Logging with pLogger

Paul Newman

March 13, 2007

1 Logging - pLogger

The pLogger process is intended to record the activities of a MOOS session. It can be configured to record a fraction of or every publication of any number of MOOS variables. It is an essential MOOS tool and is worth its weight in gold in terms of post-mission analysis, data gathering and post-mission replay.

The configuration of pLogger is trivial and consists of multiple lines with the following syntax:

$$\texttt{Log} = \textit{varname} \ @ \ \textit{period} \ [NOSYNC], [MONITOR]$$

where *varname* is any MOOS variable name and *period* is the minimum interval between log entries that will be recorded for the given variable. For example if *varname*=INS_YAW and *period* = 0.2 then even if the variable is published at 20Hz it will only be recorded at 5Hz. The optional NOSYNC flag indicates that this variable should not be recorded in the synchronous logs (see section 1.2). The optional MONITOR flag tells pLogger to send a notification if this variable isn't logged at least every 10 seconds. The notification occurs under the MOOS_DEBUG variable. If you are running iRemote (which subscribes to this variable automatically) you'll see a warning printed to the screen. This can be pretty useful when runnning a complicated system and you really do want notification that an important variable isn't being logged (probably because the process producing the variable is kaput in some sense.)

1.1 Logging Session

The logger supports the notion of Logging Sessions. For each log session the Logger will create a new directory and place all the logged files within that directory. These are typically (see later sections) an alog file a slog file a system log file, a copy of the mission file (moos file) and if applicable a hoof file (if pHelm is running). A new log session can be created by writing the variable LOGGER_RESTART to the MOOSDB or if you are using iRemote by pressing shift-g.

1.2 Log File Types

The logger records data in two file formats - synchronous ("slog" extensions) and asynchronous ("alog" extensions). Both formats are ASCII text – they can always be compressed later and usability is more important than disk space. The two formats are now discussed.

1.2.1 Synchronous Log Files

Synchronous logging makes a table of *numerical* data. Each line in the file corresponds to a single time interval. Each column of the table represents the broad evolution of a given variable over time. The time between lines (and whether synchronous logging is even required) is specified with the line

`SyncLog` = *true/false @ period*

where *period* is the interval time.

If there has been no change in the numeric variable between successive time steps then its value is written as `NaN` . It is important to note that synchronous logs do not capture all that happens - they sample it. Synchronous logs are designed to be used to swiftly appraise the behaviour of a MOOS community by examining numeric data in a tool such as Matlab or a spreadsheet. The `MOOSData` Matlab script reads in these files and with a single mouse click can display the time evolution of any logged variable.

1.2.2 Asynchronous Log Files

Asynchronous logging is thorough. The mechanism is designed to be able to record *every* delta to the MOOSDB. The use of the period variable allows the mission designer to back off from this ultimate limit and record variables at a maximum frequency. The key properties of asynchronous can be enumerated as follows:

1. Records both string and numeric data
2. Records data in a list format - one notification per line
3. Entries only made when variable is written

Asynchronous log files are designed to be used with a playback tool (for example `uPlayback` or other purpose-built executable). Although the handling of strings and numeric data adds a slight overhead to such a program's complexity the utility gain from being able to slow, stop and accelerate time during a post-mission replay/reprocessing session is simply massive.

1.2.3 Mission Backup

Simply having the *alog* and *slog* files is not enough to evaluate the mission. One also needs the things that *caused* the data to be recorded, namely the *.moos Mission file and the *.hoof file (if Task redirection was used). To this end the `pLogger` process takes a copy of these files and places them (name appended with a time stamp if desired) within the logging directory. The files extensions are renamed to **._moos* and **._hoof* respectively.

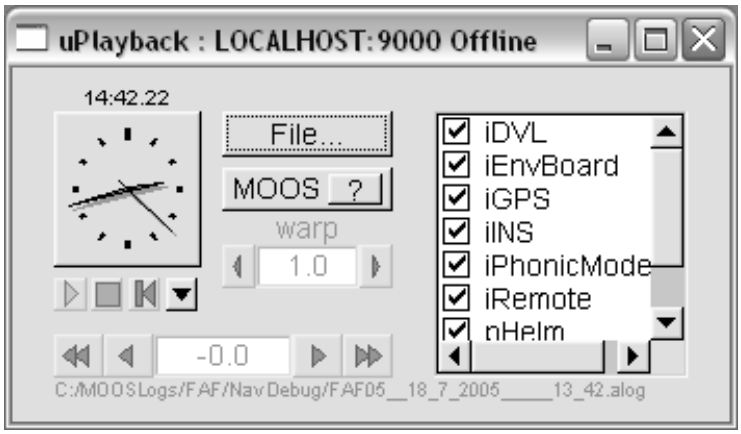

Figure 1: A screen shot of `uPB` - a cross platform "alog" playback tool

2 Replay – `uPB`

There is a FLTK-based, cross platform GUI application that can load in *alog* files and replay them into a MOOS community as though the originators of the data were really running and issuing notifications. A typical use of this application is to "fake" the presence of sensor processes when reprocessing sensor data and tuning navigation filters. Alternatively it can be used in pure replay mode perhaps to render a movie of the recorded mission. The GUI allows the selection of which processes are "faked". Only data recorded from those applications will be replayed from the log files. There is a single class that encapsulates all the replay functionality - `CMOOSPlayback` . The GUI simply hooks into the methods exported by this class. The GUI is almost self documenting - start it up and hold the mouse over various buttons.

A client process can control the replay of MOOS messages by writing to the `PLAYBACK_CHOKE` variable add writing a valid time in the numeric message field. The Playback executable will not play more than a few seconds past this value before waiting for a new value to be written. In this way it is possible to debug (halt inspect and compile-in-place etc) at source level a client application using replayed data without having the playback rush on ahead during periods of thought or code-stepping.