TEQC (Translate/Edit/Quality Check)

Table of Contents

Last modified: 30 Sept 1997

Introduction: **teqc**
Section 1.

This document describes and serves as a tutorial for the main features of **teqc** (pronounced "tek"), part of which serves as a replacement for the original UNAVCO **QC** and **rnxget** programs. Although the capabilities of **teqc** extend beyond using just RINEX files, the most common type of data format that will probably be used is the RINEX format, either as input, or output, or both. Consequently a shorthand for the three basic kinds of RINEX formats is used throughout this document:

> OBS for RINEX observation data file,
> NAV for RINEX navigation message file,
> MET for RINEX meteorological data file.

Also, **teqc** currently only handles RINEX version 2 files (and also allows the non-RINEX version 2 OBS observables S1, S2, and LC), though eventually an option will be added to convert RINEX version 1 files to RINEX version 2 files, or directly process RINEX version 1 files.

If your primary interest is translating native binary formats to RINEX, go directly to the sections on translation (Section 15 and Section 16).

If your primary interest is editing, go directly to the section on meta-data editing/extraction (Section 9), or RINEX formatting (Section 8), or cutting (Section 13)/splicing (Section 14) operations.

If your primary interest is *qc*-ing RINEX or native binary data, go directly to the section on the qc mode (Section 11) of **teqc**.

UNAVCO World Wide Web Support & Contact
Section 2.

Information about **teqc** can be obtained over the World Wide Web at

```
http://www.unavco.ucar.edu/software/teqc
```

This tutorial document, for example, can be found at

```
http://www.unavco.ucar.edu/software/teqc/tutorial.html
```

To further aid the user, look at the FAQs ("frequently asked questions") on **teqc**:

```
http://www.unavco.ucar.edu/software/teqc/faqs.html
```

a **teqc** release log (including bug fixes for the next release):

```
http://www.unavco.ucar.edu/software/teqc/log.html
```

and a **teqc** bug report:

```
http://www.unavco.ucar.edu/software/teqc/bugs.html
```

Please contact Lou Estey at UNAVCO (email: lou@unavco.ucar.edu; tel: 303-497-8036) for other questions, unresolved problems, or bug reports. (A few kind words wouldn't hurt, either.)

Types of Data
Section 3.

A. RINEX Data Files

RINEX version 2 files are the default type of files that **teqc** is expecting to process and/or produce. In order to have a valid RINEX version 2 file, the file must conform to the specifications in the document "RINEX: The Receiver Independent Exchange Format Version 2" available from the University of Berne, though **teqc** also allows the non-RINEX version 2 GPS observables S1, S2, and/or LC (a.k.a L3) in RINEX OBS files. A minimum set of header records must be present for each file. These are the non-optional header records specified in the RINEX version 2 document.

RINEX version 1 files can be read with **teqc** and are converted to RINEX version 2 files on output. Also, lowercase versions of the following header lines are recognized and converted to uppercase if output (e.g. **rinex version / type** is read and recognized and converted to **RINEX VERSION / TYPE** on RINEX output).

Each RINEX OBS file must have a header with header lines (starting at the 61st character in the line) that end with:

```
RINEX VERSION / TYPE   (must be first line)
PGM / RUN BY / DATE
MARKER NAME
OBSERVER / AGENCY
REC # / TYPE / VERS
ANT # / TYPE
APPROX POSITION XYZ
ANTENNA: DELTA H/E/N
```

```
WAVELENGTH FACT L1/2   (default values)
# / TYPES OF OBSERV
TIME OF FIRST OBS
END OF HEADER          (must be last line of header for version 2)
```

where valid information (i.e., format version = 2, file type = ’O’, satellite system = ’ ’, ’G’, ’R’, ’T’, or ’M’) must be present in the first line, the number and types of observations must be specified on the **# / TYPES OF OBSERV** record line, and default values for the L1 and L2 wavelength factors must be given. The rest of the fields in other header records can be blank if a descriptor string is expected or have some numerical value if a numerical value (even if it is zero) is expected, but these other header lines must be present with or without non-blank information. Observation data usually follows the **END OF HEADER** header record. (Note that RINEX version 1 does not have a **END OF HEADER** field, but has a blank line instead.)

Each RINEX NAV file must have a header with header lines (starting at the 61st character in the line) that end with:

```
RINEX VERSION / TYPE   (must be first line)
PGM / RUN BY / DATE
END OF HEADER          (must be last line of header for version 2)
```

where valid information (i.e., format version = 2, file type = ’N’, satellite system = ’ ’, ’G’, ’R’, ’T’, or ’M’) must be present in the first line. Ephemeris data usually follows the **END OF HEADER** header record. (Note that RINEX version 1 does not have a **END OF HEADER** field, but has a blank line instead.) (Note also: There are some inconsistencies in the way various groups are storing NAV data from satellite systems other than NAVSTAR GPS. Until a true standard is defined, there is no hope of writing a RINEX reader and writer for RINEX NAV files for these other satellite systems. The reader/writer developed for **teqc** uses a scheme self-consistent with the OBS RINEX file format.)

Each RINEX MET file must have a header with header lines (starting at the 61st character in the line) that end with:

```
RINEX VERSION / TYPE   (must be first line)
PGM / RUN BY / DATE
MARKER NAME
# / TYPES OF OBSERV
END OF HEADER          (must be last line of header for version 2)
```

where valid information (i.e., format version = 2, file type = ’M’) must be present in the first line and the number and types of observations specified on **# / TYPES OF OBSERV** header record line. Meteorological data usually follows the **END OF HEADER** header record. (Note that RINEX version 1 does not have a **END OF HEADER** field, but has a blank line instead.)

B. Trimble *.dat Download Files

Trimble *.dat files from the more recent receivers (ST, SE, SSE, SSi) are readable with **teqc**. Some *.dat data records have not been coded into **teqc** yet (e.g. the older GPS observable Records 0 and 7). However, **teqc** should be able to read the entire file and will report records which have not be coded yet. When these are encountered, please report this to Lou Estey. Efforts will be made to decode the older records once examples are located. Also, support for reading kinematic data and creating kinematic RINEX files has been added.

MES files are not required, but if they are present and you are using target file names (not stdin), then **teqc** will use the matching MES file for each target input file to help resolve certain meta-data.

The ION and EPH download files are not used by **teqc**.

C. Trimble RS-232 (real-time) Stream ("Binary Cyclic Printout Mode")

The Trimble RS-232 (real-time) binary data format from Trimble 4000 SE/SSE/SSi receivers is readable with **teqc**. Currently, only the record 55h with ephemeris information and record 57h (GPS observables) have been coded, though, again, **teqc** will report and skip other records. Please report any new records reported by **teqc** to Lou Estey.

D. Ashtech Download Files (B-, E-, and S-files)

The Ashtech download file format from Ashtech Z-12 receivers is readable with **teqc**. Normally, the Ashtech "smoothing" of the pseudoranges in not applied, but can be turned on.

E. Ashtech RS-232 (real-time) Stream

The Ashtech RS-232 (real-time) binary data format from Ashtech Z-12 receivers is readable with **teqc**. This includes the binary MBEN, PBEN, and SNAV records. Like with the Ashtech download format, the Ashtech "smoothing" of the pseudoranges in not applied by default, but can be turned on.

F. ConanBinary from TurboRogue/TurboStar receivers

ConanBinary data from the Turbo series of Rogue receivers is readable with **teqc**. (ConanBinary data from original Rogues are ordered by SV number, rather than by time, and will not be read correctly with **teqc**).

G. TurboBinary

TurboBinary data is readable with **teqc**. This includes data with normal-rate data, high-rate (50 Hz) data, and the so-called "31-second" data containing LC data. Extraction of the LC data, however, is not automatic (since it is not standard RINEX version 2) and you must use the **-O.obs** option to specify the LC data as one of the observation data types.

H. Texas Instruments 4100 GESAR, BEPP/CORE, and TI-ROM Formats

Binary data from the TI-4100 can be read with **teqc**, though the code for all records types has not been tested (since examples of certain record types have not yet been encountered in use to date). These translators are being included primarily to read legacy data. To date, **teqc** can read what has been called GESAR and/or BEPP/CORE data (can be a mixture) or can read the original TI-ROM format. Depending on the record types in the data, it is possible to extract not only P1/CA, P2, L1, and L2, but also signal-to-noise (S1 and S2) and Doppler (D1 and D2). (So far, only Record 1 of the TI_ROM format has been encounterd in actual files, which is sufficient to write RINEX OBS files.)

I. Canadian Marconi Binary Format

Teqc can now read the Canadian Marconi binary format, mainly for the CMC Allstar OEM (CMT-1200) receiver. To date, this includes record IDs 22, 23, 65, and 126--sufficient to write RINEX OBS and NAV files.

I. Rockwell Zodiac Binary Format

Teqc can now read the binary format used in the Rockwell Zodiac receivers. To date, this includes message (record) IDs 1000, 1002, and 1102--sufficient to write RINEX OBS files. (There appear to be no records in the Rockwell Zodiac format which contain SV ephemerides.) Currently, the message checksums are being ignored by **teqc**.

J. Future Plans

Support is being considered for reading all the older records of Trimble *.dat (especially Records 0 and 7), the Leica DS file format, legacy MiniMac data format, the format for the Motorola Oncore receiver, and the Trimble Standard Interface Protocol (TSIP).

Basic Modes of Operation
Section 4.

There are three different basic modes of operation of **teqc**:

- translation
- editing: meta-data extraction and/or editing, formatting, cutting and/or splicing
- quality checking (qc)

Any of these modes can be used by themselves, or in combination with one another. For example, some of the ways that you might use **teqc** are:

- check a RINEX file or files for compliance with RINEX version 2 specification; for example, missing non-optional header fields are identified
- modify (edit) any existing RINEX header fields in a RINEX file and output the resulting edited RINEX file
- quality check a valid RINEX OBS file or files, but without a RINEX NAV file or binary ephemerides (qc "lite" == no position information)
- quality check a valid RINEX OBS file or files using ephemerides data in a valid RINEX NAV file or files (qc "full" == position information possible)
- cut (specify a sub-window of time) and/or splice two or more RINEX files
- translate (convert) certain native binary formats (e.g., Trimble *.dat) to RINEX OBS and/or NAV files

These modes of operation work alone or in concert with one another. As an example, a Trimble binary stream can be translated to RINEX OBS and NAV files; have empty header fields in the OBS file (such as the MARKER NAME) filled in; have the stream *qc*-ed, explicitly time-windowed, and auto-switched from *qc*-lite to *qc*-full when enough satellite ephemerides are encountered in the data stream, all in a single **teqc** run.

It may also be helpful for the first-time user to be aware that:

- A minimal number of assumptions have been made about the file names that **teqc** uses. Essentially the file names can be any valid name for the OS, except that no input file name can start with a ’-’ or ’+’ character, and names with whitespace (like spaces) are probably best avoided. The Berne-recommended naming convention for RINEX files, though not necessary for **teqc**, is quite acceptable and can be readily used on the command line or in scripts using **teqc**.
- In general, **teqc** is design-ready not only for NAVSTAR GPS data, but also GLONASS data, NNSS Transit data, or any other future system that may become part of the RINEX standard. Just the details need to be written into the code as they become available.
- **teqc** is 100% non-interactive; it will not query the user for input or to find out if something is OK. Your may receive a "Notice", "Warning", or "Error" to stderr. If something is wrong (ususally an "Error" or usage problem), **teqc** informs the user and terminates.
- In general, **teqc** does not use hard-wired array sizes, but instead allocates and deallocates memory as needed. As long as your computer has enough computer memory, you should never run into array bound problems.

- **teqc** is conservative about memory use.

The basic design of **teqc** is command-line oriented, following the UNIX shell model. For the remainder of this document, it will be assumed that the user is using a UNIX OS and is familiar with basic UNIX commands. Documentation specific to other operating systems (e.g. DOS) will be included as the program is ported and tested on other operating systems.

Operating Systems and Hardware
Section 5.

To date, **teqc** has been tested by UNAVCO personnel and other users on:

Solaris (Sparc) 2.3 - 2.5.1
Solaris (Intel) 2.5
SunOS 4.1.1 - 4.1.3 (Sparc)
HP-UX 9.03, 10.0, 10.20 (PA-RISC platforms)
DEC Digital-UNIX OSF1 V3.2, V4.0
SGI IRIX 5.3
MS DOS 5.x (w/ 386/SX and higher)
MS Windows NT 3.51 & 4.0 (w/ 486/DX and higher)
MS Windows 95 (w/ 486DX and higher)
Linux (w/ 486DX and higher)

Support for other platforms will be included over time.

Standard Input, Standard Output, and Standard Error
Section 6.

Another basic design feature is the use of standard input (stdin), standard output (stdout), and standard error (stderr). Instead of a file as input, **teqc** can be in a pipeline accepting a RINEX format stream or binary data stream as stdin. Stderr is reserved for reporting problems that may occur any time **teqc** encounters something in any file or in stdin that it doesn’t like or understand. Stdout is used for dumping the ASCII product requested by the user consistent with the command line syntax. The output from **teqc** then has the following caveats at the present time: stdout and stderr must be able to be separated.

Consequently, the user is encouraged to use a shell that can distinctly separate stdout and stderr. For UNIX, this includes the Bourne shell (**sh**) and the Korn shell (**ksh**). For UNIX C-shell (**csh**) or for **DOS**--which do not allow you to direct stdout and stderr separately to different files--an option of **teqc** (i.e. **+err** *filename*) can be used to send what would have gone to stderr to a separate file, to avoid unpleasantries when stdout and stderr would otherwise go to the same place. For the remainder of this document, it will be assumed that the UNIX user is using either **sh** or **ksh**, though the following examples should allow a user to easily use **csh** or a MS DOS shell to achieve the same results.

Though the rest of this tutorial assumes you will be using **sh** or **ksh**, you can easily use **teqc** with **csh** or a **DOS** to control stdout and stderr. When using **csh** or a **DOS** (or with any other shells), you can use the command options **+out**, **++out**, **+err** and/or **++err** to have **teqc** internally redirect stdout and/or stderr to specific files. Thus:

> **sh** or **ksh**:
>> **teqc** {*rest of command*} **2> err.txt** *[stdout to screen]*
>
> any shell:
>> **teqc +err err.txt** {*rest of command*} *[stdout to screen]*

or

> **sh** or **ksh**:
>> **teqc** {*rest of command*} > **out.txt** *[stderr to screen]*
>
> any shell:
>> **teqc +out out.txt** {*rest of command*} *[stderr to screen]*

should be exactly equivalent. You can even use **+out** and **+err** on the same execution of **teqc** to write to the same file name:

> **sh** or **ksh**:
>> **teqc** {*rest of command*} > **temp 2>&1**
>
> any shell:
>> **teqc +out temp +err temp** {*rest of command*}

Also, you can append to an existing file using **++out** or **++err**:

> **sh** or **ksh**:
>> **teqc** {*rest of command*} >> **out.txt 2>> err.txt**

any shell:
teqc ++out out.txt ++err err.txt {*rest of command*}

To append with either just stdout or stderr, or use **++out** or **++err**:

any shell:
teqc {*rest of command*} >> **out.txt** *[stderr to screen]*
any shell:
teqc ++out out.txt {*rest of command*} *[stderr to screen]*

or

any shell:
teqc {*rest of command*} **2>> err.txt** *[stdout to screen]*
any shell:
teqc ++err err.txt {*rest of command*} *[stdout to screen]*

In short, regardless of what shell you are using, there should a way to accomplish what you want for redirection of stdout and stderr.

General Concepts About Syntax
Section 7.

The general syntax for **teqc** is:

teqc {options} [*file1* [*file2* [...]]]

or (except for **DOS** shells):

teqc {options} < stdin

or similarly

... | **teqc** {options}

The file or files listed at the end of the command line, or stdin, are the targets which are to be processed for each execution of **teqc**. This is mentioned since other file names may be present in the options, but any files listed in the options are part of the processing configuration and are not considered to be targets of the processing.

Even executing just

```
teqc
```

(i.e., no targets present) is allowed; this returns **teqc**’s best guess as to which GPS week it currently is based on the CPU’s clock (but how good this guess is depends on how well the CPU’s time is set).

There is a mnemonic governing the use of **-** and **+** preceding each option:

leading **-**:
  indicates intent to input something (besides stdin or target file list),
  or indicates intent to turn off some option

leading **+**:
  indicates intent to output something, (besides stdout and/or stderr),
  or indicates intent to turn on some option

For some options, a leading **-** and **+** do the same thing. To get help, for example,

```
teqc -help
```

and

```
teqc +help
```

both dump an extensive usage to stderr. (Following the mnemonic, only the **+help** should give help, but why further confuse the issue when the user is requesting for help? For this reason, both work here.) Also, the RINEX header editing option flags work the same way: e.g., **-O.mo** is the same as **+O.mo**. You can either think of **-O.mo** as inputting some header information (the monument name), or using **+O.mo** as forcing what you want the output header information to be.

In order to try to detect possible command line input errors, target file names at the end of the command line starting with the character ’**-**’ or

’+’ are currently disallowed. Anything starting with the characters ’-’ or ’+’ is always assumed to be a command line option.

Sometimes, especially with new features (e.g., new translators) as they are being added and debugged, you may be inundated with warning messages going to stderr. Most of these can usually be suppressed by including the **-warn** option:

> **teqc -warn** {*rest of command*}

Using **teqc** for RINEX Formatting & Format Verification
Section 8.

Suppose you have three RINEX files (using the Berne-recommended naming conventions): **fbar0010.97o**, **fbar0010.97n**, and **fbar0010.97m** being your OBS, NAV, and MET filenames respectively. Let us suppose that you first wish to verify that your **fbar*** files are RINEX version 2 format compliant, i.e. that their format is such that they have all the bits and pieces in them to make them look like a RINEX file (but not that the information in them is necessarily valid). One way of doing this is to execute:

```
teqc fbar0010.97o
teqc fbar0010.97n
teqc fbar0010.97m
```

What you see being dumped to the screen is a re-processed RINEX version 2 format. All information originally in the target file will be retained in the output version (--and if its not, there’s a bug, so please report this).

Or you could execute:

```
teqc fbar0010.97o > temp0010.97o
teqc fbar0010.97n > temp0010.97n
teqc fbar0010.97m > temp0010.97m
```

in which case the re-processed RINEX files are redirected (stdout) and saved as a set of **temp*** files.

After doing the above three commands, it might also be instructive to do something like:

```
diff fbar0010.97o temp0010.97o | more
```

to see what some of the differences between the original target file and the re-processed file might be. If the original file were produced by **teqc**, you shouldn’t see any differences (--and, again, if you do, there’s a bug, so please report this).

If, in fact, there is some format problem with any of the above input RINEX files **fbar0010.97***, **teqc** will output stdout (either to the screen as in the first set of examples, or redirected to files as in the second set of examples) until the problem is encountered, at which point it will report the problem using stderr and terminate. **teqc** makes few guesses about what a RINEX file is supposed to be; if a file has a problem, a human or some other program must be used to fix it before **teqc** will proceed further.

Now suppose that you have a list of RINEX files that you wish to check for RINEX version 2 format compliancy, but don’t want to save any re-processed stdout. There are several ways to do this (e.g., in UNIX):

```
teqc fbar0010.97o > /dev/null
teqc fbar0010.97n > /dev/null
teqc fbar0010.97m > /dev/null
```

or, using the command line option **+v** (any shell):

```
teqc +v fbar0010.97o
teqc +v fbar0010.97n
teqc +v fbar0010.97m
```

or:

```
teqc +v fbar0010.97o fbar0010.97n fbar0010.97m
```

or (UNIX shell regular expressions):

```
teqc +v fbar0010.97[m-o]
```

or even (e.g., if using **ksh**):

```
teqc +v ‘ls fbar0010.97*‘
```

Essentially the **+v** option does three things:

1. shuts off the dump to stdout--so **teqc +v fbar0010.97o** should execute faster than **teqc fbar0010.97o > /dev/null**,
2. suppresses file "splicing"--so **teqc** understands that the input files are not necessarily of the same RINEX type, and
3. dumps a short message to stderr saying that each input file conforms to RINEX version 2 specifications at the end of the execution of the file, or a error message dumped to stderr if a problem was encountered.

teqc also examines the target(s) for proper time-ordering. For OBS and MET files, the time marker being examined is the observation and/or event epoch. For NAV files, three time markers are examined: the TOC ("time of clock") epoch, the TOE ("time of ephemeris") epoch, and the TOW ("time of transmission"). For OBS and MET files, time-ordering is required. For NAV files, a sanity check is performed on the three times for each ephemeris.

When inputting multiple target files of the same type, **teqc** looks to see if this time ordering remains sequential (though neighboring epochs of exactly the same time are currently allowed)--except for RINEX NAV files, where the information is sorted before it is used. For this reason, assuming that the data in **fbar0020.97o** really follows **fbar0010.97o**, executing

```
teqc +v fbar0020.97o fbar0010.97o
```

will result in an error message and program termination at the first observation epoch in **fbar0010.97o** (assuming no other errors).

Using **teqc** for RINEX Header Editing & Extraction; Introduction to Configuration Options and Files and the **teqc** Option Hierarchy Section 9.

As experienced RINEX file users know, any or all of the RINEX header information may be incorrect. In principal, any of this information can be modified by anyone using an editor for ASCII files, such as "ed", "ex", or "vi" on a UNIX OS, on a file-by-file basis (which, incidentally, highlights one of the most severe vulnerabilities of RINEX--ease of intentional or non-intentional data tampering).

However, it often occurs that the same type of information needs to be corrected on a large set of RINEX files and that the same corrected information needs to be placed in these fields on all the effected files. In this case, it may be easier to use the RINEX modification (editing) capabilities of **teqc**.

For example, suppose the monument (marker) name needs to be corrected to read "**the foobar site**" in the OBS file **fbar0010.97o**. This can be accomplished by executing

```
teqc -O.mo "the foobar site" fbar0010.97o > temp0010.97o
```

in which case the corrected file is now **temp0010.97o**. The **-O.mo** option specifies that the original monument name in any OBS file being processed is to be overridden with the string "**the foobar site**". Notice the double quotes on the command line encapsulating the string which contains blanks as white space. If you wished to change the monument name to just "**foobar**", you could execute

```
teqc -O.mo "foobar" fbar0010.97o > temp0010.97o
```

or just

```
teqc -O.mo foobar fbar0010.97o > temp0010.97o
```

Notice that in this case, there are no blanks in the replacement field (i.e., the new monument name), so the double quotes are optional.

There is a similar mechanism to change every header field in a RINEX file, except 1) RINEX comments, in which case the user can only append more comments in the RINEX header, and 2) the first header record of the RINEX file. Rather than list all the possible options here, it is easier to have **teqc** do it, by using the **++config** option with a RINEX target file:

```
teqc ++config fbar0010.97o
```

which will dump ***all*** the changeable header information and the current values (i.e., those in the OBS file **fbar0010.97o**) to stdout. A related option, **+config**, shows only those options which have been set by command line or other means. To see the difference, try:

```
teqc -O.mo foobar +config fbar0010.97o
teqc -O.mo foobar ++config fbar0010.97o
```

Basically, **+config** means: "show me what internal default option settings/values of **teqc** have been overridden"; **++config** means: "show me how all the **teqc** options are set, including the internal defaults".

Executing just

```
teqc ++config
```

will show the generic default configuration options of **teqc** (plus a few lines about the GPS week that went to stderr).

When executing just **teqc ++config** (i.e. no target files), the two options **-st[art_window]** and **-e[nd_window]** show the total possible time range that **teqc** is able to cope with--down to a resolution less than a femtosecond (1e-15 sec). The format for the arguments of these options are YYYYMMDDhhmmss.sss. Internally, 12 bits are used to store the value of the year, giving **teqc** the capability of dealing with 4096 years. Thus, with the internal calendar starting at 1980 A.D. (the GPS calendar started on 6.0 Jan 1980), **teqc**'s calendar won't become obsolete any time soon. No need to worry about the 1999 A.D. to 2000 A.D. transition here. Whether you specify it or not, **teqc** always works with a defined time window, where executing **teqc ++config** shows the maximum bounds on that time window. Note that you can't use times before 1 Jan 1980.

After executing **teqc ++config**, you probably noticed that some of the configuration options end with something like **[*stuff*]**. The characters in the brackets and the brackets themselves are optional material included only to make the option more understandable to the user; only the characters to the left of the leading **[** is used to identify the configuration option. Thus **-O.mo** and **-O.mo[nument]** and **-O.monument** and **-O.moe_and_curly** all mean exactly the same thing: the user is trying to set the monument name with the next argument. However, like the rest of the option flag name, only printable characters are allowed in the brackets; no white space is allowed.

You can redirect this configuration information to a file, which is called a configuration file:

```
teqc ++config fbar0010.97o > my_obs_config
```

This ASCII configuration file (i.e., **my_obs_config** in the above example) can be easily edited to contain (hopefully) correct information. The meaning of the various **-O** flags should be fairly obvious to anyone familiar with the RINEX OBS header fields:

-O.s[ystem]
satellite system (G = GPS, R = GLONASS, T = Transit, M = mixed)
-O.pr[ogram]
program used to create RINEX file
-O.r[un_by]
name of user of program
-O.d[ate]
date of program execution
-O.o[perator]
name of site operator (observer)
-O.ag[ency]

name of agency

-O.mo[nument]
monument (marker) name

-O.mn
monument (marker) number

-O.rn
receiver number

-O.rt
receiver type

-O.rv
receiver software/firmware version

-O.an
antenna number

-O.at
antenna type

-O.px[WGS84xyz,m]
approximate geocentric position in WGS84 cartesian coordinates, in meters

-O.pe[hEN,m]
antenna topocentric correction, in meters

-O.c[omment]
original header comment

-O.int[erval,sec]
sampling interval, in seconds

-O.st[art]
date & time of first observation epoch

-O.e[nd]
date & time of last observation epoch

-O.def_wf
default wavelength factors for L1 and L2 (see note on **teqc**'s handling of wavelength factors)

-O.obs[_types]
list of observables and the observables themselves in the data portion of the file

There are also a few other options that can be used to input information, but are never output with **+config** or **++config**:

-O.dec[imate]
modulo decimation of OBS epochs to # time units (seconds by default); **-O.dec 30** or **-O.dec 30s** or **-O.dec .5m** results in epochs nominally at 00 and 30 seconds being output; millisecond jumps *should* be accounted for automatically

-O.pg[eo,ddm]
approximate geocentric position in WGS84 geographic coordinates, latitude and longitude in decimal degrees and elevation in meters (this input is converted to WGS84 cartesian coordinates)

-O.sl[ant]
input vertical topocentric antenna correction as slant height, antenna diameter, and vertical phase center offset (E and N are assumed to be zero) (this input is converted to the cartesian topocentric correction h 0 0)

+O.c[omment]
append a new comment field (you cannot change existing comments)

-O.rename_obs
change the character designations of the observables in the **# / TYPES OF OBSERV** in the RINEX OBS header, but does not rearrange the data in the file; *use with care!*

-O.mod_wf
set the wavelength factors for a specific set of SVs different from the default wavelength factors (this will not be present in the RINEX OBS header as a **WAVELENGTH FACT L1/2** record; see note on **teqc**'s handling of wavelength factors)

-O.mov[ing] 1
force RINEX OBS antenna position to be in kinematic (roving) state initially (especially regardless of binary flags if doing translation); note that this option has an argument: 1 to turn on and 0 to turn off

There is a similar set of editing/extraction flags for RINEX NAV files, which you could obtain by examining: **teqc ++config fbar0010.97n | more**

-N.a[alpha]
ionosphere alpha parameters

-N.b[eta]
ionosphere beta parameters

-N.leap
leap seconds for UTC time model

-N.UTC
UTC time model A0 A1 t w

-N.d[ate]
date of program execution

-N.pr[ogram]
program used to create RINEX file
-N.r[un_by]
name of user of program
-N.s[ystem]
satellite system (G = GPS, R = GLONASS, T = Transit, M = mixed)
-N.c[omment]
original header comment

and likewise for editing you can also use

+N.c[omment]
append a new comment field (you cannot change existing comments)

There is a similar set of editing/extraction flags for RINEX MET files, which you could obtain by examining: **teqc ++config fbar0010.97m | more**

-M.d[ate]
date of program execution
-M.pr[ogram]
program used to create RINEX file
-M.r[un_by]
name of user of program
-M.obs[_types]
list of meteorological observables and the met observables themselves in the data portion of the file
-M.mo[nument]
monument (marker) name
-M.mn
monument (marker) number
-M.c[omment]
original header comment

and likewise for editing you can also use

+M.c[omment]
append a new comment field (you cannot change existing comments)

-M.rename_obs
change the character designations of the meterological observables in the **# / TYPES OF OBSERV** in the RINEX MET header, but does not rearrange the data in the file; *use with care!*

Let's return to the meta-data from the **fbar0010.97o**. Assuming that the configuration file **my_obs_config** from above now contains corrected RINEX OBS fields, you could execute (in **sh** or **ksh**):

```
teqc ‘cat my_obs_config‘ fbar0010.97o > temp001a.97o
```

or (using another **teqc** command line option)

```
teqc -config my_obs_config fbar0010.97o > temp001b.97o
```

The **-config this_config_file** specifies that you are inputting a configuration file called **this_config_file**.

There is an important difference between the last two example commands, though it should not be apparent at this point (i.e., executing **diff temp001a.97o temp001b.97o** should show that the two modified files are identical). Here the option hierarchical procedure used in **teqc** is introduced. To make full use of **teqc**'s capabilities, it is strongly suggested that the user eventually become familiar with this hierarchy.

The primary rule to remember when using configuration options is:

> **The first setting/value for a configuration option that is encountered is the one that's used** (later settings/values for the same configuration option are ignored)**; except for three special cases: inputting multiple config files, multiple NAV files for the qc mode, and additional RINEX comments.**

All that is needed now is to know the order in which configuration options are processed.

The first configuration options that are processed are the command line options, processed left to right. For example, execute:

```
teqc -O.mo "the foobar site" -O.mo foobar ++config fbar0010.97o
```

Here, there are two identical configuration options on the command line, **-O.mo**, to change the monument name, but two different arguments.

Which is used? The answer is the first one encountered, which in this case is the one to the left. To convince yourself, also try:

```
teqc -O.mo foobar -O.mo "the foobar site" ++config fbar0010.97o
```

In the example execution from above (using **sh** or **ksh**):

```
teqc ‘cat my_obs_config‘ fbar0010.97o > temp001a.97o
```

the **‘cat my_obs_config‘** turns the contents of the configuration file **my_obs_config** into command line configuration options, the first line of which (at the top of the file) becomes equivalent to the leftmost command line option, proceeding to the last line of which (at the bottom of the file) becomes equivalent to the rightmost command line option.

The next configuration options that are processed come from a special environment variable (if it exists), called **$teqc_OPT**. Actually, the executable looks for an environment variable that matches the name of the executable. So if you do **cp teqc my_teqc** and then use **my_teqc** as the executable, it will look in this case for the environment variable called **$my_teqc_OPT**.

Therefore, if you set **$teqc_OPT** (assuming **sh** or **ksh**):

```
export teqc_OPT="-O.mo foobar"
```

and then execute and compare

```
teqc ++config fbar0010.97o
teqc -O.mo "the foobar site" ++config fbar0010.97o
```

you will see that the monument name is set to **foobar** in the first case (using the environment variable **$teqc_OPT**) and **the foobar site** in the second case (using the command line option).

The next configuration options that are processed are all the specified configuration files. A list of these are formed in the following way. First, all **-config** command line arguments or any in **$teqc_OPT** are stored. Then (assuming, again, that our executable is still called **teqc**), the environment variable **$teqc_CONFIG** is looked for, which may contain the name or the complete path and name to another configuration file. If **$teqc_CONFIG** exists, the name it contains is appended to the end of this list of configuration files (if any) from the command line. Now all that is needed to is to find each of these configuration files (if they exist), which is done in the following way.

For each possible configuration file name, the name is first examined to see if it starts with a **/** character assuming a UNIX-style directory naming convention. (With the DOS-style convention, a **** is used; with the MacIntosh-style convention, a **:** is used.) If the name does start with **/**, **teqc** assumes that the configuration file name is absolute, i.e., it contains a full path preceding the file name. In this case, **teqc** will try only this path and name. If the file exists and can be read, it is processed as a configuration file; if it does not exist or cannot be read, **teqc** moves on to the next file in the list.

If the configuration file name does not start with a **/**, **teqc** assumes that the configuration file name is relative, i.e. it contains only a partial path and file name, or perhaps just the file name. At this point, **teqc** looks for an **$teqc_PATH** environment variable, set up the same way as other **$*PATH** environment variable, i.e. with a **:** separating the different paths. If this **$teqc_PATH** environment variable exists, each path in it is used as a prefix to the relative configuration file name. If a file is found that can be read, it is processed as a configuration file, and then **teqc** moves on to the next configuration file in the list. If it cannot be found or cannot be read, **teqc** tries the next path as a prefix, and so on.

If **$teqc_PATH** does not exist, **teqc** will always try the path **.**, i.e., the present working directory. If you use **$teqc_PATH**, you will probably want to make sure that **.** is one of your paths; **teqc** will not automatically include it for you in this case.

To summarize, the hierarchy for processing configuration options is:

1. process command line options first, left to right
2. any **-config** arguments are stored as a list of configuration files
3. if it exists, the environment variable **$teqc_OPT** is processed according to the same hierarchy as 1) and 2) above
4. if it exists, the environment variable **$teqc_CONFIG** is appended to the end of the list of configuration files extracted from the command line and the environment variable **$teqc_OPT**
5. the accumulated list of configuration files is then processed, first to last:
    - if the name of the configuration file is absolute, only that path and name are examined
    - if the name of the configuration file is relative: the list of paths in **$teqc_PATH** is used until a file is found and read or until all the paths in **$teqc_PATH** are exhausted; if **$teqc_PATH** does not exist, only the relative path **.** is tried.
    - then each configuration file is processed from left to right and top to bottom
6. any configuration option not set by the above has a default configuration option hardwired in the **teqc** executable, or in the case of inputting RINEX or native binary formats (via stdin or files), the original information is used, or if that information is not present, it is null.

If using **teqc** in a *qc* mode, a list of found-and-read configuration files will be included in both the qc short report segment and the qc long report segment.

This may seem like a mind-boggling set of unnecessary possibilities, but you can be assured that there is a reason. So far you have only been exposed to the **-O.*** options for modifying header fields in RINEX OBS files. There are over 20 of these options. There is a similar set of **-N.*** options for RINEX NAV files and **-M.*** options for RINEX MET files. For general use of **teqc** there are about a dozen different options and with the quality checking mode (qc) there are about 7 dozen different options. To have all of these options crammed into one file (which is certainly possible), creates a near-unmanageable file if you are interested in changing only one or two parameters. Each configuration file need not have all the options specified, owing to the way that **teqc** looks at each file: more as a set of command line options with random order than a rigidly formatted file. If ideas are not already starting to churn in your head as to how to take advantage of this flexibility, an example in the section on scripts will show you how to easily make use of this option hierarchy.

Furthermore, you don't really need to use ***any*** configuration files (perhaps just a few configuration options used as command line options) to have **teqc** produce reasonable output in most cases, as demonstrated by the first examples in this document.

Configuration Options and Command Line Options; What's the Difference?
Section 10.

It may appear at first that syntactically there is no difference between what is being called a configuration option for **teqc** and a normal UNIX command line option. For many of the options, there is, in fact, no difference. That's the beauty of the whole interface design!

However, for an important subset of **teqc** configuration options there is a critical difference. These are the options that have arguments which are characters strings that might encapsulate white space, like **-O.mo** discussed in detail above. Currently, these are only the **-O.**, **-N.**, and **-M.** options which will edit RINEX header character fields.

When you execute

```
teqc -O.mo "the foobar site" fbar0010.97o
```

the argument for the **-O.mo** flag is all the characters following between the double quotes starting with the **t** in **the** and ending with the **e** in **site**, i.e., **the foobar site**. However, if you were to put

```
-O.mo "the foobar site"
```

in a file called **my_config**, or (assuming **sh** or **ksh**) execute

```
export teqc_OPT="-O.mo \"the new foobar site\""
```

(and execute **echo $teqc_OPT** to make sure it looks like the aforementioned contents of **my_config**), and then execute:

```
teqc ‘cat my_config‘ fbar0010.97o
teqc -config my_config fbar0010.97o
```

or (now falling back to the environment variable **$teqc_OPT**)

```
teqc fbar0010.97o
```

then the first argument to the **-O.mo** flag is just **"the** for all three of these executions. What happened to the trailing " **foobar site**" or " **new foobar site**", and what’s that leading double quote ahead of **the**? The short answer is: that’s the way UNIX works. For the first and second cases using **my_config**, the next two arguments (after **"the**) are **foobar** and **site"** and for the third case using **$teqc_OPT**, the next three arguments are **new**, **foobar**, and **site"**. However, there is a special parser in **teqc** which recognizes the leading double quote on **"the**, strips it out, and then looks for a trailing double quote on this or a following argument. This parser essentially builds the argument that is supposed to follow the option flag **-O.mo**, all of which is transparent to the user in this case.

If all of this is transparent to the user, why do you have to know this? Well, it won’t be transparent all the time. Why? A few reasons. Suppose you wish to use ‘**the   site**’ as the argument to **-O.mo** (i.e., lots of extra whitespace). This will work as expected using the direct command line, but will not work if you use it in a configuration file or the **$teqc_OPT** variable. When doing this via a configuration file (using either the **-config** option or the **cat** method) or the **$teqc_OPT** variable, the multiple side-by-side spaces are gone; there is no way to reconstruct what was originally there. You will get ‘**the site**’ assigned to the monument name, as the parser in **teqc** stuffs one blank space in by default. Next, what if you want to assign **"the foobar site"**--now including leading and/or trailing double quotes--as the monument name? This can be done, but it will take some experimentation to get it right, depending on which shell is used, and there is no guarantee that it will look the same on the command line, in a configuration file, or in **$teqc_OPT**. This is an example of the classic character vs. metacharacter conflict. Life isn’t perfect.

Suggestion: avoid (if possible) multiple side-by-side spaces and leading and/or trailing double quotes in these character field arguments. If you do this, life will be simplier. If you must use multiple side-by-side spaces, do it via the command line only. If you must have a leading and/or trailing double quote, you are on your own. Besides, you can always resort back to an ASCII file editor.

Using **teqc** for Quality Checking (qc) Mode

Section 11.

A large portion of **teqc** is for quality checking or ***qc***-ing of satellite positioning data, mainly NAVSTAR GPS data, but with a few additional functions, it will eventually work for GLONASS and NNSS Transit data, and the code is generally adaptable to any other future satellite positioning systems should any become available.

To use **teqc** in a qc mode, try, for example:

```
teqc +qc fbar0010.97o
```

(assuming for the moment that the RINEX NAV file **fbar0010.97o** is ***not*** in the current directory). First, notice the **+qc** option: i.e., turn qc on. Next notice that there is only a RINEX OBS file as a target file. What you are doing here is running a *qc*-lite mode, i.e., devoid of any satellite positioning information.

Executing the above should produce something if **fbar0010.97o** is a valid and complete RINEX OBS file. Exactly what is does (at this point) should only depend on the default *qc*-mode settings in **teqc**--which, incidentally, should represent what a majority of users feel they want from routine qc processing. However, nearly all qc parameters can be effected by the appropriate configuration option. And, yes, you can look at the qc configuration options to see what exactly you can effect, by:

```
teqc +qc ++config 2> /dev/null | more
```

(Some notes: Notice how this differs from executing just **teqc ++config 2> /dev/null**. In short, with the **+qc** option, you are turning on the qc mode of **teqc** and with **++config** asking for a complete configuration option set, so now you get all the qc mode options following the general **teqc** options. Also, a pipe to **more** is used because there are a lot of qc options!)

Now let's look at what the execution of **teqc +qc fbar0010.97o** (perhaps followed by a pipe to **more**) finally produced. Assuming a successful run, the user should have noticed at least two things: 1) There should have been a set of characters going to the screen (actually, to stderr) during execution which looked like

```
qc lite>>>>>>>...
```

(with perhaps some other non-critical stderr messages mixed in). This is just a linear analog indicator of how **teqc** is marching through the observation epochs, where the length of the final indicator matches the "length" of the ASCII time plot (default of 72 characters). 2) There was a screen dump (actually, to stdout) of the ASCII time plot and some simple statistics at the end. (Note: This latter part to stdout is roughly equivalent to what was written to the *.*YY*S file by the original UNAVCO **QC** program.) Also, in the future, what is dumped to stdout may change, though this will be settable with one or more configuration flags.

Next, look at the result of

```
teqc +qc ++config 2> /dev/null | grep report
```

The result should be something like:

```
+l[ong_report]
+s[short_report]
```

If either of these configuration options do, in fact, start with a **+**, then you should have a **fbar0010.97S** file in your directory, which is your qc report file on **fbar0010.95o**. There are two possible segments to this report: a possible short report segment followed by a possible long report segment. You should have the short segment if your configuration says **+s...**, and it should be absent if your configuration says **-s...**. For now, the short report segment is identical to the information being dumped to stdout. You should have the long report segment if your configuration says **+l...**, and it should be absent if your configuration says **-l...**. (The long report segment takes the place of what used to go to stdout in the original UNAVCO **QC**.)

Next, look at the result of

```
teqc +qc ++config 2> /dev/null | grep plot
```

If the result is

```
+plot
```

then you probably have one or more plot files (in UNAVCO COMPACT format), e.g.:

fbar0010.ion if **+ion** was set (L2-ionospheric observable)
fbar0010.iod if **+iod** was set (derivative of L2-ionospheric observable)
fbar0010.mp1 and **fbar0010.mp2** if **+mp** was set (MP1 and MP2 observables)
fbar0010.sn1 and **fbar0010.sn2** if **+sn** was set (S/N for L1 and L2)

These files can be viewed graphically using UNAVCO's **gt** or **qcview** executables. Using a configuration option of **-plot** suppresses all plot files. You can also convert the L2-ionospheric observable and its derivative to relative changes in TEC (Total Electron Count == 1e16 electrons/m^2) and its derivative by using **+tec**.

Next, move or create **fbar0010.97n** in the your working directory and execute:

teqc +qc fbar0010.97o

or

teqc +qc -nav fbar0010.97n fbar0010.97o

Here you are inputting an additional file, either by default in the first example or explicitly by using the **-nav** configuration option followed by the name of a RINEX NAV file. You are now running **teqc** in a *qc*-full mode, i.e., satellite positioning information is present, and if enough information is present (satellite ephemerides, plus receiver P1, P2, or C/A codes) then a position for the antenna is attempted. There is no additional flag to use the *qc*-full mode. If you supply binary data or NAV files or if **teqc** automatically finds a matching NAV file name to the supplied OBS file name, you are using qc full; if you don't supply NAV files and there is no matching NAV file, you are using qc lite. If you are supplying binary data, there is no a priori way to determine if ephemeris records are present or not without reading the entire input first, so *qc*-full is assumed in this case.

The case-sensitive algorithm for building matching NAV file names from OBS file names is as follows, where *YY* represents two digits, e.g. *YY* == 97:

```
*.YYo   ->   looks for *.YYn
*.YYO   ->   looks for *.YYN
*.obs   ->   looks for *.nav
*.OBS   ->   looks for *.NAV
```

Other cases for this NAV file auto-matching can easily be added to the code.

During execution (which will probably take slightly longer than the *qc*-lite run), you should now see:

qc full>>>>>>>**...**

going to stderr (and, again, possibly other stderr stuff mixed in). Assuming a successful run, the general things that happen should be similar to what happened with the *qc*-lite run, expanded to include information about satellite position (like elevation) and antenna position. The biggest additional items, assuming a **+plot** configuration, are the additional COMPACT plot files:

> **fbar0010.azi** and**fbar0010.ele** (satellite topocentric azimuths and elevations as viewed from the antenna for every satellite for which an ephemeris was present in the NAV file)

One of the fundamental design criteria was to have **teqc** dynamically switch from a *qc*-lite mode of operation to a *qc*-full mode of operation on a per satellite basis as an ephemeris became available. This is especially important for analyzing real-time data streams. For the normal "post-processing" (i.e., files are available), the same scheme applies. If no ephemeris is available for a particular satellite from the supplies NAV file(s), but observations for that satellite are present, the observations are completely processed using the *qc*-lite mode, even though the *qc*-full mode is on in general. Appropriate indicators are supplied in report that this occurred.

Using **teqc** with Multiple File Input or File Names Including a **,** (comma)
Section 12.

Cases always arise where you want to include multiple files as input. Let’s look at some of these cases. First, there may be multiple configuration files:

```
teqc -config my_config_1 -config my_config_2 -config my_config_3 {rest of command}
```

which works just fine, and is exactly equivalent to:

```
teqc -config my_config_1,my_config_2,my_config_3 {rest of command}
```

or

```
teqc -config my_config_1,my_config_2 -config my_config_3 {rest of command}
```

or

```
teqc -config my_config_1 -config my_config_2,my_config_3 {rest of command}
```

Notice that a default comma delimiter **,** is used to separate the different configuration file names if they are used as a single argument after a

-config option flag. A comma delimiter was selected since it usually is not a shell metacharacter and is rarely used as part of a file name. If in some circumstance, a comma ends up being part of the name of a input file, then it will be necessary to use the **-delim** option before that file name is used. For example, suppose that the configuration file called **strange,one** existed and for some reason either you couldn't or didn't what to rename it to a name that did not include a comma, but you wanted to use it and you didn't know about UNIX **ln**. Well, to do it, you could use

```
teqc -delim= -config strange,one {rest of command}
```

Here you have changed the delimiting character to **=** (equal sign), so now **strange,one** is interpreted as a single configuration file name, instead of two called **strange** and **one**. Be forewarned, however: like most everything else, you only get to change the delimiter character once.

The same type of scheme works for inputting multiple NAV files for a *qc*-full run:

```
teqc +qc -nav fbar0010.97n -nav fbar0020.97n -nav fbar0030.97n {rest of command}
```

is identical to

```
teqc +qc -nav fbar0010.97n,fbar0020.97n,fbar0030.97n {rest of command}
```

Recall the ordering option hierarchy for configuration files: the files to the left will be processed first. For inputting multiple NAV files with the **+qc** option, **teqc** sorts the ephemerides of each SV so there really isn't any need for you to keep to any special ordering, though the initial setup is slightly faster if you adhere to a TOW time-sequential multiple NAV file input.

Currently, all the target file names at the end of the command line are considered to be part of the single execution of **teqc**. For example, what should you expect when executing this:

```
teqc +qc -nav fbar0010.97n fbar0010.97o fbar0020.97o fbar0030.97o
```

You turned on the qc mode with **+qc**. Since you are supplying a NAV file with **-nav** this will be a *qc*-full run. Assuming for a moment that each of the OBS files are for 24 hours (and no errors were encountered), what you ***will*** receive is a qc report on for the entire 3-day observation period starting with the first observation on 1 Jan 1997 and ending with the last observation on 3 Jan 1997. (The ephemeris information from **fbar0010.97n** may be a little out of date by the 3rd, but that's OK for this discussion.) What you will ***not*** receive (as **teqc** is currently implemented) is three separate reports and possibly three sets of plot files, i.e. one report (and set of plot files) for each of the supplied OBS files.

Windowing (Cutting) with **teqc**
Section 13.

As mentioned above in Section 9, **teqc** always windows the input data (somewhere between 1 Jan 1980 and 31 Dec 6075), though this is usually transparent to the user. There are eight different windowing strategies for you to be aware of; different information is supplied (or not supplied) to use each of these different strategies.

The nomenclature in the following table is as follows. A bracketed value is one determined by **teqc** from the target files (i.e., implied), whereas a non-bracketed value is explicitly supplied by the user (via a configuration option using the command line, **$teqc_OPT**, **$teqc_CONFIG**, or another configuration file). *start* refers to the argument of the configuration option flag **-st[art_window]**, *delta* refers to the arguments of the configuration option flags **+d...** or **-d...**, and *end* refers to the argument of the configuration option flag **-e[nd_window]**, and *dir* refers to whether a **+** or **-** was used with the delta configuration flag. The eight different windowing options are:

1. [*start*] [*end*] (user supplies nothing except target files)
2. [*start*] *delta* (*dir* == **+**) e.g. **+dh 7** for 7 hours from the start
3. *delta* [*end*] (*dir* == **-**) e.g. **-dm 60** for 60 minutes from the end
4. *start* [*end*]
5. [*start*] *end*
6. *start end*
7. *start delta* (*dir* == **+** or **-**)
8. *delta end* (*dir* == **+** or **-**)

where the missing value is computed by **teqc**. Let’s take a look at what these cases mean, and you’ll see that this is really simplier than it initially appears.

For case (1), probably the most common, the user supplies nothing but the list of target files. **teqc** then does a fast search of the target files to be processed to determine the start and end times. If you input a file type for which a fast search algorithm has not yet been written or use stdin as the target, then you must use explicit windowing (case (6), (7), or (8)). Currently, the fast search algorithm has been implemented only for RINEX OBS, NAV, and MET files. The start time will be the first observation epoch or event epoch (RINEX OBS or MET files) or the first TOE epoch (RINEX NAV files) in the first target file listed and the end time will be the last epoch in the last target file listed. (As mentioned above, if your target files are not listed in a time-sequential order, or if one or more of your target files are not internally time-sequential, something will run afoul later in the execution process and **teqc** will tell you about it.)

As an example, let’s return to:

```
teqc +qc -nav fbar0010.97n fbar0010.97o fbar0020.97o fbar0030.97o
```

Here the target files to be processed are **fbar00*0.97o**. The fast search looks at the beginning of **fbar0010.97o** to determine the start time of the window and looks at the end of **fbar0030.97o** to determine the end time of the window. No need you to worry about much of anything here.

For cases (2) and (3), the user supplies a **+d...** option or a **-d...** option with an argument. Currently, **...** could be **Y** for years, **M** for months, **d** for days, **h** for hours, **m** for minutes, or **s** for seconds. The leading **+** or **-** means "give me a positive time delta from the start" or "give me a negative time delta from the end", respectively. Let’s suppose, again, that the **fbar00*0.97o** files above are each 24-hours worth of data. Then the configuration option **+dh 6** together with the file **fbar0010.97o** would mean: "start at 00:00:00.0 1 Jan 1997 and end at 06:00:00.0 1 Jan 1997", i.e. a delta of +6 hours from the (implied) window start. The configuration option **-dh 6** together with the file **fbar0010.97o** would mean: "start at 18:00:00.0 3 Jan 1997 and end at 00:00:00.0 4 Jan 1997", i.e. a delta of -6 hours from the (implied) window end.

For cases (4) and (5), you supply explicitly either just a new partial or complete start time or just a partial or complete end time using the configuration option flags **-st** or **-e** with an argument. The argument for a complete start or end time is easy to understand; it has the format of **[YY]YYMMddhhmmss[.sss...]**, i.e.

```
19970229105937
970229105937
970229105937.000000
```

all mean "1997 Feb 29 10:59:37.000000", given that the base year is 1980 (see result of executing **teqc ++config**: base year of 1980 implies that all two-digit years are assumed to occur from 1980-2079 A.D.) Let’s suppose we return to our command

```
teqc +qc -nav fbar0010.97n fbar0010.97o fbar0020.97o fbar0030.97o
```

but want to start the time window at "1997 Jan 1 00:09:30.004". You could use the configuration option **-st 970101000930.004** and things will work fine, though a bit cumbersome. There is, in fact, another way to use the **-st** option by using the partial argument format, for example **-st 930.004**. Here, **teqc** recognizes that you are suppling only a partial start time (using minutes and seconds, in this case). It then gets the real start time as it would in case (1) or case (2) using the fast search algorithm, and then applies a mask overlay to the start time and inserts your partial start time (changing just the minutes and seconds, in this case). So the argument for **-st** or **-e** has a format closer to:

[[[[[[YY]YY]MM]dd]hh]mm]ss[.sss...]

For cases (6), (7), and (8), you are explicitly assigning the window of interest, and, again, partial arguments can be used for the start and/or end times. You can supply an explicit (partial or complete) start and end time, explicit (partial or complete) start time with a delta (a-la cases (2) and (3)), or an explicit (partial or complete) end time with a delta.

By using case (7), you can easily force your **teqc** processing to start at the same time of day and span the same length of time, regardless of the start and end times of the input target files. For example, suppose you wish to have a start time of 00:00:00.0000000 at each day and want exactly 24-hours worth of **teqc** processing. This is easily accomplished with **-st 000000 +dh 24**.

You might think that just about every conceivable type of window possibility has been covered. Well, not quite yet. You can even introduce holes in the overall time window during which you are not interested in anything. **teqc** skips over all input that occur during your specified window holes, hole boundaries inclusive. Here the configuration option **-hole** followed by a file name is used. The file named is an ASCII file listing the holes that you want in your time window. The format of the hole file for each window hole is always

[YY]YY MM dd hh mm ss[.sss] [YY]YY MM dd hh mm ss[.sss]

where only the presence of white-space as delimiters is needed. Thus, the file:

```
97  2 15 00 00 00  1997
   2 15 03 00 00.00
  97  2 15
06 00 00        97  2 15 09 00 00
```

is perfectly OK (though a little hard to read by humans). A more readable file (for humans) that does the same thing is:

```
97  2 15 00 00 00       97  2 15 03 00 00
97  2 15 06 00 00       97  2 15 09 00 00
```

Obviously, this is a listing of start and end times for each hole desired. How many holes can you list in a file? As many as you want (or until computer memory is saturated). How many files do you get to name with the **-hole** option for each **teqc** run? Currently, one, so make sure it includes all the holes you want for the execution. Also remember: each hole includes the exact start and stop time of the hole.

Splicing with **teqc**
Section 14.

Recall that executing the command

```
teqc fbar0010.97o
```

basically spews the contents of **fbar0010.97o** back out to stdout. Suppose you have the RINEX OBS files **fbar0010.97o** for 1 Jan 1997 and **fbar0020.97o** for 2 Jan 1997 and you want to combine them into a single RINEX OBS file. It would have been easy if the RINEX standard had been written so that two RINEX files could be simply concatenated to one another to produce a new valid RINEX file, a la the UNIX **cat** system command:

```
cat fbar0010.97o fbar0020.97o > oops0010.97o
```

But, alas, the RINEX standard does not allow this sort of obvious simplicity and thus the file **oops0010.97o** is generally useless.

However, **teqc** takes care of the RINEX-idiosyncratic boundary between the two files. Thus

```
teqc fbar0010.97o fbar0020.97o > good0010.97o
```

produces a valid RINEX file, **good0010.97o**, with an added comment at the boundary:

```
RINEX FILE SPLICE                                           COMMENT
```

Multiple files can be spliced together and any of them can be for any session length. However, the order (like always) must be time-sequential.

Receiver clock reset information is not carried across the splice boundary of RINEX OBS files. Thus if there are millisecond receiver clock resets in the first file OBS file, and the second OBS file has these millisecond resets initialized back to zero, there will be a n-millisecond receiver clock jump at the boundary of the OBS splice.

If desired, you can combine the cut and splice operations in a single command. Use any of the windowing options in combination with the splice procedure.

Translating from Binary with **teqc**
Section 15.

teqc is being enhanced to handle a number of binary input formats from various receivers. For now, **teqc** handles common formats for dual-frequency (L1 and L2) Trimble, Ashtech, AOA Rogue family (though mainly TurboRogue and TurboStar), and the Texas Instruments 4100 receivers; and common formats for single-frequency (L1) Canadian Marconi (Allstar OEM) and Rockwell (Zodiac) receivers. The general use of **teqc** for all formats is similar. You need to specify three things:

1. the general type of receiver
2. the general type of native binary format from this receiver,
3. what you are interested in extracting.

(The big-endian/little-endian problem of the different binary formats is automatically handled by **teqc**, so don’t worry about it.)

An option flag is used to specify the general type of receiver, and its argument is used to specify the other two pieces of information. For receivers:

- **-aoa** or **-jpl** specifies a Rogue/TurboRogue/TurboStar receiver
- **-ash** specifies an Ashtech receiver
- **-cmc** for a Canadian Marconi Corporation receiver
- **-rock** for a Rockwell receiver
- **-ti** specifies a Texas Instruments receiver
- **-tr** specifies a Trimble receiver

Support for additional receivers will probably added in the future:

- **-leica** for a Leica receiver
- **-mot** for a Motorala receiver

For Ashtech and Trimble receivers, two distinct types of data file can be collected. This can be either a download format, i.e. the B-, E-, and S-files for Ashtech or the DAT format for Trimble, or the data stream from an RS-232 port on the receiver. The download format is specified with a **d** argument and the RS-232 stream is specified with an **s** argument.

For the Rogue family of receivers, there is no distinction between the download formats and the data-stream formats. The distinction is whether the format is ConanBinary or TurboBinary. An argument of **cb** is used to indicate ConanBinary and an argument of **tb** is used to indicate TurboBinary.

For Texas Instruments receivers, currently only the TI-4100 receiver is supported, using either the GESAR and BEPP/CORE formats, or the orginal TI-ROM format. An argument of **gesar** is used to specify GESAR and/or BEPP/CORE; an argument of **rom** is used to specify TI-ROM.

For RINEX translation, the user can specify what type file is of primary interest; if none is specified, RINEX OBS is assumed. An **o** is appended to the data type (**d** or **s**) to indicate RINEX OBS extraction to stdout. Likewise, an **n** is appended to indicate RINEX NAV extraction, or an **m** is appended to indicate RINEX MET extraction.

Thus, the possible reader/translator options (currently) are:

- **-ash d** to read Ashtech download format (and dump RINEX OBS to stdout)
- **-ash dn** to read Ashtech download format and dump RINEX NAV to stdout
- **-ash s** to read Ashtech RS-232 format (and dump RINEX OBS to stdout)
- **-ash sn** to read Ashtech RS-232 format and dump RINEX NAV to stdout
- **-aoa cb** or **-jpl cb** to read ConanBinary format (and dump RINEX OBS to stdout)
- **-aoa cbn** or **-jpl cbn** to read ConanBinary format and dump RINEX NAV to stdout
- **-aoa tb** or **-jpl tb** to read TurboBinary format (and dump RINEX OBS to stdout)
- **-aoa tbn** or **-jpl tbn** to read TurboBinary format and dump RINEX NAV to stdout
- **-cmc b** to read Canadian Marconi binary format (and dump RINEX OBS to stdout)
- **-cmc bn** to read Canadian Marconi binary format and dump RINEX NAV to stdout
- **-rock z** to read Rockwell Zodiac binary format (and dump RINEX OBS to stdout)
- **-ti gesar** to read TI-4100 GESAR or BEPP/CORE formats (and dump RINEX OBS to stdout)
- **-ti gn** to read TI-4100 GESAR or BEPP/CORE formats and dump RINEX NAV to stdout
- **-ti rom** to read TI-4100 ROM formats (and dump RINEX OBS to stdout)
- **-tr d** to read Trimble DAT format (and dump RINEX OBS to stdout)
- **-tr dn** to read Trimble DAT format and dump RINEX NAV to stdout
- **-tr dm** to read Trimble DAT format and dump RINEX MET to stdout
- **-tr s** to read Trimble RS-232 format (and dump RINEX OBS to stdout)
- **-tr sn** to read Trimble RS-232 format and dump RINEX NAV to stdout

Coming soon will be:

- **-leica d** to read Leica DS download format

Suppose, for example, that the file **fbar.bin** contains the the time-sequential, real-time binary data records for GPS week 866, 11 Aug 1996 - 17 Aug 1996 from a Trimble SSE receiver. Then, execute

```
teqc -tr so -week 866 +nav fbar2240.96n fbar.bin > fbar2240.96o
```

Let's dissect the command line. First the **-tr** option flag tells **teqc** that the target file(s) are from a Trimble receiver. The argument to **-tr** is **so**, which tells **teqc** that the native format is the RS-232 data stream and that you want to send translated RINEX OBS to stdout. But the binary file **fbar.bin**, in general, is allowed to contain both record types for both GPS observations and ephemerides. The command line option **+nav fbar224.96n** tells **teqc** to dump any ephemeris information in **fbar.bin** to the RINEX NAV file **fbar2240.96n**; if there were no ephemeris records in **fbar.bin**, then **fbar2240.96n** will be empty after execution is complete.

If you had had a Trimble *.dat file, **fbar.dat**, the analogous command line would have been:

```
teqc -tr do -week 866 +nav fbar2240.96n fbar.dat > fbar2240.96o
```

Now, what about the option **-week 866** (or using an alternative format of **-week 96:224**)? By doing this, you are explicitly telling **teqc** that the observation data ***starts*** in GPS week 866; it may run on into GPS week 867 (or later), but **teqc** will track this. If you had executed the shorter command

```
teqc -tr so +nav fbar2240.96n fbar.bin > fbar2240.96o
```

during the week of 11 Aug 1996 - 17 Aug 1996, and your CPU system time had been set corrected, then **teqc** would have computed the GPS week and used that (after issuing a warning to stderr: recall executing just

```
teqc
```

is one way for you to find out what GPS week **teqc** thinks it is).

Why must the GPS week be specified? It turns out that there is no information in the Trimble RS-232 GPS observation records to indicate which GPS week it is from; this is ancillary information that must be recorded external to the contents of the observation records. (There ***is*** GPS week information in the Trimble ephemeris records, but there is no guarantee that there will be ***any*** ephemeris records in an arbitrary

RS-232 data segment, let alone an ephemeris record in advance of all observation records. A similar argument applies for Trimble *.dat files: there is no GPS week information in Trimble's DAT observation records--though the GPS week appears in other records which are usually in a *.dat file. Additionally, when using **teqc** with DAT files as target files--not stdin--**teqc** will attempt to find a name-matching MES file to help resolve the GPS week problem. But, again, there is no guarantee that a matching MES file is present.)

Incidentally, the GPS week can be supplied by several formats when using the **-week** option:

> **-week WEEK** (WEEK = GPS week, e.g. **-week 866**)
> **-week [YY]YY:DOY** (YY = year, DOY = day of year, e.g. **-week 96:228** or **-week 1996:228**)
> **-week [YY]YY:MM:DD** (YY = year, MM = month, DD = day, e.g. **-week 96:8:15** or **-week 1996:8:15**)

You can also use a **/** (slash) as the delimiter instead of a **:** (colon).

All or part of the RINEX header field **PGM / RUN BY / DATE** is filled in automatically by **teqc** during translation. The program field is filled in with the name of the executable (**teqc** in this case) and its current version number.

The date is filled in by a query of the system time, and we are assuming that the system time is set correctly. On UNIX systems, this date is UTC, which is then written to the RINEX file. On Microsoft systems, this date may or may not be UTC. For Microsoft Windows 95 and Microsoft Windows NT systems, the date should be set according to a specific time zone, or with a known offset between local time and UTC. For these cases, the date obtained should correctly be UTC.

For Microsoft **DOS** or Windows (or Windows 95 or Windows NT, in case **teqc** cannot determine the OS), **teqc** will query for the environment variable **$UTC_MIN_OFFSET**, which if set, should contain the numerical value of minutes that should be added to the system time to yield UTC. The switches between Daylight Saving Time and Standard Time will have to be done manually. If this environment variable is not set, the system time will be queried and put into the date field as "Lcl" = Local time.

Now examine the command line:

```
teqc -tr sn -week 866 +obs fbar2240.96o fbar.bin > fbar2240.96n
```

Here the argument to the **-tr** option flag is **sn**, i.e. your main interest is the ephemeris information in **fbar.bin**, which is dumped to stdout as a RINEX NAV file (and here redirected to the file **fbar2240.96n**). The **+obs fbar2240.96o** option instructs **teqc** that if any observation records are encountered in the target **fbar.bin**, they are to be decoded and written as a RINEX OBS file **fbar2240.96o**. Again, the option **-week 866** is needed to determine the epochs of the observation data, not the ephemeris data.

Again, the analogous command line for a Trimble *.dat file would be:

```
teqc -tr dn -week 866 +obs fbar2240.96o fbar.dat > fbar2240.96n
```

If you execute the above commands for the RS-232 file **fbar.bin** and do not have the environment variables **$teqc_OPT** or **$teqc_CONFIG** set (or if you do have them set but they do not contain any **-O.*** or **-N.*** header modification options), then you will find that most of the RINEX header fields in **fbar2240.96o** and **fbar2240.96n** are blank. Why? Like the GPS week in RS-232 observation records, there are no fields in the Trimble RS-232 data records to hold the type of information that would occupy these RINEX fields. About the only fields that are filled automatically by **teqc** are those for the initial **RINEX VERSION / TYPE** record (which are implied), the default **WAVELENGTH FACT L1/2** record (implied by the receiver type, in this case), and the **# / TYPES OF OBSERV** record. However, you can override these blank values by specifying your own **-O.*** and/or **-N.*** options using the command line, **$teqc_OPT**, **$teqc_CONFIG**, or other configuration files. This is using **teqc** simultaneously in edit and translate modes.

The above translation procedure can also be windowed. Currently, though, fast search algorithms have not been written for any binary format, so you must use an explicit windowing (windowing options (6), (7), or (8)) or specify the window delta time from the start (windowing option (2)).

The translation procedure can also be ***qc***-ed. Here let’s assume that you have a Trimble *.dat file called **fbar.dat**. For the normal type of qc operation, try something like:

```
teqc +qc -week 866 [-st 960811000000] +dh 24 -tr d \
     +obs fbar2240.96o +nav fbar2240.96n fbar.dat | more
```

where now, stdout will contain what you now expect from a *qc*-mode execution, but the RINEX OBS file is still being output to the file **fbar2240.96o** using the option flag **+obs**. The **-st** option is optional, indicated by the square brackets. You use the explicit windowing (in this example, windowing option (7) using both the **-st** option and the **+d*** option).

There is also another way to ***qc*** the target **fbar.dat**. For normal ***qc***-ing, the qc ASCII time plot can be thought in "landscape" orientation; for binary data you can also use a "portrait" orientation, which has been designed with real-time data processing in mind. (Perhaps these would be better described as "time window" and "time stream" orientations, rather than "landscape" and "portrait", respectively.) To use the portrait orientation, use the option flag **+p**:

```
teqc +p +qc -week 866 -tr d +obs fbar2240.96o +nav fbar2240.96n fbar.dat | more
```

Notice that here you do not need to specify the time window; **teqc** starts at the beginning of the observations and streams out a portrait version of the ASCII time plot until the target (file **fbar.dat**, in this case) is exhausted.

To review, there are a few things to remember when using binary data:

1. Currently, you must specify the record type of primary interest, e.g. using the **-tr** option for Trimble data, with a **d** argument for download (*.dat) format or **s** for RS-232 stream format. If not doing a *qc* mode, the RINEX file type that corresponds to this record type is dumped to stdout, e.g. if **-tr do** is used, RINEX OBS file information is dumped to stdout. (For most cases when doing *qc* mode, qc information is dumped to stdout.)
2. You should specify the GPS week during which the binary stream starts, or you accept your computer system version of the local time from which the GPS week is computed. For Trimble *.dat files, you might try leaving off the **-week** option, though occasionally the record containing the initial GPS week is corrupted, in which case **teqc** defaults to using the system time for the GPS week. Additionally, some Trimble MES files have been found that contain strange years like "19116"!.
3. If doing a *qc* mode, you must supply some information about the length of the time window of interest, using either the **+d*** flag, or one of the explicit window options (6) - (8). If doing the former, the time of the first data observation becomes the start time of the window. The partial argument format for the **-st** and/or **-e** options also work.
4. If doing a *qc* mode, stdout is used to dump a copy of the short report segment. In order to capture the RINEX file type that would have gone to stdout if not doing a *qc* mode, specify the RINEX file name by using a **+obs**, **+nav**, or **+met** option.

Known bugs and limitations:

- Currently, code for only the more recent Trimble record types have been written so far:

  Trimble DAT records: 3 5 6 8 9 10 11 12 15 16 17 19 20 21
  Trimble RS-232 records: 55h 57h

The following records have not been implemented yet:

  Trimble DAT records: 0 1 2 7
  Trimble RS-232 records: all others

Additionally, many of these record types contain multiple subrecord types. e.g. Record 16 has 18 different subrecord types. Not all of these subrecords have code written for them at this point. Also, some of these subrecords contain information which is pertinent to fields in the RINEX files, but it is not clear how the information will be stored from the Trimble documentation. Therefore, an example is needed before the code can be written. You may encounter a DAT or RS-232 file where **teqc** will terminate early with a cryptic message. Contact Lou Estey for help.

Special Translator Options
Section 16.

There are several translator options which are specific to a particular native binary format.

A. Trimble *.dat or RS-232 Data Formats

There is a set of options to remove half-wavelength phase data (squaring mode) from the translated RINEX OBS file. These are **-L1_2** or **-L2_2** to remove squared L1 or squared L2, respectively. Of the types of binary data that **teqc** currently handles, the only types where these flags may be of use is with the Trimble *.dat or RS-232 data stream formats.

When using the newest generation of receivers (e.g. SSE or SSi), a few epochs of squared L2 data for a particular SV may be reported. Normally, these epochs are translated with the appropriate bit-1 of the LLI flags added to the RINEX OBS file with the (squared) L2 data (see

teqc's handling of wavelength factors for more information). These can be entirely removed during translation by using the option **-L2_2**, i.e. no squared L2 data is passed to the RINEX OBS file.

When using a Trimble DAT file as a target file (and not stdin), **teqc** attempts to find a Trimble MES file with the same path and name prefix. The name matching uses:

```
     *dat    ->   looks for *mes
     *DAT    ->   looks for *MES
```

If a matching MES file is found, it is read to obtain the starting GPS week and certain meta-data (though there is no guarantee that this information is correct).

B. TurboBinary

TurboBinary data can include normal-rate data (record 0x68), 1-sec rate data (record 0x1a), up to 50 Hz-high-rate data (records 0xdb and 0xdc, plus using information in record 0x1a), and the so-called "31-second" format which is a mix of normal-rate data (record 0x68) and 1-sec LC data (record 0xde). The default translation is to do all these record types. However, you can tailor your translation with these options:

-TBhr
leave out high-rate data
-TB1s
leave out 1-sec data (this also deletes the high-rate data)
-TB68
leave out the normal-rate data
-TBLC
leave out the LC data records

The result of various option combinations is:

(no options = default) translate all records
-TB1s
translate only normal-rate data
-TB1s -TBhr
(same as -TB1s) translate only normal-rate data
-TB68
translate 1-sec and high-rate data or LC data (whatever is left)
-TBhr
translate 1-sec and normal-rate data
-TBhr -TB68
translate only 1-sec data
-TBLC
leave out the LC data records, leaving the normal-rate
-TB1s -TB68
translate no observation data (oops!)
-TBLC -TB68
translate no observation data (oops!)

For TurboBinary data collected with firmware dated before about 1 Dec 92 (version 2.5 or earlier), it is necessary to apply a correction to obtain valid pseudoranges. To activate this correction, use the option **+TB_ca_fix**.

C. Ashtech Z-12 RS-232 Data Format

The Ashtech Z-12 RS-232 data stream contains the results of a smoothing algorithm which can be applied to the pseudoranges. Normally this is not done. However, the smoothing correction can be applied by using the option **+smoothing**.

A bug in the some earlier firmware for the Z-12 resulted in the millisecond clock resets being applied an epoch too late. If you notice periodic millisecond slips (occurring just prior to the reported clock reset times), try using the option **+Ashtech_old_clk_reset** during translation. This should remove the firmware artifact from the data.

Wavelength Factors: What **teqc** Does With Them
Section 17.

Wavelength factors, i.e. specifying whether L1 or L2 is being recording in a full-wavelength or half-wavelength (squaring mode), can be done in various ways in RINEX. In short, 1) there is a required RINEX header record **WAVELENGTH FACT L1/2** specifying the default wavelength factors for L1 and L2 for all SVs, 2) there can be other **WAVELENGTH FACT L1/2** records specifying the a different set of wavelength factors for specific SVs, and 3) there can be the use of bit-1 in the LLI flag of the L1 or L2 observations to indicate the opposite state of wavelength factor from the last **WAVELENGTH FACT L1/2** record for a specific SV. The possible set of values for any **WAVELENGTH FACT L1/2** record is "1 0", "1 1", "1 2", "2 0", "2 1", and "2 2". Setting bit-1 of the LLI flag indicates a full-wavelength mode if the last **WAVELENGTH FACT L1/2** record for that frequency and that SV--somewhere earlier in file--was set at "2" (half-wavelength). Likewise, setting bit-1 of the LLI flag indicates a half-wavelength mode if the last **WAVELENGTH FACT L1/2** record for that frequency and that SV--somewhere earlier in the file--was set at "1" (full-wavelength).

The methodology used by **teqc** is a simple specific subset of all of the possibilities for RINEX, but still retains all the same information. On output (either when translating from native binary formats to RINEX or RINEX to RINEX), only the default **WAVELENGTH FACT L1/2** header record will appear and only the L1/L2 states of "1 1" or "1 0" are used. In other words, the default setting reported in the RINEX file header is always full wavelength for L1 and L2 (if present), even for squaring receivers. Specific half-wavelength observations are indicated by setting the appropriate bit-1 of the LLI flag on the L1 or L2 observations. Period.

During translation, you have the option of excluding all half-wavelength observations. To do this, include either **-L1_2** or **-L2_2** to exclude squared L1 or L2, respectively. This works either when translating native binary formats to RINEX or during any RINEX to RINEX operations. The default settings of **teqc** for wavelength factors are **+L1_2** and **+L2_2**, i.e. include all half-wavelength observations.

Basic Commands: A Review

Section 18.

The following examples assume that the shell environment variables **$teqc_OPT** and **$teqc_CONFIG** are unset or empty:

teqc
    forces all initialization and reports the current GPS week based on the system time

teqc +id
    identification of the **teqc** version you have, plus other information like your computer system time, sent to stderr

teqc -help or **teqc +help**
    complete option list is spewed to stderr

teqc +err my_help_file +help
    complete option list is spewed to file **my_help_file** instead of stderr; **+err** option redirects all stderr to specified file

teqc ++config
    dumps the current configuration to stdout

teqc +qc ++config
    dumps current configuration and default qc settings/values to stdout

teqc +v RINEX_file
    reads the RINEX file **RINEX_file** and verifies its format; nothing is sent to stdout, though a verification message is sent to stderr

teqc ++config RINEX_OBS_file
    dumps current configuration and OBS header settings/values of **RINEX_OBS_file** to stdout (can use RINEX NAV or MET file also)

teqc RINEX_file
    reads and spews **RINEX_file** (with possibly some slight formatting improvements) back out to stdout

teqc +dh 6 RINEX_OBS_file
    reads and spews header and first 6 hours of observations of **RINEX_OBS_file** back out to stdout

teqc RINEX_file_1 RINEX_file_2

reads and splices the two RINEX files **RINEX_file_1** and **RINEX_file_2** back out to stdout as a single RINEX file; target RINEX files should be of the same type and in time order

teqc -O.mo foobar RINEX_OBS_file

read **RINEX_OBS_file** and change monument name to "foobar"; edited RINEX OBS file is spewed to stdout

teqc +qc RINEX_OBS_file

qc of **RINEX_OBS_file**; automatically searches for name-matching RINEX NAV file; qc short report segment is spewed to stdout; full qc report and qc plot files written to file system

teqc ++sym

symbol hierarchy of symbol codes and associated meanings for qc ASCII time plot are spewed to stdout

teqc -tr do +nav RINEX_NAV_file trimble.dat

translate Trimble dat file **trimble.dat**; RINEX OBS file spewed to stdout; RINEX NAV file written to **RINEX_NAV_file**

teqc -warn {*rest of command*}

shut off warnings going to stderr; other functionality remains

---

Using **teqc** in Scripts: Substitution for Batch Mode
Section 19.

Because **teqc** is 100% non-interactive, it is very well suited to be used in scripts and to be run in background. In fact, this is precisely the reason it is designed to be 100% non-interactive. Let’s look at a simple script to translate Trimble *.dat files to RINEX and then qc the resulting RINEX files:

```
#!/bin/ksh
for file in $*
        do
                echo $file
                teqc -O.int 30 -tr 17 +nav ${file%dat}97n $file > ${file%dat}97o
                teqc +qc -set_mask 15 -plot ${file%dat}97o > ${file%dat}qc
        done
#end of script
```

Make sure the script is executable, i.e. execute (in UNIX) **chmod 755 script**, where **script**, say, is the name of your script file. To use it

script *.dat

Let’s take a closer look at what is going on when executing this script. The shell expands the *.dat on the command line to include all files in the working directory the end with **.dat**. Each of these file names are processed by the script. The first line of the script forces the script to be run in a Korn shell (**ksh**). The first real part of the script is to merely echo the name of each **.dat** file. Next, each **.dat** file is translated, with a sampling interval of 30 seconds being inserted into the header. The resulting RINEX NAV and OBS files will have the same prefix as the **.dat** file, but will end in **97n** and **97o**, respectively, rather than **dat**. Next, the RINEX files are *qc*-ed, setting the elevation mask to 15 degrees (overriding the default fo 10 degrees). Since we are not asking for COMPACT plot files (option **-plot**), only two qc files are created, again both having the same prefix as the **.dat** file. The more obvious one will end with **qc**, redirected from stdout. This is the short report segment (about one page in length). The other file created will end with **97S**, and is the full report, including--in this case--both the short report segment (just like what went to stdout) and the more detailed long report segment.

If you wish to suppress the short report segment in the report (the ***.97S** file), include a **-s** on the qc command line. If you don’t want any qc report file, include a **-report** on the command line. There is no way to suppress the short report segment going to stdout by command line option, but you can always use a > **/dev/null** on UNIX to eliminate the stdout.

Obviously, this script (as written) creates RINEX files named correctly according to the Berne naming conversion for *.dat files collected only in 1997, though the script will function for most *.dat files.

Notice that the GPS week has not been specified. For most *.dat files (at least from Trimble receivers with recent firmware), one of the first records usually contains the GPS week that the data starts. If this record is missing or is corrupted, then the resulting RINEX OBS file will probably have the wrong dates for the observations, leading to a poor qc report. For these rare *.dat cases, you must explicitly state the GPS week using the **-week** option.

Differences between **teqc**’s qc mode and original UNAVCO **QC**
Section 20.

Interface Differences:

- replacement of file **qc.inp** with **teqc** command line options, or equivalent format in environment variable **$teqc_CONFIG**, or in one or more configuration files which are accessed with the **-config filename** option; all options have a reasonable default value so the user need not be initially concerned with the details
- elimination of file **qc.fil** for batch modes; use a command line script instead
- elimination of auxiliary files like **qc.tim** or **qc.sym**
- roughly,
    - original **QC** stdout is like the **teqc +qc** *.*YY*S qc report file
    - original **QC** *.*YY*S file is like the **teqc +qc** stdout (qc short report segment)

Internal Differences (or Why the New QC Results Look Slightly Different From the Old QC Results):

- (nominal) one pass of files in **teqc** [original **QC** often required two complete passes of each RINEX OBS file]
- separation of NAVSTAR GPS, GLONASS, Transit, and other systems [original **QC** designed for NAVSTAR GPS only]
- well-defined symbol hierarchy for ASCII time plot [original **QC** had only a partially defined symbol hierarchy]
- cubic spline xyz fits of SV orbits for elevation, azimuth estimates [original **QC** used direct linear fits of elevation, azimuth which were only good for stationary antennas and were not smooth; algorithm occasionally resulted in large errors in azimuth]
- improved multipath algorithm
- improved detection and reporting of observation data gaps [original **QC** often did not detect and report long data gaps for individual SVs, and had no means of reporting short data gaps for all SVs]
- correct identification of SV data below elevation mask [original **QC** often falsely reported epochs below elevation mask w/ data, when, in fact, the receiver stopped tracking when the SV was well above the elevation mask]
- correct count of ionospheric slips and "observations per slip" [original **QC** would count the first observation w/ both L1 and L2 as an SV started to be tracked again after it had been observed and then set as an ionospheric slip]
- satellite elevation and azimuth accounts for WGS 84 ellipsoidal Earth model [original **QC** assumed spherical Earth model, resulting in elevation errors up to 1/5 degree for mid-latitudes]
- more accurate tally of expected number of observations for each SV [for some data sets, original **QC** sometimes reported more observations than expected, resulting in "%" values exceeding 100%]
- sign of clock resets shown on ASCII time plot [original **QC** did not shown the sign of the clock reset]
- correct reporting of indicators in ASCII time plot [original **QC** often did not show some clock resets, sometimes incorrectly showed a clock reset to occur when there was none, and other minor problems with indicators]
- does not produce any unneeded auxiliary files (like **AUXFIL** or **temp.orb**)

Interpreting **teqc**’s qc Mode Output
Section 21.

The quality check output from **teqc** is in two portions, the short report segment and the long report segment. The short report segment include an ASCII time plot and a summary report on various parameters. The long report segment gives a more detailed breakdown on some of these parameters either by SV or by elevation (if *qc* full).

Short Report Segment:

In the short report segment, one of the most compact pieces of information from a qc output is the ASCII time plot. In this plot, a visual summary of various types of quality indicators are displayed for each satellite as a function of time. The SV PRN number is displayed on both the left and right side of the plot. The width of the ASCII time plot is controlled by the **-w[idth]** option, and is normally set to 72, though it can vary from 1 to 255; with a width of 72 and 24-hours worth of observation data, each ASCII character "bin" represents exactly 20 minutes of time. Each character shown in each spot in the ASCII time plot is the most significant item of note that took place for all of the observation epochs represented by that bin, according to a well-defined symbol hierarchy.

A listing of the entire ASCII plot symbol table can be dumped to stdout by executing:

```
teqc ++sym
```

Likewise, a summary of the symbol hierarchy table can be included in the short report segment by including a **+sym** option with any *qc*-mode run. But in this tutorial, let’s take a closer look at the symbols and their hierarchy. The symbols used on each "SV" line are as follows, with the first symbol having the highest priority in the symbol hierarchy, decreasing through the list:

C
: a clock slip occurred; a clock slip is an MP1 and MP2 slip that occurred for all satellites being observed (tracked) and had a value that was the same integral number of milliseconds to a resolution specified by **-msec_tol** in milliseconds; detection is turned off with **-cl** option

m
: similar to a clock slip, but only some (i.e. not all) satellites being observed (tracked) had an MP1 or MP2 slip that was an integral number of milliseconds, or the integral number was different for the different satellites; note: if the millisecond slip tolerance is 1e-2 (see **-msec_tol**), then there is roughly a 2:100 chance that a random MP1 or MP2 multipath slip will be tagged as an **m**, rather than **M**, **1**, or **2** (see elsewhere in this table)

I

	ionospheric (phase) slip occurred; detection is turned off with **-ion** option
M	both MP1 and MP2 (code) slip occurred, but was not integral number of milliseconds; detection is turned off with **-mp** option
1	only MP1 (code) slip occurred, but was not integral number of milliseconds; detection is turned off with **-mp** option
2	only MP2 (code) slip occurred, but was not integral number of milliseconds; detection is turned off with **-mp** option
-	for qc full, satellite was above elevation mask, but no data was apparently recorded by the receiver; for *qc*-lite (no ephemeris information), the data gap must also be less than the maximum specified (see argument of **-gap_mx** in minutes)
+	(qc full only) satellite was below elevation mask and a complete set of phase and code data was collected
^	(qc full only) satellite was below elevation mask and a partial set of phase and code data was collected
.	phase and/or code data for SV is L1 and C/A only & A/S is off; if qc full, satellite was above elevation mask
:	phase and/or code data for SV is L1 and P1 only & A/S is off; if qc full, satellite was above elevation mask
~	phase and/or code data for SV is L1, C/A, L2, P2 & A/S is off; if qc full, satellite was above elevation mask
*	phase and/or code data for SV is L1, P1, L2, P2 & A/S is off; if qc full, satellite was above elevation mask
,	phase and/or code data for SV is L1 and C/A only & A/S is on; if qc full, satellite was above elevation mask
;	phase and/or code data for SV is L1 and P1 only & A/S is on; if qc full, satellite was above elevation mask
o	phase and/or code data for SV is L1, C/A, L2, P2 & A/S is on; if qc full, satellite was above elevation mask
y	phase and/or code data for SV is L1, P1, L2, P2 & A/S is on; if qc full, satellite was above elevation mask
L	Loss of Lock indicator was set by receiver for L1 and/or L2; detection is turned off with **-lli** option
_	(underscore) (qc full only) satellite between horizon and elevation mask with no data collected by receiver; indicator is turned off

with **-hor** option

‘ ’ (blank) qc lite: no satellite tracked; qc full: no satellite calculated to be above horizon (**+hor** option) or above mask (**+hor** option or both **-hor** and **+mask** options) (see also **-set_hor** and **-set_mask** options)

Let’s examine a simple example. Suppose that the horizon is set at 0 degrees and the elevation mask is set at 20 degrees. Let’s also suppose that the receiver starts tracking a particular satellite when it reaches 25 degrees of elevation and continues to track the satellite down to 5 degrees of elevation. Let’s also assume that no slips occurred during tracking and that the **+hor** option is set. For qc full (SV ephemeris available), the SV symbol track might then look something like one of the following:

```
A/S on:    _____--oooooooooooooooooo+++++__

A/S off:   _____--******************+++++__

           1    2 3                 4    5 6
```

At time (1), the SV rises above the horizon (0 degrees). At time (2), the SV rises above the elevation mask (20 degrees), but the receiver doesn’t start tracking until is rises to 25 degrees at time (3). Between times (3) and (4) all phase and code observables are collected by the receiver (L1, L2, C/A, and P2 for A/S on; L1, L2, P1, and P2 for A/S off). Data continued to be collected as the SV dropped below the elevation mask at time (4) until the receiver stopped tracking at an elevation of 5 degrees at time (5). The SV finally sets below the horizon at time (6). For qc lite, the above SV symbol track would appear as:

```
A/S on:           ooooooooooooooooooooooo

A/S off:          ***********************

           1    2 3                 4    5 6
```

as **teqc** as no information about the elevation of the satellite.

If the **-hor** option is set, the above qc full SV symbol tracks would then appear as:

```
A/S on:         --oooooooooooooooooo+++++

A/S off:        --******************+++++

           1    2 3                 4    5 6
```

so that you can determine everything you could with **+hor** set, except for the rise and set times of the SV at times (1) and (2), respectively. For qc lite, the SV symbol tracks would remain the same as before, since the options **-hor** and **+hor** are meaningless.

Any additional symbols that occur in an SV symbol track not in the above examples fall into a "not so good" category, though seeing a lot of **-** symbols in a qc full output is also "not good". Let’s take a look in more detail at what these other indicators might be.

The first "not so good" category could generally be considered "missing data". As mentioned above, the worst missing data indicator (qc full only) is the **-**, which means that the SV was calculated using the supplied ephemeris to be above the elevation mask, but no observation data was present for this SV. In other words, all data is missing.

Following the **-** "all data missing" indicator are the partial missing data indicators. For example, if A/S is normally on, seeing **;** or **,** indicates that there was at least one observation epoch in that bin where L2 observables (i.e. L2 and P2) were missing. You are unlikely to see a **y**, as this would require a Y-code receiver (capable of tracking P1 while A/S is on). (An exception to this is when *qc*-ing data from an Ashtech receiver like the Z-12. The C/A and P1 pseudorange observables are reported for all SVs, regardless of whether A/S is on or off.) If A/S is normally off, seeing **:** or **.** indicates that there was at least one observation epoch in that bin where the L2 observables were missing. A **~** indicates that, for some reason, the receiver could not track P1, even though A/S was off, so the receiver instead recorded the C/A observable.

If you are using a P-code (not a Y-code) receiver that reports C/A and P1 pseudoranges for each SV at each epoch (like the Ashtech Z-12), you may want to use the **-Y** option, which informs **teqc** that this is data not from a Y-code receiver, and to treat the qc analysis as though from a P-code receiver.

Special Treatment of Data from Codeless Receivers

You may have a data set that was collected with a "codeless" or "squaring" receiver. For these receivers, the pseudoranges P1 and P2 are never recorded, and the default qc report will show that all observations were incomplete, as none of them can have a P2 observation (though C/A is acceptable in place of the missing P1). Also, the SV symbol tracks for the examples above would then appear as:

```
A/S on:    _____--,,,,,,,,,,,,,,,,,,+++++__

A/S off:   _____--..................+++++__

           1    2 3                 4    5 6
```

which (correctly) indicates a lack of L2 and P2, even though the observable P2 is not possible (and thus never present) and L2 may be always present.

You can inform **teqc** that the data is to be interpreted as though it were collected by a codeless receiver by including a **-P** option (and, of course, the default option in **+P**). In this case, the absence of P-codes is ignored for statistics and the data indicators change collapse to just two possibilities (from the original eight possibilities):

.
phase and/or code data for SV is L1, C/A; if qc full, satellite was above elevation mask

o
phase and/or code data for SV is L1, C/A, and L2; if qc full, satellite was above elevation mask

Notice that the A/S state (on or off) is ignored, as this is irrelevant for processing squared data. Then, when using the **-P** option, the above SV symbol tracks will appear as:

```
A/S on:    _____--oooooooooooooooooo+++++__

A/S off:   _____--oooooooooooooooooo+++++__

           1    2 3                 4    5 6
```

i.e., there is no difference whether A/S is on or off. You will probably see an occasional **.** data indicator:

```
A/S any:   _____--ooo.ooooooo.oooooo+++++__

           1    2 3                 4    5 6
```

indicating one or more observation epochs where the observable L2 is missing.

A listing of the entire ASCII plot symbol table modified for codeless receivers can be dumped to stdout by executing:

teqc -P ++sym

Other Indicators

The next set of "not so good" indicators are for slips in the observables. A common type of slip is an ionospheric (phase) slip indicated by a **I** symbol. These often occur when the SV is at low elevation, both while rising and setting, so the example SV symbol tracks from above might really appear as:

```
     A/S on:    _____--Ioooooooooooooooooo++I+I__

     A/S off:   _____--I*****************++I+I__

                1    2 3                4    5 6
```

In this example, ionospheric slips are detected shortly after the receiver starts tracking the SV and as the SV is close to setting.

The other common type of slip is for one or both of the multipath (code) observables, MP1 and MP2. Again, these frequently occur at low elevations. There are three symbols: **M** for slip on both MP1 and MP2, **1** for slip on MP1 only, and **2** for slip on MP2 only. Notice that the multipath slip indicators take a lower priority in the symbol hierarchy, so if an ionospheric slip and multipath slip occur at different observations epochs within the ASCII bin, only the **I** symbol will be seen (assuming that the option configuration is **+ion** and **+mp**). If you use **-ion** to suppress ionospheric slip detection, the above SV symbol tracks might now appear as:

```
     A/S on:    _____--Mooooooooooooooooo++2+M__

     A/S off:   _____--M*****************++2+M__

                1    2 3                4    5 6
```

Interestingly, the occurrence of ionospheric or multipath slips in the recorded observations is not only a function of the antenna environment (meaning all the way back to the transmitting SV), but is also a function of the specific receiver. Due to internal slip detection and phase observables resets, data from some receivers show virtually no ionospheric slips and nearly all multipath slips. Other receivers do not reset the phase observables, and show a larger number of ionospheric slips than multipath slips.

The last type of slip is the "n-millisecond clock slip", where the value of n is usually 1, denoted by the symbol **C** in the SV symbol track. This slip is reported if all SVs being tracked have slips in MP1 and MP2 equivalent to the same number of integral milliseconds, to a tolerance specified by the **-msec_tol** option. If you set the tolerance to 1e-17 (milliseconds), you probably will never see any of these slips. However, the default tolerance is 1e-2 (milliseconds), and with this value the qc mode of **teqc** seems pretty capable of detecting these slips if they are present.

What does it mean if you see the **C** symbol in an SV symbol track? There are several causes, some more harmless than others. If the **C** symbol is preceded by an observation gap (no data collected for any SVs), there may be one or more millisecond clock resets missing from the observation epoch time tags. Also, if you use **teqc** to splice two RINEX OBS files together and clock resets occur in the first file, a **C** will occur at the first epoch of the second file (since the **teqc** splice does not modify the observation times in the second file to account for the accumulative clock resets in the first file). In other cases, however, if the observation epochs are fairly continuous, and the **C** indicator is appearing two or more times in 24-hours of data, there is a strong possibility that the receiver was not healthy.

This latter possibility (that the receiver was not healthy) prompted the inclusion of another slip indicator, the "n-millisecond multipath slip". It was observed that some receivers get so unhealthy that, even though n-millisecond clock slips should be occurring (i.e., given the specific receiver, no millisecond clock resets are present, even though they were expected) none were being found because the multipath slips for different SVs had different millisecond equivalents. In short, the value for n was not a constant for all SVs being tracked. In this case, the **m** symbol is used. There is some probability (roughly about 2 : inverse of millisecond tolerance) that a random multipath slip will be recorded as an **m** instead of as a **M** or **1** or **2**; so, treat the occasional **m** as you would any multipath slip. However, if you start to see lots of **m** symbols, especially if you have seen **C** symbols being reported in the data from the same receiver, suspect that the receiver is ailing.

The next "not so good" category is presence of data gaps. There are really two types of gaps. One is a complete observation gap for all SVs. This can be caused perhaps by the receiver being turned off and later turned back on, by a loss of all data for a period of time either internal to the receiver itself or due to a communication breakdown with the receiver. Currently, a complete observation gap is not indicated in the SV symbol tracks, except (on occasion) if they are present in a qc lite run.

The other type of gap is the SV data gap, where the receiver stops tracking an SV for a period of time even though it is well above the horizon or elevation mask, perhaps due to an obstruction. The exact definition of an SV data gap depends on whether **teqc** is running in a qc full or qc lite mode. For qc full, an SV data gap occurs if there are one or more missing observation epochs while the SV is above the elevation mask. For qc lite, an SV data gap occurs if tracking stops for more than the specified minimum time (**-gap_mn**, again) and then tracking resumes before a specified maximum time (see **-gap_mx** option). The symbol used in the SV symbol track is now **-**, so an SV data gap might look like:

```
    A/S on:     _____--Iooooooooooo--ooooo++I+I__

    A/S off:    _____--I**********--*****++I+I__

                1    2 3          ab     4    5 6
```

which occurred at the interval (ab). The meaning is really the same as the interval (23), i.e. the SV was above the elevation mask, but no data was received.

The only other indicator for SV data, low in the symbol hierarchy, is the "Loss of Lock" indicator, **L**, which is used when the receiver issues a loss of lock for either the L1 or the L2 observable. A large number of **L** symbols may indicate an unhealthy receiver or antenna. This should rarely, if ever, be seen in the ASCII time plot.

Following the SV symbol tracks are one or four more, depending on whether you are using qc lite or qc full, respectively. For qc lite, there is a symbol line labeled "Obs". This records the maximum number of SVs that were tracked by the receiver for each bin using a hexidecimal representation. For example, if there is a **7** on this line, then 7 SVs were tracked for at least one observation epoch represented by that time bin; if there is a **b** on this line, then 11 SVs were tracked for a least one observation epoch represented by that time bin; and so on. If no SVs were tracked, a (blank), rather than **0**, is shown. If one or more s are present on the "Obs" line in a qc lite run, this is your best indicator that a complete observation gap has occurred.

For qc full runs, the "Obs" line is replaced by four lines, labeled "-dn", "+dn", "+*XX*", and "Pos". The "+*XX*" line is the one most like the qc lite "Obs" line; the *XX* is replaced by the elevation mask in degrees (rounded to the nearest degree) and indicates the maximum expected number of SVs that are above the elevation mask, according to the supplied ephemerides, again using a hexidecimal notation. The difference between the qc lite "Obs" line and the qc full "+*XX*" line is the difference between reality and theory: "Obs" shows what was seen, where "+*XX*" shows what could have been seen (above the elevation mask).

The discrepancy between reality and theory is recorded in the "-dn" and "+dn" lines, which are the SV tracking discrepancy counts, and record the two bounds of the discrepancy. They can be thought of as the "good new/bad news" to the number of SVs not tracked. The line "-dn" records the minimum discrepancy of all observation epochs for that time bin while the line "+dn" records the maximum discrepancy of all observation epochs for that time bin. The discrepancy count is also shown in hexidecimal notation, with (blank) for 0.

For the discrepancy lines to work correctly, the option **-max_SVs** must be set correctly. This states the maximum number of SVs that are capable of being tracked by the receiver, and currently has a default value of 12. If a complete observation gap occurs with qc full, a group of **c** characters will be shown (c is hex for 12), at least on the "+dn" line, and if the gap is large enough, on the "-dn" line as well.

The "Pos" line records the success or non-success of calculated code positions for the antenna at the different epochs. Generally, you should see an **o** recorded for each bin in which a position calculation was successful.

The last symbol line in the ASCII time plot is labeled "Clk". In the original UNAVCO **QC**, this line (labeled "CLK") represented all millisecond receiver clock resets present in the observation epochs with a **C** symbol. This has been replaced with either a **+** or **-** symbol, meaning either a positive or negative millisecond receiver clock reset was detected, respectively. Another symbol which may be placed on this line is ^, which is lower in the clock symbol hierarchy that either **+** or **-**, and indicates at least one missed observation epoch in that time bin, though a correct value of the observation sampling interval must be set (see **-O.int** option) for this to work correctly. This symbol was added to help reveal two things: 1) the existence of "micro-gaps", i.e. missing data periods less than that set by the **-gap_mn** option, and 2) to identify short gaps during which a millisecond clock reset may have occurred. For example, if a portion of the "Clk" symbol track is:

```
    Clk:   +  +  +  ^^ +  +  + ^+  +  +  +  +
                   12        3
```

then you can suspect a missing (positive) millisecond receiver clock reset at time (1) due to the regularity of the rest of the identified resets and the missing epoch indicator ^, and other missing observation epochs at times (2) and (3). Other micro-gaps might exist in the data, but their presence would be hidden by the **+** symbols.

Incidentally, another "micro-gap" indicator exists for qc full runs on the "+dn" line. Since this is maximum discrepancy between the number of SVs that could have been observed and what were actually observed. However, for missing observation epochs (no SVs observed), rather than placing a count of just "SVs that could have been observed" based on the ephemerides, **teqc** places a count of the maximum allowed by the receiver. So, for a receiver capable of tracking 12 SVs (see **-max_SVs** option), you will also see a **c** (hex for 12) on this line when missing a observation epoch.

Following the "Clk" line of the ASCII time plot is a scale bar with tick marks. The separation between tick marks is indicated a few lines lower at "Time line window length". The tick marks should occur at even values of the tick interval. For example, if the time window starts at 01:39:30.000 (1h 39m 30s) and the tick interval is 3 hours, the tick marks will be placed at 03:00:00, 06:00:00, 09:00:00, and so on, to the best resolution possible on the ASCII plot. The tic interval is self-scaling, from 0.1 seconds to 7 days, depending on the length of the time window.

At the end of the ASCII time plot, the beginning and end time and date follow at the ends of scale bar. Seconds are only printed out if they are non-zero.

Following the ASCII time plot in the short report segment is a report summary. First is a listing of the name of target files, and if *qc*-full, the RINEX NAV files used.

The bounds of the time window is then shown. If the times of the first and/or last observation epochs do not match the bounds of the time window, these epoch times are shown as well.

If the configuration environment variable or any configuration files were used, these are listed next.

If the observation interval is non-zero, this is given.

The total number of satellites (SVs) with any type of observations is then given. This is followed by a list of missing SVs, up to the maximum set by default (probably 32), or using the **-PRNs** option, for each satellite system that had any members. Finally, if doing *qc*-full, a list of SVs that did not have ephemeris information is given.

The number of SVs which can be simultaneously tracked by the receiver is then given. This currently has a default value of 12, or can be changed using the **-max_SVs** option.

If the observation interval is non-zero, the total number of possible observation epochs in the time window is given. This is followed the number of epochs that actually had "complete observations" from at least one SV.

The definition of a "complete observation" is important, so it will be defined in detail here. In order for an observation from a GPS SV to be "complete", it must have :

1. P1 or C/A code data
2. P2 code data
3. L1 and L2 code data
4. S/N for both L1 and L2 be at or above specified minima
5. if *qc*-full, an SV elevation at or above the elevation mask

Then the numbers of possible, complete, and deleted observations are given. If doing *qc*-full, the both the number of possible observations above the horizon and above the elevation mask are given first. Next, the number of complete observations is given; if *qc*-full, this is restricted to those observations above the elevation mask. Next, the number of deleted observations is given; if *qc*-full, this is also restricted to those observations above the elevation mask.

If the multipath option was set (which it is by default), the average multipath RMS is given. If a moving average window was used (which is used by default), information about the length of this window is given. If *qc*-full, the multipath RMS is only for observations above the elevation mask.

The number of detected millisecond receiver clock resets is then given. This is followed by the total drift of the receiver clock, an estimate of the average receiver clock drift, and, if the number of clock resets is non-zero, the average time between resets in minutes.

The length of time required before an SV data gap is reported is given next. If *qc*-lite, a maximum time is also given.

If the detection of n-millisecond clock slips is on (**+cl** option), the number of epochs with n-msec clock slips is reported. This occurs when all SVs with multipath observables must have multipath slips of the same size to within a specified tolerance (fraction of millisecond).

This is followed by the number of other n-millisecond multipath slips which do not qualify as n-millisecond clock slips. Given a non-zero tolerance, there is a certain probability that a few multipath slips fall within the tolerance. Therefore, a second value is given in parentheses and this is the total number of multipath slips for the time window (no elevation mask cutoff). If the tolerance is set to 1e-2 millisecond, ratios on the order of 2:100 are expected due to chance. Significantly higher ratios are an indication of a sick receiver.

Next if doing the derivative of the L2-ionospheric observable (**+iod**) or multipath (**+mp**), counts of the number of IOD and/or multipath slips is given. If *qc*-full, this is further broken down according to elevation mask. In order to qualify as a count here, both MP1 and MP2 must slip (though not necessarily by the same amount) at the same epoch for a particular SV.

Finally, a "SUM" line in printed, showing the start and end times of the window, the length of the time window in hours, the observation interval in seconds, the number of possible observations (if *qc*-full), the number of complete observations, the ratio of complete to possible observations as a percent (if *qc*-full), the multipath RMS values for MP1 and MP2 (limited by the elevation mask if *qc*-full), and lastly the "observations per slip".

The "observations per slip" needs a bit of explanation. First, "observations" means "complete observations" as defined above, including the elevation mask if *qc*-full. Second, "slip" means "either an IOD slip and/or both MP1 and MP2 slips occurred during the epoch having a complete observation for this SV".

Some additional information for each SV can be included in the short report segment by using the **+ssv** option. Similar to the main SUM line, shown for each SV with observations are: the expected number of observations, the number of complete observations, the number of deleted observations, ratio of complete to possible observations, multipath RMS values for MP1 and MP2, and the observations per slip.

Long Report Segment:

The long report segment contains a further breakdown of information by SV and by elevation (if *qc*-full). In the long report segment, individual SVs are often referenced. The leading character indicated the satellite system:

G: NAVSTAR GPS system
R: GLONASS system
T: NNSS Transit syste

The first portion is a list of some of the processing parameters, followed by a time stamp of the first and last observation epochs within the time window, and the observation interval.

Next is a breakdown of observations per SV. If doing *qc*-full and the SV had ephemeris data, the values in the first four columns have meaning:

#+hor:

number of observations above the horizon for this SV

<ele>:

mean elevation of SV above the horizon for epochs with observations

#+mask:

number of observations above the elevation mask for this SV

<ele>:

mean elevation of SV above the elevation mask for epochs with observations

Next are the number of reported and complete observations. If doing *qc*-full and the SV had ephemeris data, the values are for epochs only above the elevation mask; otherwise, the values are for all epochs:

#reprt:

number of observations with any data reported for this SV

#compl:

number of "complete" observations reported for this SV (see definition in "Short Report Segment")

Next are the number of L1, L2, P1, P2, and C/A observations. Again, if doing *qc*-full and the SV had ephemeris data, the values are for epochs only above the elevation mask; otherwise, the values are for all epochs:

L1:

number of L1 observations for this SV

L2:

number of L2 observations for this SV

P1:

number of P1 observations for this SV

P2:

number of P2 observations for this SV

CA:

number of C/A observations for this SV

If doing *qc*-full, any SV computed to be above the elevation mask but not having any data reported is listed next.

If doing *qc*-full, any SV not having ephemeris data but having observation data of any kind is identified with a "*".

Next, a summary tally is given. If doing *qc*-full and a site position was found, the total number of observation below the elevation mask is given (i.e., number of observations excluded because of low elevation). Next, reasons for incomplete observations (above the elevation mask if a site position was found) are summarized: missing L1, L2, P1 or C/A, or P2, or poor S/N for L1 or L2.

Following this is the number of observations reported with any code or phase data. (Note: If an SV has only, say, Doppler data, it will not be reported here.) This is followed by the number of observation deleted for any reason: below elevation mask (if *qc*-full), missing code or phase data, and/or poor S/N. Finally, the number of complete observation is given.

Next, repeat of the receiver clock offset and drift statistics is given.

"Strange" Behavior
Section 22.

- If doing qc full mode (i.e., NAV file(s) supplied either implicitly or explicitly or using binary target files) the **qc full**>>>>>>>**...** indicator may, on some file data sets, pause part way through and then appear to keep going. Don't panic. Everything is operating normally. Here's what is happening:

  The qc full mode really starts off in a qc lite mode. When using target files (as opposed to stdin), **teqc** has the luxury of being able to go to any arbitrary location in a file. The first primary goal of the a qc full run is to find the pseudorange point position of the antenna. A certain minimal amount of information is required before this is possible. There must be a certain number of SVs reporting pseudorange data for a given epoch and **teqc** must have ephemeris information for those SVs. Occasionally, this does not happen early in the file. When it does happen, **teqc** starts re-reading and re-processing the target file now knowing the antenna position. If plot files have been requested, this is when they are written. The pause you are seeing is the time it takes **teqc** to go back and re-do all these items and get back up to the epoch when the point position was determined.
- If doing qc full mode (i.e., NAV file(s) supplied either implicitly or explicitly or using binary target files) and the plot option is turned on, but no position was found (for whatever reason), no plot files are created. This is a consequence of the logic used for the qc full mode (see above item).
- Direct qc of binary files produces a slightly different result than qc of RINEX files. This is due to the direct qc of binaries being designed for direct data streams from receivers, and thus lacking the capability of treating the data stream as a file. Also for direct qc, the plot file information regarding the elevation and azimuth of each SV will begin at the first epoch where both the antenna position and an SV ephemeris are both known, whereas for qc of RINEX files, the elevation and azimuth information will be computed for the entire window of interest. This behavior can usually be correctly by pre-loading a RINEX NAV file, say, from the same site and the previous day, using the **-nav** option.

- If doing any qc mode, the user intends to input NAV file(s), but uses **+nav filename** instead of **-nav filename**, the original file filename may be re-opened and destroyed. One safeguard has been implemented to help prevent this when the qc command is ordered:

  teqc [options] +qc [options] +nav filename [rest_of_command]

  in other words, turning on the qc option before specifying the RINEX NAV filename, and the command does not involve a translation from binary. In this case, the program will not allow the filename to be re-opened in a "write" mode, which if it took place would destroy the original file.
- When using the Borland **DOS** shell version of **teqc**, ASCII lines written to files by **teqc** terminate with a newline ('\n' = CTRL J = 0x0a). ASCII lines spewed to stdout and then redirected to a file will be terminated with a carriage return ('\r' = CTRL M = 0x0d) and then the newline. This added carriage return is apparently being done by the **DOS** shell.

  The WatCom **DOS** shell version of **teqc** results in the extra carriage returns being added in both cases, both as files written by **teqc** and stdout redirected to a file.

  As usual, if you ftp the **DOS**-created files to UNIX in ASCII mode, the added carriage returns are removed; if you ftp the same files to UNIX in binary mode, the added carriage returns are left in the files.