ReadMe ICC Bot Library

**GET A COPY OF THE LIBRARY**

With git clone the repository <https://github.com/avwohl/xojo_test_bot.git> . This document is the Microsoft word .docx file in that directory.

**OVERVIEW OF THIS LIBARY**

This folder is a library of code to connect to internet chessclub chessclub.com. It handles the data connection and parsing the datagrams from the chess server. It is useful for creating clients for chess playing or robots to provide services. For a chess playing client you would also want the time lag correction provided by timestamp. This library is open source where as timestamp is not to help prevent cheating.

This library was written by Aaron Wohl [wohl@chessclub.com](mailto:wohl@chessclub.com). Although a long time ICC employee/consultant, this project was not funded by ICC. The library was written to provide a foundation to make it easy to create robots that work with ICC. Any opinions expressed in this project are those of Aaron Wohl. This project is in no way blessed or supported by Internet Chess Club chessclub.com or Xojo.com

The files in this directory that begin with ICC should not be modified for your application. These files are designed to be subclassed to customize them for a particular application. When a new verion of this directory is released the new version can replace this one and be compatible. If you make changes to the ICC files they could easily be lost.

If you find a bug in this library or wish to extend it to be more useful and share those changes with other users then it would make sense to make changes here and submit the changes.

**WHAT IS XOJO?**

Xojo is a programming language. It used to be called Real Basic but was renamed to not scare off Basic-a-phobes. It is available from [www.xojo.com](http://www.xojo.com). There is a free version for development, see the gotchas section below. The pay version is required to ship a compiled executable.

**WHY WRITE A LIBRARY IN XOJO?**

* Xojo can build an application for Macintosh OS, Microsoft Windows, apple IOS, linux interactive, windows command line, linux command line, web server, and soon android. The price for the full xojo version is about $700/year. This is a lot less than say xmarin which can ship to a similar number of platforms.
* Xojo is fun environment to develop and debug in.

**DO THIS FIRST**

Before doing anything else with this project, run xojo and on the edit menu pick preferences. Select the default save format to text (not binary or xml). If you are using the free xojo you can’t do this. If for any reason you are not using the text save format be sure to read the free vs paid section below as maintaining the source files in the free version can be very confusing and doesn’t work well with source control like git.

**GOTCHAS WITH USING THE FREE XOJO**

This library is written in xojo a basic like language (that used to be called real basic). Xojo was chosen for this project because it can build programs for windows, windows console, linux, macintosh macos, apple IOS, and soon android.

Xojo is a development environment that comes in a paid or a free version. The main difference is the paid version allows compiling the sources to build a binary executable for deployment. The free version runs code in a development IDE. The free version is more suited to development of code than distributing an application.

It may seem that an application should be developed in the free version and a license purchased when it is time to build an executable to ship. That is generally the flow the xojo sellers seem to suggest. However, there is one snag that may complicate that. The problem is, that the free version of the development IDE can only save it’s source files in a proprietary binary format. The sources here are distributed in plan text format that works well with Git, Mercurial or whatever source management control system you prefer. When changes are made with the paid IDE the text differences easily show form version to version in source control. However, with the free version the files all of the files of the project (all source files, and the project information) are all stored in the single \_xojo\_project binary. WARNING the existing source files are not deleted. So if you use a free version you end up with two copies of every sorce file, one in the file system where you can see it (but it is unchanged from the original) and your changed version inside the binary project file.

IF YOU ARE USING THE PAID VERSION or USING THE PAY VERSION WITH THE DEFAULT OPTION SAVE AS BINARY, THEN ALL YOUR CHANGES TO ALL FILES GO INTO THE \_xojo\_project\_binary\_file.

Xojo comes in many versions depending on if you are shipping to one platform (windows/mac/linux/…) or all of them. You may wish to get a paid licence for your favorite platform so git / source control works well, then later upgrade the license to include all the platforms.

**BEWARE OF THREADS**

Xojo has a thread object <https://docs.xojo.com/Thread> that executes asynchronously from the user interface and other processes. It may be tempted to use this to run long running tasks. However, extreme caution is advised when using xojo thread. The reason is that almost all of the documented xojo functions and features are not usable from a thread. The vast majority of xojo features can only be used from the one / main thread your program gets for free when the user starts it. If you create a new thread it cannot touch any window, window control. The thread can only communicate through shared memory in your program to leave messages for the UI thread to say that a job is done for example and what the output is. It used to be that if you made a mistake and called some feature that I not thread safe it would be unreliable as to what would happen, it might work, it might work sometimes, it might crash. Nowadays xojo does a better job of having anything not safe to call from a thread fail immediately and always with an error saying you can only do that from the main UI thread.

**HOW TO MORPH THE TEST PROJECT INTO A STARTER OF YOUR APPLICATION**

To use this project as a template to make your own bot, it works best to rename XojoTestBot files and edit the refrences in them with some tool such as sed for linux/mac. For each file with a name starting with XojoTestBot edit it as follows. Note leave the files named with ICC\* alone, they don’t change. For example if you where goin to make a tetris bot and wanted to use the XojoTestBot as a starting point:

sed s/XojoTestBot/TetrisBot/ < XojoTestBot.xojo\_code > TetrisBot.xojo\_code

sed s/XojoTestBot/TetrisBot/ < XojoTestBot.xojo\_project > TetrisBot.xojo\_project

and so on for each XojoTestBot file except the resources file. Just copy that one.

I found trying to all the above renames from within the project to be more painful and required detailed knowledge about this library and Xojo.

**DATAGRAMS**

The usual flow is the user sends in command and the chess server sends back a response. The default format for the response is for a human to read. There is a more program friendly format known as a level 1 datagram. It holds:

* The CN\_ command number of the command done (see formats.txt for a list)
* The user doing the command, usually you. But if someone matches you for example it would be their handle
* The text of the command output in the language set for the current login.

Before login level1 can be enabled by entering level=1(end of line) at the login prompt before entering the user and password. After login level1 can be set with the chess server command “set level1 1”

There is another kind of datagram, a L2 or level 2 datagram. Those can arrive at any time. They notify the user of some current status. See formats.txt for a list of DG\_ level 2 datagrams. Before login the level 2 datagrams can be enabled with a level2settings=. See the send\_login function in XojoTestBot\_Hub for some typical settings for logging in.

For which datagrams are set at login see the startup method in XojoTestBot\_Hub. Some datagrams can generate a huge amount of data, enough to overflow your output buffer in the chess server and cause data loss. It is best to limit the number of datagrams turned on before login. After login bots such as tomato typically send themselves a tell like boot1, boot2, etc. When they receive such a tell they know that boot phase1 has finished and they go on to phase 2 etc. Each phase can turn on another datagram to spread out the output load. Also some users have a privilege to have larger input and output buffers for handling more data. The tomato bots (which are all TDs, tournament direcitors) do the following during startup:

set tcp\_input\_size 60000

set tcp\_output\_size 60000

If you receive a control-O character you are not expecting from the chess server it means that your output buffer overflowed and some data was lost. Try increasing your output buffer size or decreasing the amount of data you having sent to you.

**WHERE DO I PUT MY GLOBAL VARIABES?**

You may notice that in the library and sample program there are few global variables. In the library there is an ICC\_Hub object. The hub receives information about events happening and passes them on to interested objects. Each instance of the hub is tied to a particular connection. This allows talking to multiple chess servers (main and queen) at the same time. It also provides a clear place to store data that will be cleaned up when a connection is closed / drops.

Global variables are somewhat a thing of the past, from a time when a program ran from the command line. Nowadays much of the data that used to be global is now per connection data. For example, for a typical bot that has a connection to main and one to the queen chess server the data associated with each connection would be stored separately. If this model works for you thing you are looking at is ICC\_Hub. However files with names starting with ICC are all library files that should not be modified. The custom version for your project is XojoTestBut\_Hub (which maybe you renamed to say PongBot\_Hub if you are making a pong bot). If you add a property to that Hub file it will have the same lifetime as the connection. So if you connect to main and queen you will get two instances of that Hub file.

That said there are times when you really do want a global. For example if you where showing stats on users logged in to all ICC chess servers. There are two ways to do that. One is to add a property to the App object. For example the dg\_map property of the XojoTestBot app subclass. The other is to create a new module then add a property to that new module. Such a module property has a global lifetime for the life of the program run.

**HOW DO I WRITE A PROGRAM IN XOJO?**

If you visit the xojo.com website and google for xojo related questions there are many resources for getting started in xojo. Also, there are many sample programs that come with XOJO. Unfortunately, if you are using the free version of XOJO those samples are all in a binary format. I prefer to read a print out or continuous listing of all the parts of a program which doesn’t seem to exist for the free version.

**TEXT VS STRING**

A string is an old data type. Is is an array of characters. A text object is designed to do the same thing as a string, but in addition to the data it also hold character set info. XOJO is in the process of transitioning from string to text. Not all the functions are converted yet. Beware that the first index of a string is 1 while the first index of a text is zero. So if translating algorithms for a string to text beware of being off by one.

**COMING FROM A DIFFERENT PROGRAMMING LANGUAGE**

* Xojo was once called real basic. In most basics arrays get a default dimension and can be redimensioned. In Xojo a new array has zero dimensions. You can add rows to it or redim it.
* Xojo arrays start at 1, c arrays start at 0. The new text string object starts at 0. For arrays there is a lower and upper index property so you never need to remember.

**BEWARE OF YOUR AVAILABLE BANDWITH AND USAGE**

If you are using xojo to make a program that runs on cell phone that maybe has a 2G or 3G connection and you are turning on all the datagrams you can find it probably is going to get buffer overflows and not work. If you are targeting a low bandwidth environment I would suggest testing that during development from a windows/mac/linux machines. The pf firewall (from openbsd, but available in linux now) can setup bandwidth limited routing paths. I have found this useful when testing clients to see how they will handle in the boondocks.

**IF LOOKING TO SOLVE A PROBLEM READ ALL THE CHESS SERVER OUTPUT**

In a lot of modern clients the chess server output gets routed to update various status items in the user interface. If something is going wrong it can be useful to log then read each datagram received. And all the chess server output. Often there is some command that didn’t work quite right or gave unexpected output.

**ON OVERRIDING A FUNCTION/SUBOUTINE**

The various classes in this test bot may or must be overridden to make a version for your bot. For example the XojoTestBot\_Hub is a subclass with app specific customizations of ICC\_Hub. In such cases when you make a custom version of a method that is in both bots there are some cases to consider.

* For recieve\_L2 method the subroutine in ICC\_Hub does nothing. It is just a place holder so other ICC\_\* can type check and have prototype to call. In this case the receive\_L2 method in XoJoTestBot\_Hub replaces the one in ICC\_Hub
* In other cases your custom routine may want to do some pre processing then call the default routine. If you are in routine foo then SUPER.foo would call your parent class. Likewise you may want to call the parent class first then do some work afterwards. In general if you look at the parent routine and it is not an empty place holder you probably want to call it. Some purists always call the parent routine even if the current implementation does nothing.

XOJO VS C++

If you are coming to xojo programming from c++ there some things the examples and docs may not make clear:

* Xojo object are like a pointer to a c++ class. The maid difference is that there is no structure assignment. If you want to copy an object with 5 fields you get to copy each field
* There is no xojo clone object (ok well that was a smalltalk thing, at least it solves the should clone be shallow or deep debate)
* Xojo arrays and string indexes generally start at 1 not zero. But the new text object to replace string seems zero based. There methods on arrays to get the upper and lower bounds so it is best not use 1 or 0 when indexing
* Xojo uses interfaces like Java rather than c++ multiple inheritance