NCCC Log processing

2012

# Log capture / submission

The overall expected input file format is a Cabrillo V3 file. Most logging programs produce this file format. Several additional header fields have been used for CQP. See appendix X for a description of the Cabrillo format expected.

Other forms have been accepted: excel using CQP prescribed template, paper – usually transcribed to excel, then converted to Cabrillo. Unusual formats have sometimes been accepted and converted to Cabrillo – ADIF.

ADIF usually isn’t acceptable because logging programs don’t provide enough information to use the log.

Any method that gets Cabrillo files onto a linux based file system is acceptable – web page, email, floppy disk, dropbox, telepathy. The end result of log submission is a file.

# Post processing

Appendix X describes the format of an expected Cabrillo file. Due to the nature of logging software development, the contents of the files needs post processing. This is also known as a normalization process.

Post processing involves two major kinds of activities, and is done in phases.

First phase is cabrillo record and header validation, correction, and normalization. Each file is read, examining the header, then each QSO record.

Validation items are:

Correction and normalization items are:

At then of the first phase is a set of files which can be processed by green – all coding errors are gone, all field values are normalized and validation is complete. QSO matching can be done with this set of files, without concern for mismatches due to incorrectly formed content.

The second phase finds and resolves issues which can only be determined by attempting to match QSOs between submitted logs. Examples: sent QTH says Santa Clara (SCV), received QTH says Los Angeles (LANG). Errors of these only become evidence when many files are compared. If every log shows LANG, except the sender log, then it’s reasonable to assume the sender QTH is wrong.

The result of this activity is corrected logs (headers, QSOs), or logs that are declared unstable – in some manner (Wrong sent number, etc).

This phase requires multiple passes – as issues are resolved, new issues will be “unmasked” to be fixed.

# Green

## Input expectations

## File format

## Matching algorithms and techniques

Green performs a match on each and every QSO in a given Cabrillo file. The fields in each QSO are:

* sentCallSign
* sentQTH
* sentNR
* receivedCallSign
* receivedQTH
* receivedNR

Other fields, but, not as critical to matching:

* Date
* Time
* Band
* Mode

Match results can be one of:

* Fully match – all fields agree, within tolerances
* Bad receivedCallSign – corrected
* Bad receivedQTH – corrected
* Bad receivedNR – Corrected
* Any combination of bad,corrected
* Unique – possible list of callsigns
* Unique – high received number, possible list of callsigns
* Can’t determined (“X”) – punt to manual checking/matching

### Techniques and algorithms

Green processes all files in a directory. It selects a file in glob order, as determined by the OS – windows XP. Each QSO in a file is examined in order and a match process begun. Processing is always from the perspective of the current, or *sender* file. Processing progresses thru each QSO in each file in order. No recursion, or backup is implemented.

Matching begins with extracting the receivedCallSign. This callsign is used to form a file name for a log that might have a corresponding QSO – no surprises here. The root call forms the file name, then Green uses the receivedNR to locate a matching QSO in this file.

The QSO indexed by the receivedNR. The remainder of the fields are checked. If everything matches, within tolerances, then the QSO line is re-written back to the “sender” file with green flags of (GP, checked, matched, okay).

If not all fields match, then green begins a series of checks.

If any of Band, mode, date, or time don’t match, but, all the other fields do, then the QSO is matched, with a status of band/mode errors.

If the callsigns match (both sent and received), but any of these don’t match – receivedNR, receivedQTH, *from the sender file*, then the QSO is accepted, and the QSO is written out reporting the errors found.

If the callsigns don’t match, then the current file receivedCallsign is checked against the sentCallSign in the *other* log. If there is a match, then the other log busted the sender callsign. QSO is accepted, written out, with the information of the busted callsign.

If the reverse is true – receivedCallSign doesn’t match, but, the other log matches to this logs sentCallSign, then the sender busted the call – QSO is written out with the error, and the correct callsign.

The above two tests also require the sent, and received numbers, and QTHs match.

If the callsign at the indexed QSO in the other log doesn’t match, and no other major information matches, then this QSO is busted. A search is now initiated for this QSO. Several techniques are used. All log files search, N+1 callsign checking, unique table.

Each log in the directory is read, examining each QSO for a match in the receivedCallSign field. On each, or any match, the rest of the major fields are compared. If a matching QSO is found (all match but sending files receivedCallSign), Green has found the corresponding QSO. The QSO is written out, marking the QSO as wrong call sign, and the correct callsign written.

The above all file search process is sped up by first performing an N+1 examination of the callsign, and then checking those files first. For example: N6DQ busted as W6DQ (prefix is off by one).

If not QSO is found, then the QSO is either 1) with a station that didn’t submit a log; 2) Is a bust that we can’t figure out.

Green maintains a list of all the callsigns heard in the contest – the “uniques” file. If the receivedCallSign is in the uniques file, then the receivedNR is examined. If it’s high then the QSO isn’t accepted as unique with a highNR. If it’s low, then the QSO is accepted as unique.

A list of possible callsigns is also written out, based on N+1 processing of the receivedCallSign.

## Checked file format

## Post checking scoring and reporting

# Report generation

# New development environment

A few thoughts on a new set of software and development environment.

1. The current set of tools are all written in Perl. Should we stay with that, or move to Python.
2. Green does matching with a file system incremental technique. WA6O approached the problem as set theory, and loaded all the QSOs in a database (well within current technology). This is a different way to process. Rather than consider each QSO only from the “sending” perspective, each QSO could be considered part of set. More accurately, a SQL query could be written to retrieve a specific set of QSOs – i.e. all QSOs that have a receivedCallSign matching “K6TD”. Processing could then be done on that basis.
3. Using a SQL database makes coding much easier, as code to match is strongly separated from code to parse the files.
4. Stuff all the code in a GIT repo on a central server, that is backup up, and snapshotted.
5. Two issues exist with the current green processing flow:
   1. No forward info can be written – when manually checking a log, if the checker discovers a problem in another log, there isn’t any way to forward that info.
   2. As each log is manually checked, QSOs are re-written with updated status. This info will be lost of logs are refreshed from a central repository.
6. Have some way to accept in a green log. Use the info in the log to do subsequent processing.