**CQP 2012 Log Processing**

**Process & Strategy**

**K6TU – Draft 1.0**

**8/29/2012**

**Introduction**

This is a design note to capture the CQP 2012 strategy for processing logs – from submission to scoring. It expands on a document [1] written by K6TD that serves as input to this note.

An issue that needs resolution is highlighted in RED.

A resolved issue or change in prior practice is highlighted in GREEN.

**Log Process Flow**

1. **Submission**
2. We require electronic logs to be submitted in Cabrillo V3. Electronic logs NOT in this format will be rejected back to the user via an email response.
3. Submission is made either via email or via a web form. Logs arriving via the web form will capture additional information such as contest overlay categories (YL, Youth etc), operating location and club affiliation. This additional information will be appended to any supplied Cabrillo header records as a new X-CQP formed header record and added to the header records in the log. The amended log will be forward via email to the same email account as regular CQP logs ([logs@cqp.org](mailto:logs@cqp.org)).
4. **Capture**
5. Logs arrive for capture via email.
6. *Assumes N5KO contest robot or similar;* Each email is inspected for a Cabrillo log by looking for all text between a START-OF-LOG and END-OF-LOG record in the email. Any log found (expected to be only one) is written to a file.
7. The filename of the incoming log is of the form: CALL-timestamp where the timestamp is in the format YYYY-MM-DD\_HH:MM of the arriving message.
8. Each file is written into two locations – a master directory to hold unaltered log files AS SUBMITTED and a directory for processing by the contest management software.
9. The following required information is extracted from the log header records:  
   Power level - low or high

Operation category - Single, Multi/Single, Multi-Multi

Claimed score

Club

Operators

1. The following information is extracted from the QSO records;

Count of the number of QSOs

Transmitted location (QTHs)

Sent Callsign (CALLs)

1. A summary of this information captured is sent back to the submitter by email to notify them that the log was accepted and proving an opportunity for them to resubmit in case of errors.

**Issue: How much pushback do we make to the user? For example, the contest robot scans the QSO records to validate their format and information. In the case of malformed records, the user is sent back a note that says x QSOs accepted and y rejected together with an explanation of why. This would make our lives much easier but do we do it?**

**This would be the place to address invalid sequence numbers, incorrect dates/times, busted multiplier definitions etc.**

**We can address a significant number of these issues by post processing BUT if a log is re-submitted, the clean up will have to be done again.**

1. Header data from the log file is extracted and captured in a database. The database is the source of record for each participant’s entry into the contest and ultimately serves to drive the generation of the results table. At this stage, the participants CALL, QTH, IN CA, CATEGORY, POWER, OVERLAY STATUS is all captured together with their claimed score. A column (or more as necessary) holds final score, time to 58 etc.
2. QSO records from the log file are extracted and input to the normalization phase via a file with the same name as the log file.

**3.0 Normalization**

1. Each submitted set of QSO records is validated for basic issues;
2. Date and time within contest
3. Sent call and QTH are consistent with header and/or other log records
4. All serial numbers are NOT 59 or 599
5. All CALLR, NR and QTHR are present and consistent with CQP abbreviations. Non-conformant representations that have historic (we’ve seen this before) aliases are mapped to the CQP abbreviation.
6. Missing CALLR are mapped to XX0XXX.
7. Missing NR are mapped to 99999.
8. Missing QTHR are mapped to XXXX.
9. Date and time of each QSO record increments throughout the contest.
10. With enough other log data, validate the offset between this participants clock and known ‘good’ clocks in other logs.
11. Any log failing the validation criteria is flagged for human intervention.
12. A log that passes validation is inserted into the master QSO table. In the vent that the log has been resubmitted, all records for this CALLS (i.e. the sender) are deleted from the database and replaced with the latest submitted set.
13. Each QSO record in the table is marked with the log name from step 2 so we have an audit trail of which records are included in the log analysis.

**4.0 Log record matching**

1. The process of matching log records begins as soon as we have more than one log initially with the list of unique calls from 2011.
2. Log matching is run against the set of all QSO records in the QSO table by applying a set of rules.
3. Each rule maps the QSO record to ONE of the following subsets;

OK All required items match (full credit)

OKA All required items match but sender has wrong band

OKB All required items match but sender has wrong mode

D1A All match except CALLR – correct CALLR in diag field

D1B All match except NR – correct NR in diag field

D1C All match except QTHR – correct QTHR in diag field

D2A All match except CALLR + NR

D2B All match except CALLR + QTHR

D2C All match except NR + QTHR

U1 Unique CALLR, NR > 10 – Likely bust of a good call

U2 Unique CALLR in list of INVALID calls (note opposite of past)

BYE Known participant but don’t have other log for validation

X Human help required

1. After each application of the full rule set, the status X entries are identified to determine their origin and see whether we have an additional normalization step to add, have a terminally damaged log that has to be removed or identifies another rule that is required.
2. Steps a through d are repeated as more logs are added; once we have all logs in (Post 10/21/2012), we generate the new 2012 list of INVALID calls and re-run the prior steps until we declare victory (subset X is empty) or punt for log checker help.
3. For D1A, D2A and D2B, if a specific identified record can establish the bust CALLR, it is noted into the table. If not, a fuzzy match algorithm (Damerau-Levenshtein) is applied to find the shortest distance between the bust call and the list of known good calls. The nearest match is noted as the probable bust.
4. If the number of subset X members is small enough, one of Dean/Stu/Alan/Alan review and assign to one of the above subsets.
5. If the number is too large, log records are written back into individual log files with the historic GREEN information appended to each record and we divide logs across all the volunteer checkers for processing as in prior years.
6. Corrected logs are collected and analyzed to move status records into the appropriate subsets for scoring. At the end of this step, all records are in a valid, score-able subset.
7. The goal is to eliminate steps g through j.

**5.0 Scoring**

1. Once all records have been resolved the table is processed to identify duplicate QSOs. In the event of a DUPE, the DUPE with the lowest score is eliminated (for example, a D1x and a OKx would result in the D1x being discarded regardless of which occurs first. A DUPE is declared for same BAND and MODE as another log record entry for the same CALLS and CALLR (or identified CALLR).
2. Scoring rules are applied to map records in each subset into a valid score category.

**ISSUE: How do we scored unique subsets especially U1?**

1. A report is run to update the master entrant table from 2(h) with the calculated score from the matching process.
2. A report is run to calculate the time to 58 for each entrant (if made) and is written into the 2(h) master table.

**6.0 Results Generation**

1. A report is run to generate the final scoring report for publication. This requires a PDF version and also a CSV version for uploading to the CQP website and certificate production.
2. A report is run which generates the text for each wine label and plaque order (for extra credit!).

**7.0 Post process party**

1. NCCC buys the group of folks running this each (at least!) one bottle of wine.