class in the settings form back into itself so it has the updated version of the settings. Most settings classes are passed around in this way

InterfaceSettings.vb

A class to hold settings about an interface, such as its type, protocol, etc. Settings specific to an interface type are stored in the InterfaceTypeSpecificSettings_ object. Filters are also (but not yet) stored in this class

Usage in the program:

The global settings form contains a list of these classes for each of the interfaces the program is using. This class is also passed onto and stored in the classes responsible for implementing the interfaces.

AGWPE/FLDigi/GoogleEarth/etc Settings.vb

These classes contain settings specific to each interface, and contain settings such as Host, Port, Baud etc.

Usage in the program:

This class is normally stored in the InterfaceTypeSpecificSettings_ variable in the InterfaceSettings class, as they form part of the interface settings.

Frame.vb

A class which stores all the data about a particular frame. Data is entered into the frame class by giving either an APRS or UKHAS packet, and functions in the class decode the packet and store the data. This class is currently a bit of a mess, and I think the UKHAS and APRS packets use different units for a start. Eventually it will contain functions that return the data in flexible ways, depending on how the programmer wants to use the data.

InterfaceParent.vb

A class which bridges the gap between the specific classes for the specific interface, and a generic interface for the rest of the program to send and receive data in a nice way. This class holds an instance of InterfaceSettings to instruct it how to operate. This class will also carry out filtering of packets, and will try to pass data back to the main form in a 'nice' way, ie sending one APRS/UKHAS packet at the same time rather than a stream of characters one at a time.

The User Controls in the SettingsUC folder

These simple controls are the specific user control to enter data for each specific interface. Each control can be passed its interface specific setting class and pass it back with updated settings to its parent dialog.

AGWPE_APRSPacketHandler.vb

A class carried over from the old program (under a new name) which maybe useful. Its primary task is to take data from my TCP interface class and process the agwpe stuff and split the packet into useful parts (ToCall/FrmCall/DataPayload etc) for type 'U' packets from AGWPE, but contains functions to create various types of agwpe packets.

Enums

A few enums are used as detailed, and are kept in the randomfunctions.vb module

InterfaceTypes - Used to store what type of interface something is thorughtout the program
InterfaceDirections – Used to store in which direction data moves thorugh the interface, and to an extent what sort of data
PacketFormats – Used to store what protocol is used for a interface

Each enum has a length field also, which as enums are zero based, its value will return the number of 'useful' elements in the enum. A BLANK option in each enum is provided as a null option, but is not considered to be a 'useful' option so is not included in the count of elements.

In the randomfunctions.vb module there are also a few functions to convert the enums to text and back, so they can appear as something more useful when they are required to be exposed to the user.

Forms and Dialogs

MainFrm

The main form displays all the packets and the current position in the same way the old program did. The actual part that displays the location is moved to its own user control to keep everything tidy, and has just been copied from the old program, updating to occur soon, and the tabpages displaying the packets will probably be the same in the end.

settingsFrm

This form is where all the settings are changed. When the form exits, it writes all the settings on the form to an instance of GlobalSettings, as explained above

EditInterfaceSettings

This dialog is used to edit an individual interface, and is called by the settingsFrm. There is a space for a usecontrol which is used to set the specific settings relating to the type of interface. When a type of interface has been selected, one of the user controls in the InterfaceSpecificSettings folder is loaded. Upon closing this form, the settings from within the interface specific user control are copied into an instance InterfaceSettings along with the other settings in the form.

UplinkFrm

This is a small dialog where text to be uploaded to the balloon is entered. There is a combobox which contains all the possible interfaces to use.

InterfaceStatus

A currently empty form which should allow the user to see which interfaces are active, which have errors, which are disabled etc.. and be able to manage them

Ideas Currently Not Implemented / Implemented Well

Error handling – currently nothing has been done, but when an error is detected, probably by a try-catch statement, depending on the error, it will probably be best to update the interfaceStatus form and a little status message at the bottom rather than a messagebox every time

Validation – when implemented probably best to put in the relevant settings class. One thing to be aware of is that not every interface type/direction/protocol combination is practical and what are needs to be decided by a function in the InterfaceSettings Class (there is already a AllowableFormats function to start this off)

'Data Movement' – The bit of the program that when gets a packet from the balloon, for example, works out what to do with the data. Firstly, it needs to be logged, but then it may need to be shared via APRS. However, with data sharing it needs to decide whether the packet is a duplicate, as the main status window should not be clogged up with duplicate packets.

Frame.vb – Currently not written particularly well as just contains functions from different sources, and a mixture of units being used. Needs sorting, and needs functions to allow the data to be displayed in a useful way (eg failed checksums red)

Interfaces – Various interfaces have not been implemented or implemented very basically. The interfaceParent also has limited functions, that will probably be increased as the program progresses.

Saving Settings – Needs to be implemented in the settings classes

The Interfaces

FLDigi – A reasonably simple interface to code, just uses a tcp interface (127.0.0.1:7322 by default, local only) and shoves out data as it gets it. Will need to make use of the 'tiding' up function in the interface parent

MapPoint – The regular version of MapPoint will have some sort of way to control it, however, the a standalone program (requiring MapPoint),

APRSPoint, which was used for APEXI can take agwpe packets if the standard MapPoint interface is too annoying

GoogleEarth – Accepts data on the tcp interface. Just need functions to format the data. Functions to create a Google earth kml exist, possibly reusable

Serial Port – included incase it is needed. Probably only needed for out data due to the use of FLDigi. Interface already programmed

Internet DL server – Used to upload data to the DL server, havnt looked into this, DL-FLdigi does it so not priority. Grabbing data from the DL maybe more useful, there is a webpage with the raw data arranged nicely, which should be simple to get and decipher

Agwpe – A right pain, but work has already been done, just needs putting into a nice class