RetroShare: Writing Plugins

RetroShare Team

October 16, 2013

1 Introduction

So you have been using RetroShare for a while, and as a computer programmer, you want to add your favourite plugin to share with others.

Where to start?

Well, that is the purpose of this guide. It will provide a guide for developing it and getting it working with RetroShare with the least amount of effort possible.

This was written as a plugin was being developed, so the screen shots displayed are for that plugin.

2 Concepts

There are 3 types of plugins:

1. services based on friend communication eg VoIP
2. services based on turtle router eg ZeroReserve
3. services based on cache exchange eg LinksCloud

There are 3 basic parts to each plugin:

1. Serialiser - messages that you want to send over the network. You just need in the constructor of your class to declare each member and what type it needs to be serialized with.
2. libretroshare service - send/recv msgs and prepare data for GUI.
3. rs-gui window - get info from service, and display it!

All plugins working in the 0.5.* version and releases, will continue to work with 0.6 which changes to a new system using GXS.

3 Important Things

3.1 Language

RetroShare is written in c++ with some parts in c. You need to have a good understanding of c++.

3.2 Tools and Libraries

3.2.1 QtCreator

QtCreator is a free tool that can be used to develop the project, as well as editing, and compiling to create a library as either a *.so for Gnu/Linux or *.dll for Windows or *.so for Mac.

3.2.2 Editor

Many programmers use vim (from `http://www.vim.org/`) with some addons to make editing easier and faster, especially on older, slower computers. You can get a complete collection of plugins and settings for vim from `https://github.com/morpheusbeing/vim`.

3.2.3 Libraries

Unless your plugin is very complicate, all the required libraries should be available if you can compile a working version of RetroShare.

3.2.4 Version Control

RetroShare uses svn for its version control. If you are compiling your own, you will understand the use of the svn up command on Gnu/Linux.

For a project like this, git and `https://github.com` is a good option. You will need to create an account if you don't have one, and create a repository to store your code in. This allows you to keep an offsite copy of your work, as well as allowing for collaboration. There are many useful tutorials on using git, but essentially it is as follows:

1. Login or create an account at `https://github.com`.
2. Create a new repository (best to use the same name as your plugin name).
3. cd into the plugin folder you are working with
4. create a README.md file and edit the contents of it.
5. Initialise your local repository:

   ```
   git init
   ```

6. Add the file you jsut created to your repository:

   ```
   git add README.md
   ```

7. Commit the changes made into your repository:

   ```
   git commit -m first commit
   ```

8. Map your repository to the remote repository using the command:

   ```
   git remote add origin https://github.com/[your user name]/[your plugin].git
   ```

9. Now add any other files you have created. To add a folder, just use the folder name, and will track all the files.
10. Now copy your repository to the remote :

```
git push -u origin master
```

11. Now check the status of local git:

```
git status
```

For more information about using git, check the internet for tutorials and information.

3.3 Structure

It is possible to put all the files in one folder of the plugin name so that it is ~/retroshare/plugins[your plugin] as shown in 1

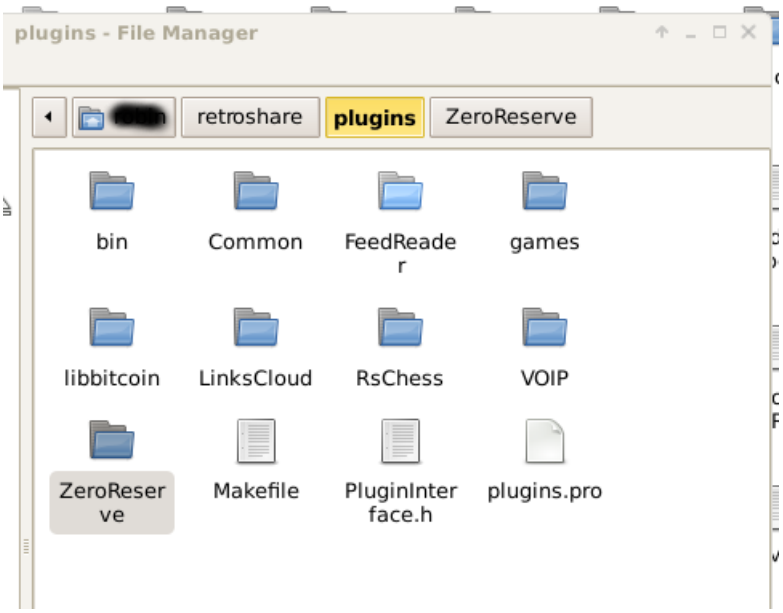

Figure 1: Folder Structure For Plugin

As a suggestion, establish some folders in you plugin to store various parts of your project into (refer to 2)

You will notice a folder .git - this is for versioning control - refer to 3.2.4.

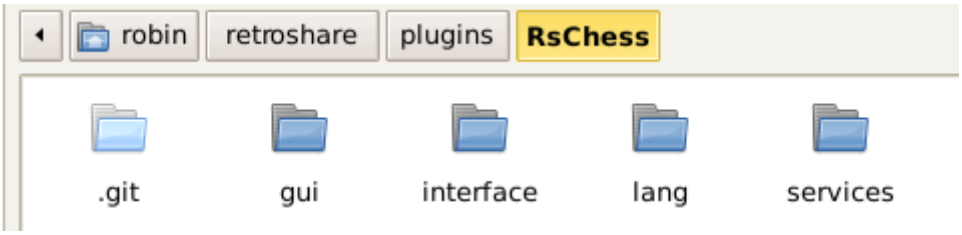

Figure 2: Sub-folders for the plugin (suggested)

4 Putting It Together

4.1 Serialiser

5 Documentation Processes

Index