The **backstop** book

backstop release 6.2
April 2018

Table of Contents

Introduction

The `backstop` tool provides critical command load verification functionality in its facilitation of the following tasks:

- comparison of candidate command load files with OFLS timeline report;
- automated inspection of the load for violation of a variety of constraints, based on the sequence, timing, format, and content of commands;
- production of a checklist combining aspects of the automated inspection of the load and conditions that are checked manually;
- production of state reports for use during real-time support of the spacecraft.

The tool consists of a set of scripts, written in `perl` and `tcsh`, which when run in sequence, provide for the above-described functionality. In addition, the tool draws upon several ancillary data files, which are listed below and are revision controlled along with `backstop`. All are now found in the `aux` directory:

- `disabled_list.txt` – list of disabled command mnemonics
- `FOT_CHECKLIST_TEMPLATE.txt` – template for generation of the FOT checklist
- `SPM_patterns.txt` – SPM command hex patterns
- `MSIDCMDF.txt` – details of the fields of each single-part command (by mnemonic)

Purpose & Scope

The backstop book is intended to provide an overview of the functionality provided by the scripts and libraries comprising the backstop tool. Each script is discussed in varying levels of detail, as are selected elements of the functionality provided by backstop shared libraries.

This document can be used to gain a basic understanding of the tool, its components, and how backstop works. However, the document should not be held as the primary or definitive source for the finer points of the tool; the code itself is the only primary source in this regard. Inconsistencies between this document and the code should be investigated and corrected. In particular, if this document reflects incorrect or unintended behavior, the code should be investigated in order to determine whether the software is indeed deficient, or whether the error lays only within this document. While efforts have been made to ensure correctness of this document, the most reliable source for backstop is the code itself.

Creating New Tests

The following is the scheme employed for creating new tests.

- Script name: backstop_<name>.pl
- Output file name: TEST_<name>.txt

It is crucial that the script and output file names be coordinated. This coordination is assumed by the implementation of backstop.pl. The new test (by <name>) can then be added to the list of tests executed in backstop.pl (which is discussed below). Other considerations for implementation of new tests can be understood by looking at backstop.pl directly.

Selected Library Functions and General Discussion

.cld File Sorting

Prior to backstop 3.0, this behavior was implemented in a separate script, backstop_yearrollover.pl. For backstop 3.0, the behavior was folded into the only script which called backstop_yearrollover.pl: backstop_validate.pl.

The CLD file sorting capability provides computation of the correct time ordering of command load files (*.cld) in a backstop directory. The computation is strictly based on file naming conventions. The original intent was to provide correct ordering in the event of a year rollover, where the file, for example, for day 001 should be taken after the file for day 365, despite the fact that 001 is (numerically) less than 365.

Command load files are named as follows:

CL<day>_<hr><vv>[_<num>].cld

Where

- <day> - day number at which the command load file starts
- <hr> - the hour within that day at which the file starts
- <vv> - is command load revision number (and will be the same for the set of command load files for which filenames are being sorted)
- <num> - optional suffix appended to the file name if there are two command load files for the same day starting at the same hour. The convention used for <num> is as follows: the first such file (sequentially) will not have the suffix, while the 2nd through nth will have <num> equal to 1 through (n-1).

If a file is present with day number **less than 100** AND a file is present with day number **greater or equal to 300**, we assume that the year rolled over during this schedule (admittedly a generous range). In this case, for the purposes of comparison, all day numbers less than 100 are incremented by 366 in order to provide for correct ordering by day.

The sort algorithm is as follows, given two file names:

1. Compare day numbers: if unequal, lower day number compares first; if equal, continue
2. [Equal day numbers] compare hours: if unequal, lower hour compares first; if equal, continue
3. [Equal day numbers and hours] compare suffixes. If one file name has a suffix number and another does not, the file name without compares first; if both have a suffix, lower suffix number compares first.

A comparison result must be reached in Step 3 because, in order to assert equality at that point, two distinct files would need to have the same name (save the CLD revision number), which is not possible. This is of concern because the script is capable of handling exactly one occurrence of a set {day, hr,

suffix number}, a tuple which is assumed to be unique; an error message will be added to the script to catch the case where a non-unique condition is encountered.

.tlr File Processing

The TLR file is parsed in a manner much the same as in the backstop_validate script (**and, in fact, the parsing code will be factored into a shared library**). The following is a quick summary:

- Skip lines **1-45** (file header and first page header)
- Skip any line containing `COMMAND TIMELINE REPORT` and the **6** lines following it
- Skip any line which begins with an asterisk
- Skip any line containing `COMMENT`
- Skip any line starting with any of the following:
    - a space followed by '-' (dash)
    - a space followed by '|' (pipe)
    - '|' (pipe)
    - a space followed by '_' (underscore)

Any line not skipped based on the above is parsed into an array indexed by the value of Number (per field) in the below table. The line is parsed sequentially, field-by-field, based on the field lengths listed in the table, until reaching the end of the line. Thus, it is possible that one or more fields may be undefined. Fields not defined on the line are set to the empty field. The resultant record is an array with one entry per field.

The following is the table of the eleven fields comprising a TLR file record:

Number	Name	TLR field (col hdr)	Length (char)
0	TIME	GMT (extended)	21
1	ORBPT	Orbit/DSN Event	8
2	MSID	Command Mnemonic	14
3	CRIT	Is Critical Cmd or Not	1
4	DESC	Command Description	36
5	HEX	Command Data (Hex)	8
6	REF	Trace ID	14
7	VCDU	OBC Clock	8
8	OBC	OBC ID	3
9	SCS	SCS	3
10	STEP	Step	4

One exception to this table is the case of an orbit point record; if the `ORBPT` field is non-empty (e.g. whitespace only), the remainder of the line beyond that field is taken as the command description (with all other fields beyond `ORBPT` remaining undefined). This is based on the irregular format of orbit point records.

Fields are individually stripped of leading and trailing whitespace after the line is split. Additionally, the HEX field, taken as eight characters, is actually three hex digits followed by a space followed by another four hex digits. This space is removed in processing, leaving only the seven digits.

Maneuver Tracking

The following document is a discussion of the current maneuver tracking scheme for the backstop tool. The implementation was based on an earlier discussion, which (likely) took place during the initial specification of backstop_rptstate (reports by state and attitude).

Maneuver tracking functionality is provided in a shared library available to all backstop scripts, and depends on ManeuverCompLib, which provides functions for computation of the duration of a manuever. Clients of this library initialize a single ManeuverTracker object for use throughout an entire script, as its implementation provides for easy re-use.

The ManeuverTracker object has two attributes, based on the current maneuver:

- Start time
- End time

Upon construction of the object, both parameters are set to 0.

In using the object, clients "set" a manuever into the object, by providing two quaternions and the start time of the maneuver.

The ManeuverTracker attributes are populated as follows based on the above-listed parameters:

- Start time: provided
- End time: calculated based on start time and the quaternions provided

Tracking: Operationally, having retained the above attributes (in the case that a maneuver has been set into the object), a script continuously tracks the state of a maneuver relative to its iteration of a backstop file by checking the maneuver end time relative to the times of encountered commands. In providing the timestamp of a command (presumably encountered during iteration of the backstop file) to the ManeuverTracker object, clients receive the following status:

- NOTIN – Maneuver has not been set into the object
- INCOMPLETE – Maneuver is incomplete as of the timestamp provided (complete time < timestamp provided)
- COMPLETE – Maneuver is complete as of the timestamp provided (complete time >= timestamp provided)
- ERROR – ManeuverTracker object is in an unexpected state or timestamp provided is less than the start time of the maneuver

This library additionally provides helper functions for clients to obtain the maneuver start time and end time, as well as to reset the ManeuverTracker object for re-use. (The ManeuverTracker **must**

be reset by the client prior to setting another maneuver into it, and this is typically done when the maneuver is detected as having been completed.)

Perigee Library

The following document is a discussion of the Perigee Library for the `backstop` tool. The Perigee Library provides a setoff functions that construct an array of perigees using the history file EPHEMERS.txt, entries from the backstop file and estimates projecting forward in time beyond the backstop file entries. Each element of the array is a hash containing the time of perigee (Key: CMD_TIME) and the source of the data, History file, Backstop file, or Estimate (Key: SOURCE).

The source is stored as an integer with constants defined for the values:

History file: SRC_HISTORY (=-1)
Backstop file: SRC_BACKSTOP (=0)
Estimate: SRC_ESTIMATE (=1)

The time is stored in the format YYYYDDD.HHMMSSXXX (where XXX is milliseconds).

The Library includes four functions

`initializePerigeeTimes`

This function has one input argument, First Backstop Time one output, a pointer to the array of perigees, and one return value, result. The function initializes the array of perigees, using the last two perigee times in the history file that precede the First Backstop Time. The SOURCE component of these perigees will be SRC_HISTORY. The return value will be PASS, if the data can be accessed from the history file and FAIL, if not.

`addBackstopPerigeeTime`

This function has one input argument, Backstop Time and one input/output, a pointer to the array of perigees. The function adds one perigee time (Backstop Time) to the array of perigees. The SOURCE component of these perigees will be SRC_BACKSTOP.

`addEstimatedPerigeeTimes`

This function has one input argument, Last Backstop Time and one input/output, a pointer to the array of perigees. The function adds estimated perigee times to the array of perigees. The estimates will be computed using the estimated period (i.e. the time between the times of the last two perigees in the perigee array). The SOURCE component of these perigees will be SRC_ESTIMATE. The number of estimates added will be such that the last estimated time is greater than the Last Backstop Time.

`getPerigeeData`

This function has two input arguments, Time and SOURCE and returns one value. The value returned is a hash element of the hash array containing the two input arguments.

.backstop file records

Record Type	Event Type	Event Data string fields
SIM Packet	SIMTRANS SIMFOCUS	POS (calculated value) **or** SIMPKT (hex)
ACIS Packet	ACISPKT	TLMSID (MSID from TLR) CMDS (# commands) WORDS (# 4-digit-hex packets) PACKET(40) (accumulated hex)
Orbit Point Record	ORBPOINT	TYPE (ORBPT from TLR)
Regular (Single-Part)	COMMAND_SW COMMAND_HW	TLMSID (MSID from TLR) HEX (from TLR) MSID – (mnemonic from single-part cmd database) (if mult fields) <field> = <value> (if unknown) ??????
Multipart Commands		**TLMSID** **CMDS**
AOUPTARQ	MP_TARGQUAT	Q1, Q2, Q3, Q4
AOSTRCAT	MP_STARCAT	16 x (IMNUMX, YANGX, ZANGX, MAXMAGX, MINMAGX, DIMDTSX, RESTRKX, IMGSZX, TYPEX)
AODESMOS	MP_SYSMOM	TOTMOM1, TOTMOM2, TOTMOM3
COAOSQID	MP_OBSID	ID
EOECLETO EOECLSTO EOECLST2	MP_ECL_TIME	TIMECNT TIMESEC
AODITPAR	MP_DITHER	ANGP, ANGY, COEFP, COEFY, RATEP, RATEY
AOSETINT	MP_ACAINT	CMDFIELD INT TIME
AVAMPOOF	COMMAND_HW	MSID
AVBMPOOF	COMMAND_HW	MSID
AOSAPARS	COMMAND_HW	AOSSA1AS, AOSSA2AS, AOSSA1RR, AOSSA2RS

Record Format/Common Fields: TIME (from TLR record)**,** VCDU (tracking value when the command was encountered)**,** MC**,** Event Type (above), Event Data string (above), SCS (from TLR record) STEP (from TLR record).

backstop.pl (infrastructure)

Main script for backstop tool execution; intended to be run from the current week's schedule directory. Output appears on the screen and in log file(s) (the latter not including user responses to prompts)

The advent of split loads lends itself to a workflow dependent on whether the loads indeed are split. Such a workflow is designed to maintain backward compatibility. If the behavior of a particular script is dependent on whether loads are split, such behavior is addressed in the specification for that script.

The indicator for whether loads are split is the presence of a vehicle directory (and, by extension, a vehicle timeline report file in that directory) within the weekly schedule directory at the time backstop is run. One log file is generated for timeline report present in the weekly schedule directory.

Initially a check is performed to verify that the schedule directory is free of any residual outputs except for the folder Thermal_Data in the output folder and the three files moved to the output folder by the untar_question when the ORViewer is used to generate the schedule (see untar_question). Notify the user if any of the files moved by the untar question are missing. The script looks at the place(s) where backstop file(s), log file(s), and test outputs are to be created, and verifies that no such items exist at those locations. If any such items are present prior to the start of backstop, the user is notified and prompted as to whether to continue. **It is strongly recommended that any existing residuals be cleaned up and backstop rerun.**

Next, a check is performed on all files (excluding subfolders of the matlab folder) containing the string "Schedule generated by MATLAB Tools" followed by a Time Stamp (YYYY:DDD:HH:MM:SS.MMM). If all of the Time Stamps match, the user sees the message "All Time Stamps match". Otherwise, the user sees the message "Not all Time Stamps match" followed by each file name and the Time Stamp appearing in that file (sorted by Time Stamps). In this case, the user sees the prompt as to whether to continue.

The following is the general progression of the workflow for the backstop tool.

History Update
Independent of whether the loads are split, the user is prompted to update the history, and must indicate the schedule directory containing the backstop file from which records are to be added to the history files. Regardless of whether that schedule directory contains split loads, the backstop file matching the pattern CR*.backstop is used; this file is presumed to represent the combined backstop file (in the case of split loads) or the sole backstop file (if non-split loads).

Import – Run once per TLR file – Results appear only in the respective log file

- backstop_validate
- backstop_invert
- backstop_timecheck
- backstop_historycheck

Split Checks – Run only for split loads – Results appear in all operative log files

- backstop_cmdcatcheck
- backstop_synccheck

Export – Run once per TLR file – Results appear only in the respective log file

- Report Generation (results not aggregated)
  - State report by-attitude (backstop_rptstate)
  - State report by-comm (backstop_rptstate)
  - MCC input (backstop_mccexport)
- Tests – Run in order; individual test results reported along with an overall result of `ALL TESTS PASSED` or `ONE OR MORE TESTS DID NOT PASS`
  - backstop_acacheck
  - backstop_mechcheck
  - backstop_acispkt
  - backstop_lonenpm
  - backstop_cmdcheck
  - backstop_iumodecheck
  - backstop_dither
  - backstop_nmmdurationcheck
  - backstop_npmdurationcheck
  - backstop_unloadcheck
  - backstop_eclipsecheck
  - backstop_msscheck
- backstop_grep
- FOT Checklist Creation (backstop_fotchecklist)

Directory Listings – Run once per TLR file - Results appear only in the respective log file

- Appropriate top level directory and output directory
- mps/ode/characteristics and mps/ode/constraints subdirectories

backstop Weekly Schedule Directory Structure

`fot/`
`History/` (added after backstop completes)
`log/`
- `combined.log`

`mps/`
`output/`
- combined or sole set of test outputs, reports, etc., e.g. `TEST_acacheck.txt`
- relevant non-backstop items may be added after backstop completes

`sausage/` (added after backstop completes)
`vehicle/` **(SPLIT LOADS ONLY)**
- `VRXXX_YYYY.tlr` (vehicle TLR)
- `VRXXX_YYYY.backstop` (vehicle backstop file)

- V_FOT_Load_Review_Checklist_MMMDDYYYX.txt (vehicle FOT checklist file)
- log/
  - vehicle.log
- output/ (vehicle set of test outputs, reports, etc., e.g. V_TEST_acacheck.txt; relevant non-backstop items may be added after completion of backstop

CLXXX_YYYY.cld (command load files)
CRXXX_YYYY.tlr (combined or sole TLR file)
CRXXX_YYYY.backstop (combined or sole backstop file)

Main Line Scripts

backstop_acacheck

History Items

ATTITUDE

NOTE: If provided a vehicle-only backstop file, identified by its name, the script will return `[N/A]`; the non-run will be indicated in the output file. This script is not effective against the vehicle-only backstop file because it relies on ObsID commands, which do not normally appear in a vehicle-only backstop file.

General Discussion

The script is built around iteration of the backstop file. As each `AOMANUVR` command is encountered, the current and previous quaternions (from `MP_TARGQUAT` events) are used to calculate the maneuver end time. As each star catalog event (`MP_STARCAT`) is encountered, information from that event is saved. At the cue for reporting and testing, only the most-recently-encountered tracking information is used.

Star catalog information is reported and tested upon detection of the end of a maneuver for which autotransition was enabled at maneuver end time. Additionally,
should the backstop file end with a maneuver in progress (e.g. as of the timestamp of the last backstop record) the last-encountered star catalog will be tested based on the information available at that time.
Note: a crucial assumption of this script is that the OBSID and star catalog information correspond at the time of the reporting and testing.

Finally, a test is run to verify that the star catalog was converted with SAUSAGE; the test checks that, for all lines in the md*dot file, every line containing the string `ACQ` or `FIDSEL` have an M in the fourth to last character.

StarCatalog Library

The star catalog object/library provides auxiliary functions for processing and printing of star catalog information, based on data from the backstop file as well as from the guide star summary file.

In essence, StarCatalog represents a star catalog.

The following are items of interest with respect to a star catalog:

- `IDX` (1-16)
- `IM#`
- `TYPE` (`ACQ`, `GUI`, `BOT`, `FID`, `MON`, `NUL`)
- `IMGSZ` (4x4, 6x6, or 8x8)
- `MAG` (thousandths, obtained from guide star summaryfile)

- MINMAG (thousandths)
- MAXMAG (thousandths)
- YANG (tenths; arcseconds)
- ZANG (tenths; arcseconds)
- DIMDTS
- RESTRK
- HALFW

Population
The following fields are taken from the backstop event data record: IMNUM, TYPE, IMGSZ, MINMAG, MAXMAG, YANG, ZANG, DIMDTS, RESTRK. Adjustments are made as follows:

- TYPE is adjusted to NUL if MAXMAG and MINMAG are both 0 **FORMERLY TYPE ALSO HAD TO BE 0 – need to return that criterion?**
- YANG and ZANG are converted from radians to arcseconds
- Star magnitudes are obtained from the mg*sum file and included as the MAG field, based on the ObsID in play. If the ObsID is unknown, or if the guide star summary file does not contain information corresponding to the current ObsID, star magnitudes are unknown and will be reflected as such; tests will not be run if this is the case; see below.
- HALFW is computed as follows for each star: for types NUL and MON, 0; otherwise, $(40-35(RESTRK))*DIMDTS+20$

Note that accurate star catalog population depends on the correspondence of the ObsID and event data information from the backstop file.

Output
The following fields are printed for each star: IDX, IM#, TYPE, IMGSZ, MAG, MAXMAG, YANG, ZANG, DIMDTS, RESTRK, HALFW. Stars of type NUL are not printed. Additionally, for each quaternion encountered (event of type MP_TARGQUAT), the four components of the quaternion are presented, along with the RA, Dec, and Roll in degrees; the latter three items are computed by the AttitudeCompLib library.

Star Catalog Tests
DDTS Entry 6975 detailed a list of tests to be run against a star catalog. Further discussions from that point led to the following list that was actually implemented as part of backstop_acacheck. The tests are not run if the star catalog was not successfully populated (at the time of this writing, this would be due to the fact that star magnitudes could not be obtained from the guide star summary file.)

First, the star catalog is printed (according to the above discussion).

The following set of definitions will be used throughout the balance of this document:

- ACQ Stars: The group of star catalog entries with type ACQ or type BOT
- GUI Stars: The group of star catalog entries with type GUI or type BOT
- MON Windows: The group of star catalog entries with type MON

- FID Lights: The group of star catalog entries with type `FID`
- Brightness threshold: magnitude `10.3`

A set of statistics is then printed for the star catalog:

- ACQ Stars: number, median magnitude, the faintest of the group, and the number of stars that have magnitude less than or equal to the brightness threshold
- GUI Stars: number, median magnitude, the faintest of the group, and the number of stars that have magnitude less than or equal to the brightness threshold
- FID Lights: count
- MON Windows: count

Finally, the set of tests is run against the star catalog. **Please note that if particular condition causes a test to PASS, all other conditions should be assumed to FAIL.**

1. **If the number of ACQ stars is 8, PASS**.
2. **Combination of other entries in the star catalog based on OBSID for the star catalog.** From the above discussion, we can conclude that the OBSID at the time these tests are run will be extracted from the last OBSID record in the backstop prior to the tests being run (the condition prompting testing of a star catalog is discussed above).
    a. OBSID >= 38000: **If the number of GUI stars plus the number of MON windows is 8, PASS.**
    b. OBSID < 38000: **If the number of FID lights is 3, and the number of GUI stars plus the number of MON windows is 5, PASS.**
3. **If the median magnitude of the group of ACQ stars is less than the brightness threshold, PASS.**
4. **If the median magnitude of the group of GUI stars is less than the brightness threshold, PASS.**
5. **Combination of readouts based on the OBSID for the star catalog**
    a. OBSID >= 38000: **If all slots have an 8x8 readout, PASS.**
    b. OBSID < 38000: **If all FID Lights and MON Windows have an 8x8 readout, and all ACQ Stars and GUI Stars have 6x6 readouts, PASS.**
6. **MON Window test based on OBSID for the star catalog:**
    a. OBSID >= 38000: If **DTS IMNUM matches that of the MON window (e.g. self), PASS**
    b. OBSID < 38000: If **DTS for MON refers to a guide star with magnitude equal to that of the brightest guide star, PASS.**

backstop_acispkt

History Items

None

For each backstop file record containing the string `ACISPKT`, obtain its `WORDS` event data field, which is explained in commentary as the number of 16-bit ACIS commands in the ACIS packet.

If that number is greater than 14, compute the maximum / worst-case number of VCDUs needed for the ACIS packet as follows:

```
int((WORDS-15)/56) + 2
```

From that, compute the maximum ending VCDU for the ACISPKT.

Check the VCDU of the record immediately following (with the exception of records containing the string `ORBPOINT`) and compare with the maximum ending VCDU for the ACISPKT (accounting for a VCDU rollover if necessary, see below). If the latter is greater, a violation is raised, resulting in a failed test.

This script determines that a rollover occurred between the `ACISPKT` command and the subsequent command as follows: if the `ACISPKT` VCDU is greater than 2**23 and the following command VCDU is less than 2**23. This method was determined to be reasonable for the purposes of this test.

backstop_cmdcatcheck

History Items

None

Note: This script takes two backstop files as input (combined and vehicle-only), and is intended to be run against split loads only.

This script verifies that commands are placed correctly within the vehicle and observing sets of commands, as represented by backstop records.

The vehicle set is simply the set of records from the vehicle backstop file. The observing set is the set difference of the records from the combined backstop file and the records from the vehicle backstop file.

The check is facilitated by the use of the auxiliary file command_cat.txt. The file, for selected mnemonics and selected backstop record types, indicates the set(s) in which each item is expected to be found. Some items are expected only in the vehicle set, others are expected in the observing set, and still others can be found in either set.

Each record in each set is tested to verify whether it is placed properly, according to the following algorithm:

1. If the backstop record representing the command has a TLMSID field, and the value of that field is a mnemonic listed in command_cat.txt, determine based on the categorization of that mnemonic whether the command is properly placed.
2. Otherwise, if the backstop record type for the record is listed in command_cat.txt, determine based on the categorization of the record type whether the command is properly placed.

A WARNING is generated if a command is improperly placed. A FAIL is generated if the command is not in command_cat.txt. .

backstop_cmdcheck

History Items

None

The backstop file is searched for disabled commands, the mnemonics for which are listed in the disabled commands file (mentioned above). The output of the script reports a count of disabled commands found, along with the time and MSID of each such command. The script returns `PASS` if zero disabled commands were found in the backstop file, `FAIL` otherwise.

backstop_dither

History Items

None

For each record with event type `MP_DITHER`, extract the following parameters:

Event Data Field Name	Value
`COEFP`	Z amplitude (radians)
`COEFY`	Y amplitude (radians)
`RATEP`	Z frequency (rad/sec)
`RATEY`	Y frequency (rad/sec)

Convert Y amplitude and Z amplitude to arcseconds.
Compute Y period and Z period by dividing `2π` by the corresponding frequency.

Compute Y speed by multiplying Y frequency by Y amplitude (in arcseconds).
Compute Z speed in the same manner using Z frequency and amplitude.

Compute maximum speed as follows:

$$\max speed = \sqrt{(Yspeed)^2 + (Zspeed)^2}$$

The following checks are conducted based on the above calculations:

- max speed <= 2.0 arcsec/sec (or FAIL)
- Y amplitude <= 65.0 arcsec (or FAIL)
- Z amplitude <= 65.0 arcsec (or FAIL)

If these criteria are satisfied, a case name is assigned to the set of parameters using the table below. If the parameters do not match the values in this table, then the case name will be Large Dither if Y amplitude or Z amplitude is greater than 30.0 arcsec or else Non-standard Dither.

Dither Case	Y amplitude (arcsec)	Z amplitude (arcsec)	Y Period (sec)	Z Period (sec)
Standard ACIS	8.0	8.0	1000.0	707.1
Standard HRC	20.0	20.0	1087.0	768.6
Crab Observation	1.0	1.0	1000.0	707.1
ACIS Subarray	8.0	4.0	1000.0	707.4
ACIS Contaminant	8.0	64.0	1000.0	2647.0
Zero Amplitude	0.0	0.0	Any Value	Any Value

For the case of Large Dither or ACIS Contaminant (a subcase of Large Dither), the following criteria must also be satisfied (exact times and tolerances are shown in parenthesis).

a. The case for the previous dither command must be Standard ACIS or Standard HRC
b. The dither enable command following the dither command must be 5 minutes (300-10.251± 0.258 sec) after the end of the maneuver preceding the dither command.
c. The case for the subsequent dither command must be Standard ACIS or Standard HRC
d. The dither enable command following the subsequent dither command must be 5 minutes (300+10.251± 0.258 sec) before the start of the maneuver following the dither command.

If any of these criteria are not satisfied, result of the dither command is FAIL. If all of these criteria are satisfied, the result of the dither command for the ACIS Contaminant case is PASS and for the Large Amplitude case is WARN. For the other cases listed in the table above, the result of the dither command is PASS. For the Non-standard Dither case the result is WARN. For all dither commands, there must be an enable dither command 1 second (1.0± 0.258 sec) after the dither command or the result is FAIL.

Special cases with dither commands for case Large Dither or ACIS Contaminant

- If the dither command is the first in the backstop file, the first criteria (a) cannot be checked and the result is WARN.
- If the dither command is the last in the backstop file, the criteria (d) cannot be checked and the result is FAIL.
- If the dither command is not followed by a maneuver in the backstop file the criteria (e) cannot be checked and the result is WARN.

These checks are made against every `MP_DITHER` command, and such a command passes if ALL of the above checks are true. Overall the test passes if all `MP_DITHER` commands in the backstop file pass.

The script also prints all occurrences of `AOENDITH` and `AODSDITH`. If the dither command is followed by an enable command without a parameter command, the message script prints the message "No Parameter Command" preceded by a WARN or PASS. A WARN is used if there are no previous parameter commands otherwise PASS is used.

If none of the above-mentioned commands occur in the backstop file, output a message indicating this, and indicate that the test passed.

backstop_eclipsecheck

History Items

ATTITUDE
ECLIPSE_TIMERS

The following is an analysis of the functionality provided in the backstop_eclipsecheck script. This document attempts to provide an analysis "closer to the code" than was the analysis of the current version of the script, and is intended to provide a discussion of the methods employed in implementing the elements of the agreed-upon specification. Diagrams and tables are provided for the illustration of data structures and the like.

Unless otherwise noted, a **WARNING** does not prevent the script from continuing; an **ERROR** or **FAILURE** will stop the script.

I. Process the STK File (ECLIPSE.txt)

The STK file is processed according to its construction. The major change to its format is the new 'TYPE' identifier, which can be PSHORT, NSHORT, NORMAL, or NOCONN. Please refer to Appendix 1 at the end of this document, which presents the 2008 edition of ECLIPSE.txt along with some remarks about its format.

This file format leads to a state-based parser which follows the state diagram below.

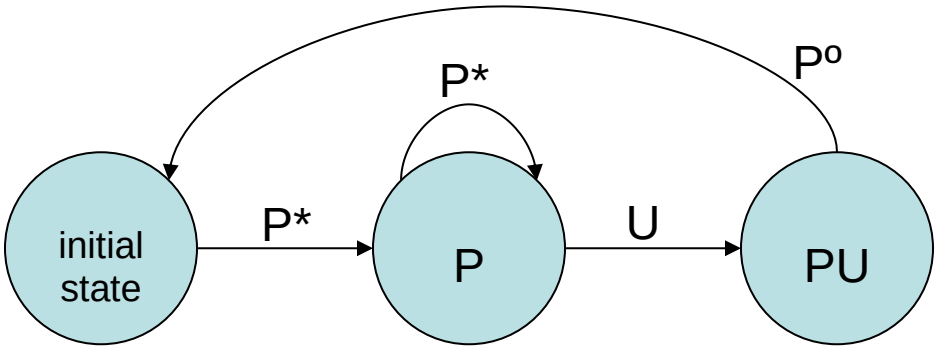

Figure 1. State diagram for STK file parsing

P* is a penumbra line with entry timer & type; Pº is a penumbra line without entry timer & type; U is an umbra line.

(Note that all other state transitions are invalid.)

In the event of any invalid state transition or parsing problem (the majority of which are simple defects in the file), an **ERROR** is cited and the script stops trying to process the file.

The last step in processing is to remove entries in this array that do not fit into the range covered by the current schedule (which is the span of the first and last records in the backtop file). By performing this step, the ECLIPSE.txt file can contain (perpetually) at least two eclipse seasons' worth of data, but only the eclipses relevant to the current schedule will be checked. **ASSUMPTION: the ECLIPSES.txt file is presented in chronological order**.

Upon parsing completion, we have the STK eclipses array which is shown below. The STK file provides, for the purposes of eclipsecheck: start time, end time, entry timer, and type.

S_START	S_END	S_ENTRYTIMER	S_TYPE
2007358.054715500	2007358.065451800	1639	NORMAL
2007360.211330460	2007360.223941020	1405	NORMAL
2007363.124604260	2007363.141800280	1405	NORMAL
2008001.042445240	2008001.054920420	1405	NORMAL
2008003.201300150	2008003.210916970	2341	NORMAL

Figure 2. STK Eclipses Array

II. Process the OFLS File (.backstop)

The method employed for finding the eclipses laid out in the backstop file begins with a simple grep for commands indicative of eclipse(s) being present in the file. These commands are as follows:

- PENTRY/LSPENTRY ("entry commands")
- PEXIT/LSPEXIT ("exit commands")
- EOECLETO/EOECLSTO/EOECLST2
- EOESTECN

Processing continues (e.g. the script actually runs) if even one of the above-listed commands is found. Based on the results of this grep, lists of entry and exit commands are formed, along with lists of entry timer commands (including the last entry timer command record from the history file). As of backstop 3.0, ECLIPSE_TIMERS.txt only tracks eclipse entry timer commands (exit timer commands are no longer of interest; a **WARNING** is raised if any exit timer commands are found: EOECLSTO/EOECLST2).

The following checks are performed prior to assembling the OFLS eclipses array:

1. Same number of entry (LSPENTRY and PENTRY) and exit (LSPEXIT and PEXIT) commands. If different, an **ERROR** is cited and processing stops.
2. Correct sequencing of (LS)PENTRY and (LS)PEXIT commands. This is accomplished with a state-based parser. All state transitions not shown in the state diagram below are invalid and result in an **ERROR**; processing stops.

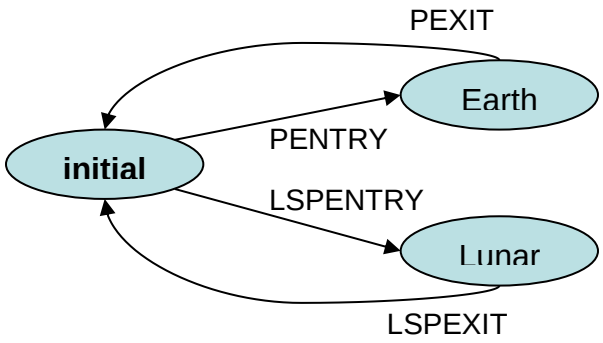

Figure 3. State diagram for sequencing of entry/exit commands

3. Comparison of numbers of entry timer commands and entry-exit pairs. At this point each entry-exit pair is regarded as an eclipse. If there are at least as many entry timer commands as number of entry-exit pairs, this may indicate a superfluous entry timer set command, and a **WARNING** is displayed. (The likely explanation for extra entry timer commands is that we tend to reset each timer to its default value immediately after an eclipse which uses a non-default value for the entry timer. This information was provided by Rino.)
4. If any eclipse timer is set twice with commands having identical timestamps, an **ERROR** is cited and processing stops.
5. If any eclipse timer command occurs less than 24 hours prior to an entry command (PENTRY/LSPENTRY), a **WARNING** is cited.
6. Choose the eclipse timer command/setting associated with each eclipse. This would be the closest (time-wise) command of each type occurring before a PENTRY. If, for a given timer, there is no command (from the list described above) that occurs before a given PENTRY, an **ERROR** is cited and processing stops. The script will report, for each eclipse, the eclipse timer command chosen (timestamp and value) for each type of eclipse timer. **Note: it is extremely unlikely that this error would be raised because there will always be historical information available. Without historical information the script would simply raise an error because the history lookup failed.**

Assuming processing did not stop, the OFLS eclipses array is intact and takes the following format initially (as information will be added later). The table initially contains, for each eclipse, the start and end times along with the record time and value for each type of eclipse timer.

O_START	O_END	O_ENTRYTIMER O_ENTRYTIMER_RECTIME	O_EXIT1TIMER O_EXIT1TIMER_RECTIME	O_EXIT2TIMER O_EXIT2TIMER_RECTIME
		1639	4683	5151
2007358.054705258	2007358.065455258	2007356.050001000	2007356.050002000	2007356.050001000
		1405	4683	5151
2007360.211324317	2007360.223944317	2007358.200001000	2007358.200002000	2007358.200003000
		1405	4683	5151
2007363.124604126	2007363.141804126	2007358.200001000	2007358.200002000	2007358.200003000
		1405	4683	5151
2008001.042447883	2008001.054927883	2007358.200001000	2007358.200002000	2007358.200003000
		2341	4683	5151
2008003.201305690	2008003.210925690	2008001.190001000	2008001.190002000	2008001.190003000

Figure 4. Initial OFLS Eclipses Array

III. Reconcile STK and OFLS Data

With the STK Eclipses and the OFLS Eclipses arrays having been constructed, we compare common data in order to determine whether the STK file agrees with the backstop file. The following checks are performed.

1. Numbers of eclipses – if not equal, **FAIL** and reconciliation stops.

Assuming this first check passes, we compare corresponding entries in the arrays (first entry in STK array with first entry in OFLS array, etc.)

2. Eclipse start and end times are within tolerance, which is currently 60 seconds. If not, **WARN** with reconciliation continuing.
3. Entry timer settings match exactly, and if not, **ERROR** with reconciliation continuing.

For this reconciliation, the result for each eclipse is reported (with the high watermark of conditions encountered reported as the result). The high watermark of all of the individual eclipse conditions (not the high watermark of the individual high watermarks) is reported as the overall result.

IV. backstop file iteration and command checks

Assuming reconciliation proceeded to completion (at worst **WARN**), we continue to the main body of the script which revolves around parsing of the backstop file. First, we complete the OFLS eclipses array, providing the type (from the STK eclipses array) and the extended eclipse period start (**start time minus 1201 seconds**) and end times (**end time plus 1200 seconds**). The final OFLS eclipses array is as follows:

O_START O_EXT_START	O_END O_EXT_END	O_ENTRYTIMER O_ENTRYTIMER_RECTIME	O_EXIT1TIMER O_EXIT1TIMER_RECTIME	O_EXIT2TIMER O_EXIT2TIMER_RECTIME	O_TYPE

2007358.054705258 2007358.052704258	2007358.065455258 2007358.071455258	1639 2007356.050001000	4683 2007356.050002000	5151 2007356.050001000	 NORMAL
2007360.211324317 2007360.205323317	2007360.223944317 2007360.225944317	1405 2007358.200001000	4683 2007358.200002000	5151 2007358.200003000	 NORMAL
2007363.124604126 2007363.122603126	2007363.141804126 2007363.143804126	1405 2007358.200001000	4683 2007358.200002000	5151 2007358.200003000	 NORMAL
2008001.042447883 2008001.040446883	2008001.054927883 2008001.060927883	1405 2007358.200001000	4683 2007358.200002000	5151 2007358.200003000	 NORMAL
2008003.201305690 2008003.195304690	2008003.210925690 2008003.212925690	2341 2008001.190001000	4683 2008001.190002000	5151 2008001.190003000	 NORMAL

Figure 5. Final OFLS Eclipses Array (PENTRY times are in bold)

The iteration of the backstop file is conducted as follows – for each eclipse in the OFLS eclipses array, the script picks up where we left off in the backstop file (or, in the case of the first eclipse, starts from the beginning of the backstop file) and continues until a record is encountered with a timestamp later than the end of the extended eclipse period for the current eclipse. If the last record is earlier than the end of the extended eclipse period, a FAIL is raised and the following message is printed:

> Did not check eclipse ##. If this is the last eclipse, this is likely due to the fact that the backstop file ended in the middle of the extended eclipse period for the last eclipse. Regardless this condition may impact the entire script run.

If the last record is within 11 minutes of the Penumbra exit and the eclipse type is not NOCONN, the following warning message is printed:

> SCS 33 will be executed after the time of the last record in the backstop file. This may result in the incorrect analysis of the Enable and Disable of the SPM in the subsequent load.

Important note: this means that the iteration of the backstop file will not necessarily be complete. Iteration will stop at the first command occurring after the end of the extended eclipse period for the last eclipse, which will not necessarily be the last command in the backstop file. (If, at some later point, there is a desire to continue to the end of the backstop file and report on commands that occur after the eclipse period for the last extended eclipse, code changes would be necessary, though not a huge effort. Based on the spec there is likely no need for this functionality now.)

The following checks are performed:

1. **Last maneuver prior to current (extended) eclipse (period)** (executed when first record falling after the start of the extended eclipse period is found). Calculates maneuver end time and checks the following:
   - For NORMAL Eclipses, 70<= pitch <=135 (or FAIL)
   - For NSHORT eclipses, 90 <= pitch <= 115 (or **FAIL**)
   - For PSHORT eclipses, 90 <= pitch <= 115 (or **FAIL**)-1 <= off-nominal roll <= 1 (or **FAIL**)
   - Autotransition enabled prior to last maneuver (or **FAIL**)

- Autotrans ENAB command did not occur since the start of the maneuver (or **WARN**)
- Autotrans DISA command did not occur since the start of the maneuver (or **FAIL**)
- Maneuver ends before extended eclipse period start time (or **FAIL**)
- For PSHORT and NSHORT Eclipses, maneuver ends no earlier than 30 minutes prior to PENTRY/LSPENTRY (or **FAIL**)

2. **Check commands for this eclipse (e.g. from this loop)** (executed when first record falling after the end of the extended eclipse period is found) Checks the following:
   - **WARN**/**ERROR** commands as flagged below, sections 3 and 4 (according to the spec)
   - Time-specific commands as outlined in the spec (with respect to PENTRY time; see below for the commands themselves):
     - -10:00 Momentum Unload Disable
     - -05:00 SPM Disable
     - -02:00 Eclipse Entry
     - -01:57 Select Database (PSHORT only)
     - -01:56 Init Thermal Control (PSHORT only)
     - -01:55 SCS 71 (PSHORT only)
     - -01:50 SCS 72 (PSHORT only)
     - -01:45 SCS 73 (PSHORT only)
3. **Commands/events tracked independent of extended eclipse period**
   - AOMANUVR – note autotransition state; accounting for last maneuver (as any maneuver could be the one right before the next eclipse); must not occur between the start of the extended eclipse period and (LS)PEXIT or **ERROR**
   - AONM2NPD/E – if occur within extended eclipse period: **ERROR** for a DISA, **WARN** for ENAB; script does accounting for last maneuver check (including continuous tracking of autotransition state)
   - MP_TARGQUAT – continuously tracking most recent two quaternions for maneuver check
4. **Commands checked ONLY within the extended eclipse period – time-specific commands**

   For each occurrence of a time-specific command encountered during the extended eclipse period, we call on a separate object to determine if it was found in the expected situation, based on three attributes:

   (i) correct time with respect to PENTRY for the current eclipse
   (ii) for a non-PSHORT eclipse, the command is not exclusive to PSHORT eclipses
   (iii) an occurrence of the command has not already been found satisfying (i) & (ii).

   Each occurrence of a time-specific command will be evaluated based on these three attributes, and if all three are satisfied, the command is regarded as 'found', otherwise an **ERROR** will be cited. In particular, based on (iii), all occurrences of the command subsequent to the one which resulted in the command being marked 'found' (within the extended eclipse period for this eclipse) will result in **ERROR**. Errors are based on occurrences of commands are thus independent of each other (e.g. a command can be marked as found for an eclipse but an additional occurrence of a command can still result in an error

for that eclipse).

- Time-specific command checks
  - AOFUNCDS / AOPCADSD 21 (Momentum Unload Disabled)
  - AOFUNCDS / AOPCADSD 30 (SPM Disabled)
  - EOESTECN (Eclipse Entry)
  - 4OTTCDBT – **PSHORT ONLY** (Select Database)
  - 4OTELTCR – **PSHORT ONLY** (Init Thermal Control)
  - COACTSX / COACTS1 71/72/73 – **PSHORT ONLY** (SCS 71,72,73)
- Non-time-specific commands checked
  - AOFUNCEN / AOPCADSE 21 or 30 – **ERROR**
  - AOMUNLDS/AOMUNLEN/AODESMOS/AOMUNLGR – **ERROR**
  - AONMMODE – **ERROR** if prior to PEXIT

Based on the accumulated warnings and errors (if any), the high watermark per eclipse is presented. As before, the overall result of the backstop file iteration is presented as the high watermark of all warnings and errors found (not just the high watermark of the high watermarks).

The overall result of the script is, again, the high watermark of all conditions encountered during the various components of the script (assuming that all were completed – as is noted above, the script exits in the face of some failures).

Appendix 1. ECLIPSE.txt through the end of 2008.

Satellite-Chandra: Eclipse Times 2008/094 Epoch

Start Time (UTCJFOUR)	Stop Time (UTCJFOUR)	Duration (sec)	Current Condition	Obstruction	Entry Timer	Type
037/2008 22:15:42.58	037/2008 22:46:29.67	1847.096	Penumbra	Moon	4215	PSHORT
164/2008 19:48:58.69	164/2008 19:54:28.32	329.633	Penumbra	Earth	1639	NORMAL
164/2008 19:54:28.32	164/2008 20:08:06.74	818.419	Umbra	Earth		
164/2008 20:08:06.74	164/2008 20:13:16.67	309.924	Penumbra	Earth		
167/2008 11:08:32.97	167/2008 11:10:59.88	146.916	Penumbra	Earth	1171	NORMAL
167/2008 11:10:59.88	167/2008 11:51:25.03	2425.148	Umbra	Earth		
167/2008 11:51:25.03	167/2008 11:53:32.89	127.857	Penumbra	Earth		
170/2008 02:34:53.20	170/2008 02:36:49.97	116.766	Penumbra	Earth	1171	NORMAL
170/2008 02:36:49.97	170/2008 03:27:35.91	3045.938	Umbra	Earth		
170/2008 03:27:35.91	170/2008 03:29:14.26	98.351	Penumbra	Earth		
172/2008 18:04:53.20	172/2008 18:06:37.13	103.931	Penumbra	Earth	1171	NORMAL
172/2008 18:06:37.13	172/2008 19:02:11.57	3334.436	Umbra	Earth		
172/2008 19:02:11.57	172/2008 19:03:37.71	86.146	Penumbra	Earth		
175/2008 09:37:18.03	175/2008 09:38:56.27	98.237	Penumbra	Earth	1171	NORMAL
175/2008 09:38:56.27	175/2008 10:35:40.69	3404.420	Umbra	Earth		
175/2008 10:35:40.69	175/2008 10:37:01.75	81.057	Penumbra	Earth		
178/2008 01:10:59.00	178/2008 01:12:36.54	97.544	Penumbra	Earth	1171	NORMAL
178/2008 01:12:36.54	178/2008 02:07:28.66	3292.124	Umbra	Earth		
178/2008 02:07:28.66	178/2008 02:08:49.62	80.959	Penumbra	Earth		
180/2008 16:44:58.58	180/2008 16:46:40.73	102.147	Penumbra	Earth	1171	NORMAL
180/2008 16:46:40.73	180/2008 17:36:43.21	3002.481	Umbra	Earth		
180/2008 17:36:43.21	180/2008 17:38:09.36	86.155	Penumbra	Earth		
183/2008 08:19:27.21	183/2008 08:21:21.90	114.687	Penumbra	Earth	1171	NORMAL
183/2008 08:21:21.90	183/2008 09:03:19.94	2518.035	Umbra	Earth		
183/2008 09:03:19.94	183/2008 09:04:59.23	99.296	Penumbra	Earth		
185/2008 23:57:13.44	185/2008 23:59:43.22	149.778	Penumbra	Earth	1171	NORMAL
185/2008 23:59:43.22	186/2008 00:28:43.25	1740.033	Umbra	Earth		
186/2008 00:28:43.25	186/2008 00:30:58.22	134.976	Penumbra	Earth		
188/2008 15:44:46.58	188/2008 15:49:37.29	290.707	Penumbra	Earth	1171	NOCONN
347/2008 12:59:54.73	347/2008 13:04:33.64	278.904	Penumbra	Earth	1405	NORMAL
347/2008 13:04:33.64	347/2008 13:50:42.24	2768.599	Umbra	Earth		
347/2008 13:50:42.24	347/2008 13:56:01.08	318.844	Penumbra	Earth		
350/2008 04:23:39.08	350/2008 04:26:55.74	196.660	Penumbra	Earth	1171	NORMAL
350/2008 04:26:55.74	350/2008 05:38:07.87	4272.126	Umbra	Earth		
350/2008 05:38:07.87	350/2008 05:42:06.27	238.403	Penumbra	Earth		
352/2008 19:53:27.44	352/2008 19:56:25.99	178.549	Penumbra	Earth	1171	NORMAL
352/2008 19:56:25.99	352/2008 21:19:17.86	4971.872	Umbra	Earth		
352/2008 21:19:17.86	352/2008 21:22:59.98	222.115	Penumbra	Earth		
355/2008 11:28:00.47	355/2008 11:31:08.49	188.024	Penumbra	Earth	1171	NORMAL
355/2008 11:31:08.49	355/2008 12:54:44.78	5016.291	Umbra	Earth		
355/2008 12:54:44.78	355/2008 12:58:38.21	233.424	Penumbra	Earth		
358/2008 03:07:26.53	358/2008 03:11:20.00	233.473	Penumbra	Earth	1171	NORMAL
358/2008 03:11:20.00	358/2008 04:22:45.75	4285.750	Umbra	Earth		
358/2008 04:22:45.75	358/2008 04:27:26.54	280.789	Penumbra	Earth		
360/2008 18:56:59.60	360/2008 19:05:35.95	516.352	Penumbra	Earth	3278	NORMAL
360/2008 19:05:35.95	360/2008 19:34:31.58	1735.631	Umbra	Earth		
360/2008 19:34:31.58	360/2008 19:43:57.23	565.646	Penumbra	Earth		

In parsing, all blank lines will be skipped along with the lines in yellow, which are matched for their starting strings only ("Satellite-Chandra", "Start Time", and "-----"). **IMPORTANT:** changes to this file format or to the set of immaterial non-blank lines will necessitate code changes.

backstop_ephemcheck

History Items

EPHEMERIS

This script checks for correct sequencing of EPHEMERIS commands and PERIGEE orbit events, and verifies the orbital parameters comprising each EPHEMERIS command.

Sequencing

EPHEMERIS commands and PERIGEE orbit events must strictly alternate. Initial condition is taken from the EPHEMERIS history file, and is regarded as the last-encountered item in the relative sequence. The following conditions are raised based on checking the relative sequencing of these items:

- `WARN` if an EPHEMERIS command is followed by another EPHEMERIS command without an intervening PERIGEE orbit event.
- `FAIL` if a PERIGEE orbit event is followed by another PERIGEE orbit event without an intervening EPHEMERIS command.

Timing

An EPHEMERIS command must be within 30 seconds of each APOGEE orbit event.

- `FAIL` if there is no EPHEMERIS command within 30 seconds of any AOPGEE event.
- `FAIL` if there is no EPHEMERIS command after last APOGEE orbit event.

Orbital Parameter Verification

The EPHEM_CERT.txt file contains the expected times of each PERIGEE command in the backstop file, and for each such time, the expected values of the orbital parameters.

The check attempts to match each EPHEMERIS command in the backstop file with exactly one entry in the EPHEM_CERT file, based on their times, and within a tolerance of at most ten seconds. `FAIL` is raised for each EPHEMERIS command that matches fewer or more than one EPHEM_CERT entry. The EPHEM_CERT file is not checked to ensure that each entry matched an EPHEMERIS command, as an excess of data in the EPHEM_CERT file may be desirable.

The EPHEMERIS command parameters found in the backstop file are listed in the backstop_invert section of this document. Table 1. Shows the relationship between these parameters and those in the EPHEM_CERT.txt file. Two of the command parameters, which are expected to be fixed values (AOEPHEM1=8133401 and AOEPHEM2=8A3001A), are not found in the EPHEM_CERT.txt file. For each successful match, the values of the respective orbital parameters (found in Table 1) are compared. `FAIL` is raised for any orbital parameter for which a difference in value greater than 10^{-4}% is observed. The percent difference is determined by computing the absolute value of the difference in values and dividing it by the expected value of the orbital parameter (from EPHEM_CERT). Also, FAIL is raised if either of the two fixed EPHEMERIS command parameters are not exact matches to their expected values. The output will show the EPHEMERIS command parameter values and the values found in the EPHEM_CERT.txt file.

Command Parameter Name	EPHEM_CERT Name
AORATIO	GAUSS_C
AOARGPER	APERIG
AOECCENT	ECC
AO1MINUS	M_COS_I
AO1PLUS	P_COS_I
AOMOTION	MEANM
AOITERAT	NUM_ITER
AOORBANG	ORB_MOM
AOPERIGE	T_PERIG
AOASCEND	RANODE
AOSINI	SIN_I
AOSLR	SEMI_LR
AOSQRTMU	SQRT_MU_AE

Table 1. Parameter Names

backstop_fotchecklist

History Items

DITHER
SCS84
SPM

As one of the last steps in the backstop tool, this script generates a partially filled FOT Checklist for use in the load review process.

The following inputs are provided to this script:

- Main directory – "home base" for files of interest to this script for a given set of loads. Script expects to find the following in this directory:
  - **exactly one** backstop file (matching pattern `*.backstop`) at the top level
  - `output` directory at the top level
  - `log` directory at the top level, and **exactly one** log file therein (matching pattern `*.log`).

  Script raises `ERROR` and exits `FAIL` if any of these conditions are not satisfied.
- (optional) – String to be prepended to the standard FOT Checklist file name. File naming is discussed below; the typical application of this option is to distinguish the vehicle FOT checklist for split loads.

The FOT checklist is generated from a template, which is stored in the backstop `aux` directory (`FOT_CHECKLIST_TEMPLATE.txt`), and which is configured separately from backstop software. The template is populated by the `backstop_fotchecklist` script, which is configured with backstop software.

The template, with the exception of items to be populated by the script, reflects the formatting of the FOT checklist. Placeholders for items to be populated by the script are in the form of tags, enclosed by angle brackets, such as `<DATE>`; tags are replaced by an item computed or generated by the script. All tags must be known to the script, and as such, adding a new tag or changing the computation for a particular tag requires a software change. If a tag is no longer needed, its usage can cease (rather than removing from the code); a software change is not needed in that case.

The following is the list of tags known to the script, along with the information which replaces each. New tags should be comprised of all capital letters. Spaces are not allowed, nor are commas (see below for the use of a comma within a tag for substitution mode specification).

Tag	**(Source) Description**
`<DATE>`	mm/dd/yy at runtime
`<TIME>`	HH:MM at runtime (local machine time; at OCC this will be GMT)
`<LOADNAME>`	(Schedule directory name) MMMDDYYX
`<BACKSTOPFILE>`	backstop file parsed in generating FOT Checklist
`<SCH-START>`	(Schedule summary file) Schedule start time

<SCH-STOP>	(Schedule summary file) Schedule end time
<TS-YEAR-ONLY>	YYYY:000:00:00:00.000 (YYYY is the year at runtime)
<DISABLED-CMD-COUNT>	(TEST_cmdcheck.txt, which is output from backstop_cmdcheck script) Count of disabled commands
<FOT-REQUEST-NAME>	MOMMGT_mmmddyy.fot (MMMDDYY portion of <LOADNAME> above)
<DITHER-RANGES>	(backstop file) Formatted string indicates time ranges for which dither is disabled*, along with stock strings to be used for justification of dither disabled Relevant commands: AOENDITH and AODSDITH
<SCS84-RANGES>	(backstop file) Formatted string indicates time ranges for which SCS-84 is enabled* Relevant commands: COENASX for which COENAS1=84; CODISASX for which CODISAS1=84
<SPM-RANGES>	(backstop file) Formatted string indicates time ranges for which SPM is disabled* Relevant entries: Enable: • AOFUNCEN command for which AOPCADSE=30 -or- • 11 minutes after ORBPOINT of TYPE=PEXIT or LSPEXIT (assumes SCS 33 run 11 minutes after eclipse penumbra exit) when preceded by a SW_COMMAND of MSID=EOESTECN two minutes before the corresponding ORBPOINT of TYPE=PENTRY or LSPENTRY Disable: • AOFUNCDS for which AOPCADSD=30
<TLR-CMD-COUNT>	(backstop log file) Total number of commands in the .tlr file
<CLD-CMD-COUNT>	(backstop log file) Total number of commands among all the .cld files
<BS-FIRST-NON-ORBPT-TIME>	(backstop file) Time of first record that is not of type ORBPOINT
<RESULT-**TESTNAME**>	(backstop log file) Result of **testname**, e.g. from backstop_**testname**. The tag expresses in UPPERCASE the name of the test (test names are lowercase)

* The range string will be one of the following: an indication that the item is enabled throughout the schedule, an indication that the item is disabled throughout the schedule, or a list of ranges of times for which the condition cited above is met. The first range in the list may begin with `BOS` ("beginning of schedule"), if, based on historical data, the condition was met prior to the start of the backstop file; the last range in the list may end with `EOS`, if the condition continues to be met after processing the last record in the backstop file.

Tag Substitution

As mentioned above, the template represents the actual formatting of the FOT checklist with the exception of the tags placed therein. By default, a tag will be substituted for in the template, effectively, by deleting it and inserting the information it represents at the character position of the left angle bracket (<) of the tag. This mode of substitution will be referred to as **replacement**.

However, for formatting reasons, it may be desirable to substitute information for a tag by overwriting the tag with the desired information. This mode of substitution is referred to as **in-place** substitution. There are three possibilities for in-place substitution: either the tag is longer, shorter, or the same length as the information which will replace it.

- If the tag is longer, the portion of the tag beyond the end of the substituting information will be converted to spaces
- If the tag is shorter, the information beyond the end of the tag will overwrite whatever is on the line (for a well-designed line, it would only be spaces which were overwritten). If the information would go past the end of the line (based on the number of characters in the information and the number of characters on the line), replacement mode is used instead.
- If the tag is the same length, the information simply replaces it

The following examples demonstrate replacement versus in-place substitution. In this case, the information is shorter than the tag.

```
0123456789012345678901234567890123456789012345
The quick brown fox jumped over the <TAG> dog.   (Original)
The quick brown fox jumped over the lazy dog.    (Replace)
The quick brown fox jumped over the lazy  dog.   (In-place)
```

Another example demonstrates a well-formatted in-place substitution (based on a knowledge of the width of the replacement information). In this case the information is longer than the tag.

```
01234567890123456789012345678901234567890123456789012345678901234567890123
The quick brown fox jumped over the <TAG>     dog.       (Original)
The quick brown fox jumped over the lethargic     dog.   (Replace)
The quick brown fox jumped over the lethargic dog.       (In-place)
```

As mentioned above, replacement mode is the default. Either replacement mode or in-place mode can be specified as part of a tag as follows, with the tag name, followed by a comma, followed by a one-character specification of the mode (`R` for replacement mode, `I` for in-place mode). Spaces are not allowed in tags.

Thus, for a given tag, the following three specifications are possible. In all cases, the software will substitute for everything including the angle brackets.

<TAG,R> provides the information associated with TAG, with replace substitution mode.
<TAG> indicates the same (since R is the default)
<TAG,I> provides the information associated with TAG, with in-place substitution mode.

The script, backstop_fotchecklist, takes two arguments:

- The backstop log file, typically uie.log, to the point at which this script is run (since this script is run as part of backstop.pl)
- The backstop file

The script will exit with code FAIL (from ResultLib) if any of the following occurs:

- Incorrect number of arguments
- Unable to open backstop file
- Any problem in obtaining historical information for the above items
- Unable to open log file
- Unable to open FOT checklist template file
- Unable to open output file for writing

The script will report errors for the following conditions, but will continue. The code raised will be incorporated into the overall result of the FOT Checklist generation:

- FAIL – Schedule start and stop times could not be obtained
- FAIL – Disabled command count could not be obtained
- FAIL – Irregularity in generating the FOT request name (based on irregularity in the schedule directory name)
- FAIL – Could not determine the time of the first non-ORBPOINT record in the backstop file (presumably because all were ORBPOINT)
- FAIL – Unknown result string reported in the backstop file for a given test (something other than FAIL, WARN, PASS, or N/A)

In cases where the substitute information could not be determined (such as the errors above), the tag is substituted for in the template by ERROR, which is indicative of missing information in the FOT checklist. Such replacement will occur for any tag encountered for which substitute information is not available (or for an unknown tag), and FAIL will be raised (in addition to the errors which cause the information to be unavailable).

Checklist File Naming Convention
The conventional checklist file name is FOT_Load_Review_Checklist_<LOADNAME>.txt, where LOADNAME is MMDDYYX as taken from the name of the weekly schedule directory. This script allows for modification of this name (prepending of a provided string) in order to distinguish, for split loads, the vehicle and combined checklists.

NOTE: The checklist file is generated at the top level of the main directory provided as an argument to the script. If a file by the expected name exists at the top level of the main directory, it is deleted before creating a new file.

backstop_grep

History Items

None

This script greps for various items in the backstop file and produces several output files based on the results. The greps/outputs are as follows:

grep for (pattern)	Output file
`AONPMODE`	NPM_check.txt
`RMP`	RMP_check.txt
Patterns listed in `aux/SPM_patterns.txt`	SPM_check.txt
`AOMUNLGR`, `AODESMOS`	UNLOAD_check.txt

If the exit status of a particular `grep` indicates an error, this error will be reflected in the status of the individual grep (from the above table) as well as in the overall result for the script.

backstop_historycheck

History Items

ATTITUDE

This script performs two simple checks with regard to the backstop file and history data (as represented by the ATTITUDE history file).

1. There is at least one record in the ATTITUDE history file within 216000 seconds (60 hours) of the first backstop record.
2. There are no records in the ATTITUDE history file with timestamp falling after the first backstop record time.

Each check will raise `WARN` individually should a check fail; as with most other scripts, the overall result composes the results of the individual checks.

backstop_history (plus HistoryLookup)

The HistoryLookup object searches a set of history files for the most recent state of a set of spacecraft schedule-related items. The backstop_history script is responsible for updating those history files. As such, it makes sense to discuss the two together.

Record Composition
Most generally, history records are composed with a timestamp (in the format `YYYYDOY.HHMMSSXXX`) and the state of the item in question, separated by '|'. The exact formatting of lines has varied (with respect to whitespace) in past historical records, and as such, whitespace is trimmed when a history file is parsed.

All history records since backstop 3.0 are composed as follows:
"`<timestamp> | <state>`", e.g. one space separating the timestamp from the separator, and one space separating the separator from the state. Additionally, for any history item for which the state is comprised of multiple tokens (such as attitude, which has four), a single space separates the tokens. Whitespace trimming ensures backward compatibility.

Lookup
Clients in need of historical information construct the `HistoryLookup` object with a timestamp, and call for the individual history items of interest. Thus, the unavailability of a single history item (for instance, if a history file is missing) will only affect a given client if the client requires that item. The aforementioned timestamp is nominally the time
of the first record in the backstop file; the history record of interest is most recent prior to this timestamp. If all records in the file fall before the given timestamp, the last record is selected. If all records in the file fall after the given timestamp, `WARN` is raised. Clients are responsible for handling the result of the lookup for each item and proceeding accordingly.

Content / Updating History Files
In most cases, history files are updated by iteration of the backstop file and gathering information from relevant records; the backstop_history script actually performs this task for most history files. The table following this discussion lists each history file, its category (vehicle, observing, or not applicable) along with the type(s) of backstop records forming the basis of these updates, the number of tokens in and a description of the content of each record. Unless otherwise noted, each file is updated by the backstop_history script. Additionally, the table indicates whether records from the file can be obtained via HistoryLookup, and if so, any remarks regarding retrieval of such records via HistoryLookup.

The HistoryLookup library provides functionality for creation of individual history records, with the client of such functionality being backstop_history. Irregularities in history record creation generally results in an exit `FAIL`, given the need to preserve history file integrity.

If the history is being updated from non-split loads or from a combined backstop file, records will be added to all files managed by backstop_history, as derived from the backstop file. If the history is being updated based on the vehicle-only portion of split loads, records will only be added to those files with the vehicle category in the below table, as derived from the backstop file. Regardless of the source of the history update, a comment is added to every history file managed by backstop_history, indicating the

name of the schedule from which records were, the starting time of the schedule, and whether a vehicle-only backstop file was used as the history update source.

An additional caveat: HistoryLookup was designed to be most successful with a single category of records. For instance, the ATTITUDE history file contains nothing but quaternions; the DITHER history file contains the sequence of dither enables and disables. While the GRATINGS file records can indicate one of three states (HETG, LETG, or NONE), this is still one category of records: the grating state. In contrast, prior to backstop 3.0, the ECLIPSE_TIMERS history file tracked three different types of records – EOECLETO, EOECLSTO, and EOECLST2 (each of the three eclipse timers); with release 3.0, only EOECLETO records are now added. However, some consideration will be needed if, in the future, a more complex history file is desired.

History File	Cat	Basis for Update	#Tok	Record State Contents
ATTITUDE.txt	VEH	Event type MP_TARGQUAT	4	Contents of Event Data fields Q1, Q2, Q3, Q4 Single space separates them
ECLIPSE_TIMERS.txt	VEH	Event Type MP_ECL_TIM & TLMSID field is EOECLETO	2	TIMECNT and TLMSID fields, separated by a single space
EPHEMERIS.txt	VEH	Event type MP_EPHEMERIS and TLMSID field is AOEPHUPS Event type ORBPOINT and TYPE field is EPERIGEE	1	Depends on item: - AOEPHUPS --> EPHEMERIS - EPERIGEE --> PERIGEE
FIDSEL.txt	OBS	TLMSID is AFIDP	var	If MSID contains pattern AFLCX --> <MSID> X ON If MSID contains AFLCRSET --> <MSID> RESET
GRATINGS.txt	OBS	Event Type is COMMAND_SW & TLMSID is one of {4OHETGIN, 4OHETGRE, 4OLETGIN, 4OLETGRE}	2	Depends on TLMSID: - 4OHETGIN --> HETG HETGIN - 4OHETGRE --> NONE HETGRE - 4OLETGIN --> LETG LETGIN - 4OLETGRE --> NONE LETGRE
SIMTRANS.txt	OBS	Event type is SIMTRANS	1	POS field
SIMFOCUS.txt	OBS	Event type is SIMFOCUS	1	POS filed
RADMON.txt	OBS	Event type COMMAND_SW & TLMSID is one of {OORMPEN, OORMPDS}	2	Depends on TLMSID: - OORMPEN --> ENAB <TLMSID> - OORMPDS --> DISA <TLMSID>
DITHER.txt	VEH	Event type COMMAND_SW & MSID is one of {AOENDITH, AODSDITH}	2	Depends on MSID: - AOENDITH --> ENDITH <MSID> - AODSDITH --> DSDITH <MSID>
SCS84.txt	VEH	Event type COMMAND_SW & (Enable) TLMSID is COENASX with COENAS1 field = 84 (Disable) TLMSID is CODISASX with CODISAS1 field = 84	1	Depends on Enable/Disable: - Enable --> ENAB - Disable -> DISA
SCS98.txt	VEH	Event type COMMAND_SW & (Enable) TLMSID is COENASX with COENAS1 field = 98 (Disable) TLMSID is CODISASX with CODISAS1 field = 98	1	Depends on Enable/Disable: - Enable --> ENAB - Disable -> DISA
SPM.txt	VEH	Event type COMMAND_SW & (Enable) TLMSID is AOFUNCEN with AOPCADSE field = 30 (Disable) TLMSID is AOFUNCDS with AOPCADSD field = 30 Event type COMMAND_SW and TLMSID is EOESTECN 2 min. before Event type ORBPOINT & TYPE field PENTRY or LSPENTRY (Enable) 11 minutes after Event type ORBPOINT & TYPE field PEXIT or LSPEXIT	1	Depends on Enable/Disable: - Enable --> ENAB - Disable -> DISA

TIMEREF.txt	N/A	Manually (not associated with backstop file)	2	VCDU at the time of the records

backstop_interrupt

History Items

ATTITUDE
ECLIPSE_TIMERS
FIDSEL
GRATINGS
SIMTRANS
SIMFOCUS
RADMON
DITHER
SCS84
SCS98
SPM
(presumably, ALL ITEMS, though not required)

The backstop_interrupt script handles cleanup of the history files in the event of a schedule interruption. This script prompts the user for the type of interrupt and the associated time or times, depending on the interrupt type. The interrupt type is used in several ways:

- Keys the comment line inserted into a history file indicating the reason for an interrupt
- For selected interrupt type(s), artificial records are added to certain history files
- For selected interrupt type(s), different history files are truncated at different times

This script should be run in the operational history directory (as of the time of this writing, `/home/mission/Backstop/History` on the GRETA network).

Interrupt types are as follows:

- TOO
- SCS-107 without Loss of Attitude Reference
- SCS-107 with Loss of Attitude Reference
- General Replan

Except in the case of an SCS-107 without Loss of Attitude Reference interrupt, the user is prompted for the interrupt time, and all history files are truncated at that time.

In the case of an SCS-107 without Loss of Attitude Reference interrupt, the user is prompted for two times: the interrupt time and the resumption time. Files are truncated as follows:

- Truncated at interrupt time: FIDSEL, GRATINGS, SIMTRANS, SIMFOCUS, RADMON, SCS84
- Truncated at resumption time: ATTITUDE, ECLIPSE_TIMERS, DITHER, SCS98, SPM

In all cases:

- History files are assumed to be ordered time-sequentially for truncation purposes.

- Records exactly matching the truncation time are **not** retained. Thus the file is truncated starting from the first record (sequentially, from the top of the file) found at or after the truncation time.

If a file is truncated, a comment is added to the file indicative of the time and type of interrupt at the appropriate place in the file. If the file was truncated at resumption time, an additional comment is added, indicative of the resumption time. If a file is not truncated, no comments are added to it.

Following truncation, in the case of **either** type of SCS-107 interrupt (e.g. with or without loss of attitude reference), the following history records are added:

- SIMTRANS history file – record with position -99616 at the interrupt time
- RADMON history file – record indicating DISA at the interrupt time
- FIDSEL history file – record indicating that all fid lights were reset at the interrupt time

IMPORTANT NOTE: As of the time of this writing, none of the files to which these "artificial" records are added are vehicle files. Thus each of these files is truncated at the interrupt time, regardless of the interrupt type, so records are always added to the end of the file. The script will raise a `CODE` error, and `FAIL`, if an attempt is made to add an artificial record anywhere but the end of the file (based on the interrupt time). This `CODE` error would indicate a malformed history file, a bug, or the need to update the software to handle insertion of artificial records at other points in the file.

Status is reported for each history file as well as for the overall history update, based on the successful execution of the above process. It is possible for the script to experience errors during the process; if an error is reported for a particular file, the user should manually inspect and adjust the file as necessary.

backstop_invert (rough)

IMPORTANT NOTE: This specification is known to be in a rough state. Several suboptimal elements of the script were not improved, given the heavy external reliance on the format of the backstop file (among other concerns). Changes are generally limited in scope given such concerns. This specification should be taken with a grain of salt.

History Items

NONE

This script, a major part of the backstop tool, is responsible for production of the backstop file from the timeline report (TLR).

The script begins by initializing the single-part command database, the algorithm for which is discussed at the end of this document.

Once the TLR line file is parsed, its records are individually processed, first with the `VCDU` field is taken for script-wide tracking; the initial tracking VCDU is **-1**. The record is processed in context. The command is determined to fall in one of the following groups:

- Command continued from a previous line (e.g. continuing to accumulate hex for a multi-part command)
- Orbit point record (printed according to **Command Printing**, with eventdata string comprised of the `ORBPT` field as `TYPE`, and event type `ORBPOINT`)**. Special note:** though the timestamp for the record is taken from the TLR, the VCDU counter value for the backstop record is the most recent tracking VCDU counter value (based on TLR file iteration). As such, the VCDU counter value in the backstop record may be wrong, and may disagree with the timestamp for such backstop record.
- One of the following:
    - Special command
        - Update VCDU (if `MSID` is `COSATDLY`)
        - Multipart command (start)
        - SIM packet (if `MSID` is `3SIMHD`)
        - Skipped command
    - ACIS Packet Start (if `HEX` begins with '`D8`')
    - [fall-through] If `MSID` is non-empty, "regular command", expected to be a single-part command.

Command processing based on each of the above situations is discussed below. In general, the principal in processing commands involves extraction of data of interest (and accumulation of that data into an event data string specific to the command), followed by printing of the record, including a mixture of fields common to each command along with the event data string.

Update VCDU
VCDU is tracked based on commands encountered with `MSID COSATDLY`. In the TLR, such commands provide, both in the command hex and the `VCDU` field, the next VCDU value. The intent is that the values match. VCDUs are provided in the backstop file,
and as such, the current VCDU is tracked via `COSATDLY` in order to match up commands with VCDUs (initial script tracking value is -1, to be reset upon the first valid `COSATDLY` command, see below).

The information is provided twice in the COSATDLY because, while the maximum VCDU is `16777215` (which requires `24` bits), only `23` bits are used in the command hex to express the new VCDU. This limitation requires external tracking of the high bit (left-to-right, numbering the bits `23-00`, this would be bit `23`), as well as the ability to determine whether the high bit is set (which can be done by looking at the `VCDU` field).

For the purposes of the discussion, the "hex VCDU" will be the value extracted from the command hex (`HEX` field), and is comprised of 23 bits (however, it will be compared against 24-bit values, and as such, the leftmost/top bit should be regarded as 0). "command VCDU" will be the numeric (non-hex) value extracted from the `VCDU` field, which can take on the full range of VCDU values. If the command VCDU is comprised entirely of asterisks (this may be due to overflowing the field in the TLR), its value is regarded as 0 (and will be distinguished in this discussion from the command VCDU actually being 0).

The first `COSATDLY` command is used to determine the initial setting of the high bit. The high bit is a global setting checked and used with each `COSATDLY` command. If either of the following is satisfied, the initial high bit setting is 1 (otherwise it is 0):

- Command VCDU is comprised entirely of asterisks
- (Hex VCDU < Command VCDU) **and** the two differ only by the high bit (e.g. the Command VCDU is 8388608 greater than the hex VCDU). **This should be equivalent to a test performed later; it would seem that all we need to do is test whether the two only differ by the high bit and the high bit is set in the command VCDU.**

At this point, regardless of whether this was the first command, the "new VCDU", which is the VCDU value intended by this command, is constructed by combining the hex VCDU with the high bit setting (with the latter used as bit `23` in the hex VCDU). **The command VCDU value will be 0 in the case that it is comprised entirely of asterisks, but this will not result in a VCDU mismatch, see below.**

Based on the new VCDU:

If command VCDU is not comprised of asterisks, and the new VCDU does not match the command VCDU, one of the following applies:

1. If the new VCDU and command VCDU differ only in the top bit being set in the command VCDU, the high bit setting is changed to 1, which causes them to match.
2. Otherwise, if the new VCDU and command VCDU are equal in their lower 23 bits (22-00), the high bit setting is changed to 0, which causes them to match
3. Otherwise, there is a true mismatch between the command VCDU and the new VCDU, **and a warning is raised**.

Only in cases 1 and 2 is the tracking VCDU updated, and is set to the command VCDU.

- Otherwise, the new VCDU is equal to the command VCDU, the command VCDU was comprised of asterisks, or both. In this case, regardless, the tracking VCDU is updated, and is set to the new VCDU; the latter case will not raise an error.

In three above-listed cases where the tracking VCDU is updated, hooks are present for tracking of file-wide values of the minor cycle and number of minor frame commands (both to 0 at the time the tracking VCDU is updated), **but the subroutine which updates those values is never called in this script, and as such, both are always 0.**

Multipart Command Processing
A selection of multipart commands are processed by this script, are listed below (listed along with the number of parts/lines involved, not including the first part). Processing is specific to each command, and is discussed along with each; generally, the hex from each of the lines/parts of the command is accumulated prior to processing, and processed if the result is non-empty. On the first line of the command, set aside as applicable to the command are VCDU and **minor cycle – but the routine that updates it is not called (and should be removed), thus mc is always 0 in the backstop file**.

Prior to processing the accumulated command data (hex), the following attributes (common for output of multi-part commands) start the event data string that will eventually be printed as part of the backstop record : `MSID` (as `TLMSID`) and the number of parts of the command including the first part (as `CMDS`).

Note: in this discussion, for a series of `n` hex digits (words), the hex will be referenced as words `01` through `n`, left to right.

- `AOUPTARQ – 7`

  Event type: `MP_TARGQUAT`.

  With a total of 8 parts, the accumulated hex will be of length **`56`**. The result is a quaternion with four components (q_1–q_4). The hex breaks down as follows:

Words	Usage
01-14	skipped
15-26	converted to double-precision numbers, with value representing q_1, q_2, and q_3, respectively (scientific notation, with 8 digits after the decimal point)
27-38	
39-50	
51-56	skipped

(These values confirmed in multipart cmd DB)

q_4 is derived as follows: $q_4 = \sqrt{1 - q_1^2 - q_2^2 - q_3^2}$

The four quaternion components are appended to the event data string (comma-

delimited, with no comma following the fourth component, fields named Q**n**).

- AOSTRCAT – 48

  Event type: MP_STARCAT

  With a total of 49 parts, accumulated hex is of length **343**. However, the data of interest, which begins at word 15**,** is split into 16 parts of length 20, representing each of the 16 stars that the command accommodates. For this discussion, each part's hex digits will be referenced from 01-20, and the parts themselves will be identified in the range 1-16.

  For each part, extract the following fields from the 20-word hex; bits are referenced 00-79 sequentially from left to right.

Bits	**Field Name**	**Remarks**
00-02	Ignored	
03-05	IMNUM	Image Number
06-25	YCNT	Y position (CCD column)
26-45	ZCNT	Z position (CCD row)
46-55	MAXCNT	Maximum magnitude (counts)
56-65	MINCNT	Minimum magnitude (counts)
66-71	DIMDTS	Either: • Dimension • Designated Track Star
72	RESTRK	Either: • Resolution Flag • Track/Monitor
73-74	IMGSZ	Image Size
75-77	TYPE	Star Type
78-79	Ignored	

(These values confirmed in multipart cmd DB)

The following values are derived from the above

- YMAG= $(YCNT * (3.0300855 * 10^{-8})) - 1.58859113 * 10^{-2}$ (converts from column to radians?)

- ZMAG= $(ZCNT * (3.0300855 * 10^{-8})) - 1.58859113 * 10^{-2}$ (converts from row to radians?)
- MAXMAG= $(MAXCNT * (1.5625 * 10^{-2})) - 2$ (convert counts to magnitude, which is unitless)
- MINMAG= $(MINCNT * (1.5625 * 10^{-2})) - 2$ (convert counts to magnitude, which is unitless)

  Each mag has 10 bits (range 0 to (2^10)-1); conversion results in a value ranging from -2 to 13.98.

The following is added to the event data string for each of the 16 parts (the following is an example for part X). The presumption (based on the result in the backstop file) is that fields are already populated with appropriate default values in the case that there are fewer than 16 stars in the catalog, because all parts are processed identically.

Field	**Field from above; Rmks on Output Fmt**
IMNUMX	IMNUM
YANGX	YMAG; exp fmt, 8 digits after dec pt
ZANGX	ZMAG; exp fmt, 8 digits after dec pt
MAXMAGX	MAXMAG; exp fmt, 8 digits after dec pt
MINMAGX	MINMAG; exp fmt, 8 digits after dec pt
DIMDTSX	DIMDTS;
RESTRKX	RESTRK;
IMGSZX	IMGSZ;
TYPEX	TYPE;

The fields are comma-delimited, **with a bug resulting from the fact that a comma is incorrectly placed after the last field for the last star**.

- AODESMOS – 5

  Event type: MP_SYSMOM

  With a total of 6 parts, accumulated hex is of length **42**, which contains the three components of momentum. $TOTMOM_1$ is taken from words 15-22, $TOTMOM_2$ is taken from words 23-30, $TOTMOM_3$ is taken from words 31-38. Each is converted to a double precision number (with 8 digits after the decimal point), and added to the event data string with the field name TOTMOMX.

  (These values confirmed in multipart cmd DB)

- COAOSQID – 2

  Event type: MP_OBSID

  With a total of three parts, the accumulated hex is of length **21**; the result is the new OBSID, and

is represented by words 17-21; the rest of the hex is ignored for the purposes of this script. The OBSID (converted to decimal) is appended to the event data string as ID.

(These values confirmed in multipart cmd DB)

- EOECLETO – 2
- EOECLSTO – 2
- EOECLST2 – 2

Event type: MP_ECL_TIM

With a total of three parts for each, the accumulated hex is of length **21**. The data of interest, the number of VCDUs ("count") for the eclipse timer, is represented by words 17-21. The count (having been converted to decimal) is added to the event data string as TIMECNT, along with that count converted to seconds (e.g., multiplied by 0.25625), as TIMESEC.

(These values confirmed in multipart cmd DB, though some inconsistency in descriptions of fields)

- AODITPAR – 8

Event type: MP_DITHER

Total of 9 parts; accumulated hex length is **63**. It is broken down by field as follows (all are converted from hex to double):

Words	Field Name	Description
15-22	ANGP	Pitch (Z) phase angle (rad)
23-30	ANGY	Yaw (Y) phase angle (rad)
31-38	COEFP	Pitch (Z) amplitude (rad)
39-46	COEFY	Yaw (Y) amplitude (rad)
47-54	RATEP	Pitch (Z) frequency (rad/sec)
55-62	RATEY	Yaw (Y) frequency (rad/sec)

(These values confirmed in multipart cmd DB)

All are added to the event data string, printed in exponential format with 8 digits following the decimal point. **Suffers from the same superfluous comma problem as does AOSTRCAT, though for no good reason; these fields are not processed within a loop.**

- AOSETINT – 2

Event type: MP_ACAINT

Total of 3 parts; accumulated hex length is **21**. The result of interest is integration time in seconds. The value is stored in bits 63-75. Added to the event data string is this hex value, converted to decimal (as CMDFIELD) along with the hex value multiplied by 0.016 (believed to be inverse of 2 major frames) (as INT_TIME), which, according to the commentary provided, is the integration time in seconds.

(These values confirmed in multipart cmd DB, except 0.016)

- AVAMPOOF – 1

  Event Type: Command_HW

  Total of 2 parts. However, since this multi-part command is fully predefined, no fields are extracted from the record. The only event data field added to the record is MSID= AVAMPOOF

- AVBMPOOF – 1

  Event Type: Command_HW

  Total of 2 parts. However, since this multi-part command is fully predefined, no fields are extracted from the record. The only event data field added to the record is MSID= AVBMPOOF

- AOSAPARS – 6

  Total of 7 parts; accumulated hex length **49**. Broken down by field as follows:

Words	Field Name	Description
15-22	AOSSA1AS	
23-30	AOSSA2AS	
31-38	AOSSA1RR	
39-46	AOSSA2RS	

  All are added to the event data string, printed in exponential format with 8 digits following the decimal point.

  (Documented in the Multipart Command Database.)

- AOEPHUPS – 17

  Event type: MP_EPHEMERIS.

  With a total of 17 parts, the accumulated hex will be of length **119**. The result is a 15 parameters. The hex breaks down as follows:

Words	Usage
01-07	AOEPHEM1 – Command Constant
08-14	AOEPHEM2 – Command Constant
15-22	AORATIO – SQRT (Apogee to Perigee Ratio)
23-30	AOARGPER – Argument of Perigee
31-38	AOECCENT – Eccentricity (e)
39-46	AO1MINUS – (1/2) * (1 – cos(i))
47-54	AO1PLUS – (1/2) * (1 + cos(i))
55-62	AOMOTION – Mean Motion
63-66	AOITERAT – Number of Iterations
67-74	AOORBANG- Specific Orbital Momentum
75-86	AOPERIGE – Time of Perigee
87-94	AOASCEND – Right Ascension of the Ascending Node
95-102	AOSINI – sin of Inclination (i)
103-110	AOSLR – Semi-Latus Rectum
111-118	AOSQRTMU – SQRT (mu * a * e^2)

(These values confirmed in multipart cmd DB)

SIM Packet Processing

If a command is encountered with MSID 3SIMHD, we have a SIM packet. As with multipart command processing, SIM packet processing involves concatenation of hex from multiple commands. However, only a certain portion of each SIM packet command is useful.

For each 28-bit command (identified left-to-right, 12-39), the value represented by bits 19-34 (as four hex digits) is extracted and appended to the accumulating hex for this SIM packet. In the first command in the packet, the lower (rightmost) 8 bits of **this value** represent the number of commands in the SIM packet (in addition to this first command, e.g. this is the number of commands left to process in this SIM packet). Technically it is the lower 6 bits, but the top two of the lower 8 are always 0.

(Information not yet verified. CM07d Rev A not sufficient. Pending verification in latest rev).

Upon extracting the above listed information from all commands in the SIM packet, the accumulated hex is parsed as follows. Note: the hex digits, left to right, will be identified beginning with 0.

If hex digits 04-07 match the strings:

- **2033**
  - Event type is SIMFOCUS
  - Determine POS field
    - Take the lower (rightmost) 15 bits of hex characters 08-11
    - If the value is more than 16384, subtract 32768.
- **2011**
  - Event type is SIMTRANS
  - Determine POS field
    - Take the lower (rightmost) 15 bits of hex characters 08-11
    - If the value is more than 16384, subtract 32768.

- Multiply the value by 8

If either of the above was true, the record is printed according to **Command Printing**, with the event data string containing only the above-determined `POS` field. Otherwise, a string is simply printed, "`SIMPKT`", along with the accumulated hex. **The latter was not found in the backstop archive.**

After the SIM Packet is processed, the record is printed according to the discussion of **Command Printing**.

Skipped Commands
The following "commands" are not processed, based on `MSID`: `AON`, `AOFF`, '`COMMENT: 03`'. The mechanism for doing so is to call an empty routine in conjunction with them (which may not be the most graceful handling method). Note that all lines containing the string 'COMMENT' will be skipped anyway.

ACIS Packet Processing
As discussed above, ACIS packet processing begins with a TLR record with hex beginning with `D8 (Ken to verify)`. This is its first record, and its last three hex digits indicate the number of 16-bit ACIS commands in the ACIS packet. Since there are seven hex digits per command, the total number of commands in the ACIS packet (not including the header) is computed as follows:

```
ceil((4 * (# of 16-bit ACIS commands) / 7)
```

Note that the ceiling function is not used explicitly in the code, **but should be**; the calculation is unnecessarily complicated in the code but appears to be identical to `ceil`.

As with multi-part command processing, the hex from commands associated with the ACIS packet (the count of which was determined above) is accumulated. Once complete, if the hex is non-empty, the event data string is assembled as follows:

- `TLMSID` – `MSID` field from the TLR line
- `CMDS` – Number of commands including the header
- `WORDS` – Number of 16-bit ACIS commands as discussed above
- `PACKET(40)` – Accumulated hex for the ACIS packet

After the ACIS packet is processed, the record is printed according to the discussion of **Command Printing**.

Regular Command Processing
If a command does not fall into one of the above categories, it is processed as a "regular" command, which is expected to be a single-part command.

The event data string is assembled as follows, with additions as described below.

- `TLMSID` – the `MSID` field from the TLR record
- `HEX` – command hex (`HEX` field from the TLR record)

1. Next, the command hex is looked-up in the fully pre-defined hash of commands (given that command data is the key). If found, added to the event data string is the command mnemonic (as `MSID`). A modifiable command may be built (its modifiable fields set) such that it matches a fully-predefined command in the single-part command database. The latter command would likely have been added to the database for convenience. In this case, TLMSID and MSID will not match; the TLMSID will be the modifiable command; the MSID will be the fully-predefined command. In this case, option 2 below does not apply (because the command ends up being processed as a fully-predefined command).

2. Otherwise, the command hex may match the pattern of pre-defined fields in a modifiable command – and if such a match is found, the command mnemonic is appended to the event data string (as `MSID`), along with, for each modifiable field, the field name and its decimal value converted from command hex (determined by using the mask for each field and applying it to the value of the command hex), as "`<field>=<value>`".

3. If neither 1 nor 2 applies, the command is unknown and cannot be processed. A series of question marks ('????????') are appended to the event data string, along with raising a warning to this effect. This **may** be due to the occurrence of a multipart command in the schedule not known to this script.

If bit 12 (leftmost, most significant in referencing the bits `12-39`) of the command hex is set, event type is `COMMAND_SW`, otherwise, event type is `COMMAND_HW`. This is based on matching commands in the single-part command database.

After the command is processed, the record is printed according to the discussion of **Command Printing**.

Command Printing

Fields of a record are printed as follows, based on having accumulated an event data string (as appropriate and described above); fields, where used, come directly from the TLR record;

- `TIME` (string)
- VCDU (presumably the tracking value when the command was encountered) – width of 8 digits
- Minor Cycle – 1 digit (as discussed, always 0)
- Event type (16 characters)
- Event data string (as accumulated depending on command type)
- `SCS` (integer)
- `STEP` (integer)

Initialization of the Single-Part Command Database

Comprehensive information about single-part commands is provided in `MSIDCMDF.txt`, which is generated from the command database (CDB) (in particular, from the scripts that generate the flat file databases for GRETA).

The file is comprised of comma-delimited lines, each beginning with a command mnemonic, and strings enclosed in double-quotes (the double-quotes are stripped upon parsing). The fields of interest, numbered from 0, are as follows (omitted numbers correspond to fields not used here):

Number	Name	TLR field (col hdr)
0	CMD	Command mnemonic
1	FIELD	Field Name
2	CMDTYPE	**M**odifiable or **P**redefined Field
3	INTYPE	**B**inary or **H**ex
4	UPTYPE	Uplink Type: • IUNS (unsigned integer) • IDIS (binary integer discrete) • ICHK (check code) • IPAR (parity bit field) • ITWO (twos compl signed int)
8	LENGTH	Length of field in bits
10	WORD	Word at which the command/field data starts (1-7) (can have 8, but ignored by script - ex: CRC CODE)
11	BIT	Bit at which command/field data starts (0-3)
12	INITDATA	Initial Data
17	DESC	Description

Lines of this file are processed as follows based on the information in these fields. Commands are divided as follows:

Type	Discriminant	Remarks
Single-Field (always fully pre-defined)	CMDTYPE is **P**, LENGTH is **28**	Retain INITDATA and CMD (mnemonic).
Multiple-Field, fully-predefined	WORD < **8**, LENGTH < **28** indicates handling of a field	Pre-defined fields for the command cover all 28 bits
Modifiable (multiple-field only)		Pre-defined fields for the command do not cover all 28 bits; presumes that any bits not covered are part of a modifiable field or not used
OBC Var Cmd(?)	CMD is **OBCVARCMD**	Skip? (**COMMENTED OUT**)

The 28 bits of hex representing the command data will be referenced as a range from **12-39**, with 12 representing the left / most significant bit, and 39 the right / least significant bit.

b	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
w	1				2				3				4				5				6				7			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

The `word` and `bit` fields correspond to the 'w' and 'b' rows above, respectively (starting word and bit for a field).

Bit 12 is later used to differentiate for the labeling of software and hardware commands in the backstop file. Per command documentation, Bit 12 is always 0 for hardware commands, and is not used for software commands. The code modifies bit 12 as follows:

- **Single-Field Commands**: No change to bit 12
- **Multiple-Field Fully-Predefined Commands**: Upon processing of all fields, set bit 12 (consider not doing this)
- **Multiple-Field Modifiable Commands**: Upon processing of all pre-defined fields, if bits 12-15 **are all not set**, set bit 12. This is reflective of the use of bits **12-16** to represent an RCTU, if together they represent one of the following numbers: {3, 6, 9, 12, 15}, indicative of a hardware command. The binary representations of each of those numbers require that at least one of the most significant four bits (12-15 in this case) be set (technically, this test could test for at least one of the bits 13-15 being set, based on the binary representations of the aforementioned RCTU numbers). (do this for all commands)

IN PRACTICE:

- Single-field commands (1730) **: 1199 set, 531 unset**
- Multiple-field Fully-Predefined commands (227): 89 set, 138 unset
- Modifiable commands (154) – all have bits 12-15 non-set – so it will be set for all

Bit 12 is later used to provide event type (hardware or software command) for commands matched in the single-part command database. Bit 12 does not necessarily establish a particular command as a hardware or software command *per se*, but the hex in the TLR file must match the bit 12 setting (as well as all other pre-defined fields of a command) in order to have it recognized as the appropriate command.

The following data is extracted based on command types:

- **Single-Field Commands**: the `INITDATA` and `CMD` fields.
- **Multiple-Field Fully-Predefined Commands**: Simply store the command data accumulated from the `INITDATA` fields from the fields of this command, along with the command (`CMD` field).

  The above two types of commands are stored in a single array and converted to a hash at the end of processing, keyed by the command data `INITDATA` (which had better be unique, given that the spacecraft knows nothing about mnemonics)

- **Multiple-Field Modifiable Commands**: The command mnemonic (`CMD`), command data with pre-defined data set into it (accumulated `INITDATA` for the command's pre-defined fields), and a mask indicating locations of pre-defined fields within the command data, are stored in separate

arrays as well as combined to be sorted by the mask (implicitly assumed to be unique). The end result is a set of three arrays (mask, command name, and pre-defined command data) sorted by mask (with corresponding data at corresponding indices).

Modifiable field information is stored separately in an array which includes, individually for the field, the command (`CMD`), field name (`FIELD`), along with a shift amount and a mask which taken together provide the position of the modifiable field in the command data. **It appears to be the case that these fields have no data (e.g. value 0) throughout the process; in any case, even if it were nonzero, the data is not retained (only the location, field name, and command mnemonic)**.

KNOWN LIMITATION: If the last command in the database file is a multiple-field, it will not be processed properly. This is due to the loop structure in reading the file and should be repaired. This problem was documented in the code. At the time of this writing, the last command in the single-part command database file is not comprised of multiple fields.

Numeric Conversion

4-byte and 6-byte hex strings are converted to double based on the MIL-STD-1750A; subroutines for these tasks are provided.

Known Limitations – Idiosyncrasies left alone

- This (and most, if not all, of backstop) is dependent on 32-bit integers. Major changes would be expected on a switch to a 64-bit OS.
- The Track_New_Cmd routine, used for tracking of minor cycles, is not used. It will not be touched. Minor cycle, which appears in every backstop record, is 0.
- `AOSTRCAT` records have an extra comma incorrectly placed after the last field for the star 16
- `AODITPAR` records have an extra comma after the last event data field
- The last command in the single-part database file must **not** be a multiple-field command.
- Ken believed that the correct way to determine HW vs SW commands was based on checking bits 12-15; if all not set, SW command; otherwise, HW command. This particular check is only performed for multiple-field modifiable commands; shortcuts are taken for single- and multiple-field pre-defined commands.

Action Items

- Ken will be researching SIM packet behavior. It seems that the most recent version of CM-07d will have correct SIM packet behavior information
- Ken will verify that all ACIS packet headers begin with hex ‘D8’ (and nothing else does)
- Ken will be researching OBCVARCMD to make sure it should not go to backstop file (as it does not now)
- Dave to verify that counting of SIM commands is correctly discussed in the behavior document
- Dave to discuss specifics of results of “Bit 12” testing with Ken. “Bit 12” testing was re-done with a generally-expected result.
- Dave to provide Ken with the list of uplink types actually present in the single-part command database.
- Ken may have additional notes

backstop_iumodecheck

History Items

None

According to script commentary, verifies that each IU mode select command `CIMODESL` is followed within 2 seconds by the corresponding transmitter-on command. IU mode is determined from the eleventh most-significant bit of the command hex (where 0=A, 1=B, and these assignments would be reversed if IU B were made prime).

Error conditions are as follows:

1. Command to turn on the transmitter corresponding to the IU mode select command is not found within two seconds of the IU mode select command.
2. If `CIMODESL` is encountered but a transmitter is already on, the turned-on transmitter mode (A or B) does not match the mode indicated in the `CIMODESL` command.
3. A transmitter-on command (`CTXAON` or `CTXBON`) is not preceded by turning both transmitters off (with `CTXAOF` or `CTXBOF`). The script commentary used the phrase "immediately preceded", but immediate precession is not required by the algorithm. The algorithm only requires that, at the point the transmitter-on command is encountered following a `CIMODESL`, a transmitter-off command for both transmitters has been encountered (since that `CIMODESL`).
4. Consecutive CIMODESL commands are separated by less than 2 seconds

---

The above-specified behavior is accomplished with the following algorithm.

For each `CIMODESL` command, determine the IU mode indicated by that command as prescribed above. Iterate forward (from the `CIMODESL`) in the backstop file and act as follows based on the first of the following commands encountered:

- `CIMODESL` – stop; indicates that no transmitter-on command was found corresponding to the current `CIMODESL`.
- `CTX?ON` (`?` representing `A` or `B`) – stop and verify that since the current `CIMODESL`, transmitter-off commands have been encountered for each transmitter since the current `CIMODESL` (and if not, an error is raised but processing continues)

Different actions are taken depending on whether a transmitter-on command was found in the above processing:

- **If found**, check that the transmitter-on command is correct (matched the IU mode select command) **and** that the transmitter-on command is within 2 seconds of the IU mode select command. If either check fails, raise an error.
- **If not found**, verify that the IU mode select command matches the last transmitter-on command *that followed a* `CIMODESL` *command* . Note that this is necessarily the last transmitter-on command as enforced by the CSD. If this check fails, raise an error.

backstop_lonenpm

History Items

ATTITUDE

The commentary for this script indicates its purpose as follows: “Creates a list of candidate momentum dump times”. The script essentially tracks the PCAD mode (NPM or NMM) and related commands with respect to the maneuvers laid out in the backstop file, and raises errors in certain cases.

Initial assumptions:

- NMM-NPM autotransition status (enabled/disabled) **ENABLED**
- PCAD mode (NPM/NMM) **NPM**

PCAD mode is tracked as follows:

- Upon encountering the first command after the calculated end time of a maneuver (or segment), if autotransition is enabled, PCAD state is **NPM**
- If an AONPMODE command is encountered, mode is **NPM**
- If an AONMMODE command is encountered, PCAD state is **NMM**

With that in mind, the following checks are performed:

- An error is raised if a quaternion update is encountered within the calculated timespan of a maneuver.
- A warning is raised if AONMMODE is encountered and current PCAD mode is **NOT NPM**
- An error is raised if AONPMODE is encountered and current PCAD mode is **NOT NMM**
- An error is raised if AONPMODE is encountered during a maneuver
- An error is raised if AOMANUVR is encountered and the current PCAD mode is **NOT NMM**
- An error is raised if AOMANUVR is encountered during a maneuver (and the encountered AOMANUVR is ignored for the purposes of the script in order to preserve script flow)
- A warning is raised if AOMANUVR is encountered and autotransition is not enabled

If one or more of the above errors is raised, the test fails, otherwise the test passes. Note that a warning(s) will not cause the test to fail overall.

backstop_mccexport

History Items

ATTITUDE

This script simply produces a list of the maneuvers and quaternion updates in the backstop file. This report, however, is based on the schedule span (output from either OFLS MPS or MATLAB), rather than on the span of the backstop file. To this end, NULL maneuvers are added (simply by adding artificial records) at the beginning and the end based on the following conditions:

Beginning: If the first update-quaternion record (Event type `MP_TARGQUAT`) in the backstop file occurs **more than four minutes after** the schedule start time, two artificial records are added to this report.

The first is an artificial quaternion-update record with the following parameters:

- Quaternion: identical to the quaternion retrieved from the ATTITUDE history file, presumably the most recent quaternion prior to the start of the backstop file
- Record time: schedule start time minus 7 seconds

The second is a maneuver-start record with the schedule start time.

End: If the last maneuver-start record (MSID `AOMANUVR`) occurs **more than four minutes and seven seconds before** the schedule end time, two artificial records are added to this report.

The first is an artificial quaternion-update record with the following parameters:

- Quaternion: identical to the quaternion from the last quaternion-update record in the backstop file
- Time: schedule end time minus 4 minutes 7 seconds

The second is a maneuver-start record, for which the time is the schedule end time minus 4 minutes.

backstop_mechcheck

History Items

ATTITUDE
SIMFOCUS
GRATING
SIMTRANS

The backstop_mechcheck script is based on an iteration of the backstop file and is largely concerned with the occurrences of grating moves and sim translations with respect to maneuvers.

Initial assumptions:

- NMM-NPM Autotransition (autotrans) **ENABLED**
- PCAD Mode **NPM**
- SIM trans is not in progress
- Momentum monitoring **ENABLED**
- Momentum unloading **ENABLED**
- Grating motion **DISABLED** as of the time of the first record in the backstop file
- Grating move is **NOT IN PROGRESS**
    - **IN PROGRESS** when grating move command is encountered
    - **NOT IN PROGRESS** when 4MC5ADS is encountered
- Telemetry format **IS NOT 4**

Note: when tests below mention consideration of the last grating move or sim trans end time, such tests are not run if a grating move or sim trans has not yet been encountered in the backstop file. This follows from the assumption that the backstop file is assumed to start without either in progress.

Backstop file iteration proceeds at this point. Actions based on that iteration are as follows:

- Determine maneuver state. If maneuver completed prior to or concurrently with the current command
    - If autotrans is enabled
        - PCAD Mode is changed to **NPM** (and this change will be visible to specific-command processing as listed below)
        - If the maneuver ended before the end of a sim trans in progress, **VIOLATION – Switched to NPM prior to end of SIMTRANS (FAIL)**
        - If the manuever ended before the end of a grating move in progress, **VIOLATION – Switched to NPM prior to end of grating move (FAIL)**
- If the current record event type is SIMTRANS, take the POS event data field as the new/destination SIMTRANS position
    - Checks:
        - If momentum monitor is enabled, **VIOLATION (FAIL)**
        - If momentum unloading is enabled, **VIOLATION (FAIL)**

        - If the current simtrans position matches the new position, **VIOLATION – Redundant sim trans (FAIL)**
        - If PCAD mode is **not** NMM, **VIOLATION (FAIL)**
        - New simtrans position check:
            - If outside the inclusive range [-102792, 101856], **VIOLATION (FAIL)**
            - Otherwise, if outside the inclusive range [-99616, 92904], **WARNING**
    - Compute the duration of the sim trans as follows (where `Δpos` is the current/last-encountered sim position subtracted from the new sim position)

```
int(|Δpos| / 727) + 65
```

    - Compute the end time of the SIM translation as the current record time plus the computed duration
- If the current record event type is SIMFOCUS, take the POS event data field as the new sim focus position
    - Checks:
        - New simfocus position check:
            - If outside the inclusive range [-1061, -418], **VIOLATION (FAIL)**
        - If the current simfocus position matches the new position, **VIOLATION – Redundant sim focus move (FAIL)**

Other checks are based on MSID, for records with event type COMMAND_SW *or* COMMAND_HW:

- If the current MSID is representative of a grating move (4OHETGIN, 4OHETGRE, 4OLETGIN, 4OLETGRE)
    - 4OHETGIN – New grating is **HETG**
    - 4OLETGIN – New grating is **LETG**
    - 4OHETGRE / 4OLETGRE – New grating is **NONE**
    - Checks:
        - If momentum monitor is enabled, **VIOLATION (FAIL)**
        - If momentum unloading is enabled, **VIOLATION (FAIL)**
        - If telemetry format is not 4 or if the last-encountered change to format 4 was less than 32.8s (one major frame) prior to the current record time, **VIOLATION (FAIL)**
        - If the new grating matches the current grating, **VIOLATION – Redundant grating move (FAIL)**
        - If the new grating does NOT match the current grating, and neither is **NONE**, **VIOLATION – trying to insert one grating when the other is already inserted (FAIL)**
        - If the PCAD mode is **not** NMM, **VIOLATION (FAIL)**
    - Compute the grating move end time by adding 180s to the current record time
    - Grating move state is **IN PROGRESS**
- 4OOTGEN
    - If grating motion is **enabled**, **WARNING – Consecutive grating motion enable commands without a disable**

    - Otherwise grating motion is currently disabled; if the last disable command was less than 0.512 seconds prior to the current record time, **VIOLATION (FAIL)**
    - Grating motion is **ENABLED**
- 4OOTGDS
    - Grating motion is **DISABLED**
- CSELFMTX – Change to TLM FMT X
    - If X is 4, TLM FMT **IS 4** (and the last-encountered change to FMT 4 is set as the current record time)
    - Otherwise (change to a TLM FMT other than 4)
        - TLM FMT **IS NOT 4**
        - if grating move state is **IN PROGRESS**, **VIOLATION (FAIL)**
- 4MC5ADS
    - If TLM FMT **IS NOT 4**, **VIOLATION – Must be in FMT 4 at end of grating move**
    - Grating move state is changed - **NOT IN PROGRESS**
- AOMUNLEN or AOFUNCEN/AOPCADSE 21 – **Momentum Unload Enable**
    - Momentum unloading is **Enabled**
    - Checks:
        - If momentum monitoring is not enabled, **ERROR (FAIL)**
        - If the current record time is less than either of the following, **ERROR (FAIL)**
            - Last simtrans end time + pad (180s)
            - Last grating move end time + settle time (180s)
- AOMUNLDS or AOFUNCDS/AOPCADSD 21 – **Momentum Unload Disable**
    - Momentum unloading is **Disabled**
- AOFUNCEN / AOPCADSE 32
    - Momentum monitoring is **Enabled**
    - If the current record time is less than either or both of the following, **ERROR (FAIL)**
        - Last simtrans end time + pad (180s)
        - Last grating move end time + settle time (180s)
- AOFUNCDS / AOPCADSD 32
    - Momentum monitoring is **Disabled**
    - If momentum unloading is not disabled, **ERROR (FAIL)**
- AONMMODE
    - PCAD mode is **NMM**
- AONPMODE
    - PCAD mode is **NPM**
    - If record time is less than either or both of the following, **VIOLATION (FAIL)**
        - Last sim trans end time
        - Last grating move end time
- AONM2NPE
    - Autotrans is **enabled**
- AONM2NPD
    - Autotrans is **disabled**
- AOMANUVR
    - Maneuver based on the most recent two quaternions is assumed to start

After completion of the iteration of the backstop file, check whether a grating move and/or SIMTRANS is in progress at the end of the backstop file. This check is based on the grating move end time plus settling time, and the SIMTRANS end time plus pad, respectively. If either or both are in progress at the end of the backstop file, **WARNING**.

Furthermore, if at the end of the backstop file,

- PCAD mode is NMM, **WARNING**
- TLM FMT is 4, **FAIL**

The result of the script will be reported as the high watermark of the severity of the of the collected results. The results, in order of severity: FAIL, WARNING, PASS.

backstop_mkdist

Creates the load review products distribution file, providing the contents of the load directory along with the history against which it was run (e.g. history directory not updated with the backstop file in the load directory). **Consider removing this from the backstop directory.**

backstop_msscheck

History Items
ATTITUDE

This function checks for the commanding associated with disabling/enabling the OFP Multiple Stars Suspected filter and SCS 97. The output is included in an output file called TEST_msscheck.txt.

If there are no commands to disable/enable the OFP Multiple Stars Suspected filter and SCS 97, the output file only reports that there are no such commands.

All times below are relative to t = 0.000 being (1) the end of a given maneuver (relative to the AOMANUVR start time) for which auto-transition from Normal Maneuver mode (NMM) to Normal Pointing mode (NPM) is enabled (labeled "NMM2NPM enabled" below); or (2) a command to go to Normal Pointing mode (all times are reduced by 0.75 seconds).

The four commands which are grouped and then checked are:

(1) 'Disable SCS 97' at t = -360.000 ± 0.258 sec.
(2) 'Disable Multiple Stars Suspected filter' (AOMSSDS) at t = -10.000 ±0.258 sec.
(3) 'Enable SCS 97' at t = 5.125 ± 0.258 sec.
(4) 'Enable Multiple Stars Suspected filter' (AOMSSEN) at t = 485.125 ±0.258 sec.

The following cases are identified (along the results of the check) for each group of these four commands.

Case 1: PASS
All 4 commands are present for a maneuver with NMM2NPM enabled.

Case 2: PASS
Commands (1), (2) & (3) are present for a maneuver with NMM2NPM enabled, but command (4) shows up during the next maneuver (but prior to the (2) command for that maneuver) because the NMM command for the next maneuver occurs earlier than when command (4) would occur.

Case 3: WARN
In a sequence of 2 or more consecutive maneuvers, all with NMM2NPM enabled, for the first maneuver, commands (1) & (2) are present and commands (3) & (4) are not present. This maneuver is followed by

zero or more maneuvers with none of the commands present. For the final maneuver in the sequence, commands (1) & (2) are not present, but commands (3) & (4) are present, unless the final maneuver is also the final maneuver in the load, in which case commands (3) & (4) may not be present. All maneuvers in the sequence are labeled WARN.

Case 4: WARN
For a maneuver with NMM2NPM disabled and no explicit NPM command, none of the commands (1), (2), (3) or (4) are present.

Case 5: WARN
If there is an explicit NPM command, all 4 of the commands (1), (2), (3), (4) are present. All 4 commands occur 0.750 sec earlier than listed above.

Case 6: WARN
If there is an explicit NPM command, commands (1), (2) & (3) are present, but command (4) shows up during the next maneuver (but prior to the (2) command for that maneuver) because the NMM command for the next maneuver occurs earlier than when command (4) would occur. All 4 commands occur 0.750 sec earlier than listed above.

For all other cases, the result of check is FAIL.

backstop_nmmdurationcheck

History Items
ATTITUDE

Generally: tests to make sure spacecraft does not remain on gyros for longer than the acceptable duration. Operation of this check is based on iteration of the commands in the backstop file.

The start of a range of time on gyros is determined by the `AONMMODE` command, and is not reset by a subsequent `AONMMODE` command until the end of the time range is determined in one of the following ways:

- `AONPMODE` encountered – time range ends at the `AONPMODE`
- `AOMANUVR` encountered **and** NMM-NPM autotransition is enabled at the end of such maneuver – time range ends at the calculated end time for the maneuver. Essentially the "automated" version of the first case.

Any such time range in excess of 3600 seconds raises FAIL.

A gyro hold occurs within such a time range when the end of a maneuver is reached with NMM-NPM autotransition disabled. `WARN` is raised if a gyro hold occurs, with a message indicating that at most three consecutive ranges of time on gyros are permitted that contain gyro holds.

NOTE: It was agreed that the load reviewer, not backstop, would be responsible for detecting occurrences of three consecutive ranges of time on gyros contain gyro holds.

Additionally, `FAIL` is raised if a time range on gyros containing a gyro hold is not immediately followed by a period of 1185 seconds or more in NPM.

WARN is raised if the backstop file ends less than 1185s after the end of such a time range. This is based on the greater of the durations of the two above-described NPM periods.

WARN is raised if the command AONPMODE occurs during the NPM period. WARN is raised if the command AONMMODE occurs during the NMM period. FAIL is raised if the time of the last record in the backstop file is during the NMM period. FAIL is raised and the check ends if the AONPMODE occurs during a maneuver.

After analysis of a given range, the start and end times are reset and the next range is determined as above. A failure is raised if analysis of a range is attempted when the start and/or end times could not be determined. Such failures may occur in the event of a lone NPM command, or if the file begins with an AOMANUVR command without a `AONMMODE` command having been encountered.

Note that NMM-NPM autotransition is assumed disabled at the start of the schedule – more conservative approach, and that the initial quaternion is taken from the history file.

backstop_npmdurationcheck

History Items

ATTITUDE

Generally: tests to make sure spacecraft remains in normal pointing mode (NPM) for longer than the acceptable duration after a maneuver. Operation of this check is based on iteration of the commands in the backstop file.

The start of time range in NPM is determined in one of the following ways:

- `AONPMODE` encountered – time range starts at the `AONPMODE`
- `AOMANUVR` encountered **and** NMM-NPM autotransition is enabled at the end of such maneuver – time range starts at the calculated end time for the maneuver. Essentially the "automated" version of the first case.

The end of a range of time in NPM is determined by the `AONMMODE` command

Any such time range less than 280 seconds raises FAIL. If the angle of the maneuver preceding the time range is greater than 0.9 degrees, any such time range less than 1200 seconds raises FAIL.

WARN is raised if the time of the last record in the backstop file is within 280 seconds of the start of the time range or within 1200 seconds of the start of the time range is the preceding maneuver is greater than 0.9 degrees.

The Start Time cannot be determined if the following conditions are true:

- AONPMODE is not encountered
- AOMANUVR encountered **and** NMM-NPM autotransition is disabled at the end of such maneuver

The condition of an undetermined Start Time is noted with a PASS if the angle of the maneuver is less than or equal to 0.9 degrees. The condition of an undetermined Start Time is noted with a FAIL if the angle of the maneuver is greater than 0.9 degrees

backstop_rptstate

History Items

ATTITUDE
GRATING
SIMTRANS
SIMFOCUS
RADMON
DITHER
SCS84
SCS98
SPM

The backstop_rptstate script parses the backstop file, printing out a variety of commands as they are encountered, as well as a periodic status printout. The report is generated in two flavors, by-attitude and by-comm. The former reports status based on maneuvers, while the latter reports status based on transmitter on/off command. The type of report produced by the script is based on arguments to the script. The backstop file to be processed is also provided as an argument.

Throughout the script, the radzone status is derived from the `RADMON` state (RADMON ENAB → NOTIN; RADMON DISA → IN). Radmon start/end time is also derived based on the time of the state change. Additionally, the science instrument in play is determined based on the `SIMTRANS` pos:

- `pos < -85000` → `HRCS`
- `-85000 < pos < 0` → `HRCI`
- `0 <= pos < 82000` → `ACISS`
- `otherwise` → `ACISI`

If history lookup of any of the above elements fails, the script exits `FAIL`.

Initial States not Taken From History Files

PCAD Mode – NPM
Autotransition – Enabled
IU Mode - Unknown
PA Mode: Unknown
OBSID: Unknown
All FID Lights: Unknown
Transmitter (A/B): Unknown
Transmitter A Status: Unknown
Transmitter B Status: Unknown
Amp A Status: Unknown
Amp B Status: Unknown
Telemetry Format: Unknown

Backstop File Processing (both reports) along with cues for report-specific status printouts:

If maneuver was completed prior the time of the current record, **take into account the transition to NPM** if autotransition is enabled.

- **Status Printout Cue**: Maneuver end status printout if report type is by-attitude or currently in-comm (see below).
- **Otherwise:** Print a conventional `MANEUVER END` record with the maneuver end time and an indication of the PCAD mode following the end of the maneuver (e.g. `MANEUVER END -> <PCAD MODE>`).

The following records result in output to both reports:

1. Event Type is `SIMTRANS`
   - Obtain `POS` event data field; compute duration of SIM move as follows:

     `int(|Δpos| / 727) + 65`

     Where `Δpos` is the new SIM position (`POS` field) less the previous
   - **Output**

     `>> <rec time>  SIMTRANS from <init pos> to <new pos> Dur <duration>`

2. Event Type is `SIMFOCUS`
   - Obtain `POS` event data field
   - **Output**

     `>> <rec time>  SIMFOCUS from <init pos> to <new pos>`

3. Event Type is `MP_OBSID`
   - Obtain `ID` event data field
   - **Output**

     `>> <rec time>  UPDATE OBSID <ID>`

4. Event Type is `MP_STARCAT`
   - Save off the star catalog information (event data)
   - **Output**

     `>> <rec time>  UPDATE STAR CATALOG`

5. Event Type is `MP_TARGQUAT`
   - Obtain quaternion components from event data fields `Q1,Q2,Q3,Q4`
   - Perform various attitude computations based on the quaternion
   - **Output**

     ```
     >> <rec time>  UPDATE TARGET <Q1> <Q2> <Q3> <Q4>
     >>                   RADEC: <RA> <DEC>
     >>                   ALPHA <alpha>  BETA <beta>
     ```

Until otherwise noted, Event Type is `COMMAND_SW` *for the following*. The following commands are selected based on the `MSID` event data field.

6. `4OLETGIN`, `4OLETGRE`, `4OHETGIN`, `4OHETGRE`
   - New grating setting is as follows, as of the time of the record
     - `4OLETGIN` → `LETG`

    - 4OHETGIN → HETG
    - 4OLETGRE, 4OHETGRE → NONE
  - **Output**
    ```
    >> <rec time>  GRATINGS <curr set> to <new set>
    ```
  - **Additionally** if 4OLETGIN/4OLETGRE, append to this record the following indication that 4LLORBX is out of state:
    [4LLORBX OUT -> <time of current record + 180s>]

7. AONMMODE
   - **Output**
     ```
     >> <rec time>  NORMAL MANEUVER MODE
     ```

8. AONPMODE
   - **Output**
     ```
     >> <rec time>  NORMAL POINTING MODE
     ```

9. AONM2NPE
   - Autotransition is **ENABLED**
   - **Output**
     ```
     >> <rec time>  AUTOTRANSITION ENABLED
     ```

10. AONM2NPD
    - Autotransition is **DISABLED**
    - **Output**
      ```
      >> <rec time>  AUTOTRANSITION DISABLED
      ```

11. AOMANUVR
    - Set the most recent two quaternions and the record time into the ManeuverTracker
    - **Output (by-comm report only)**
      ```
      >> <rec time>  MANEUVER START
      ```

12. OORMPDS
    - Radmon state is **DISA as of the time of this record**
    - **Output (two records)**
      ```
      >> <rec time>  RADMON DISA
      >> <rec time>  RADIATION (BASED ON RADMON) ENTRANCE
      ```

13. OORMPEN
    - Radmon state is **ENAB as of the time of this record**
    - **Output (two records)**
      ```
      >> <rec time>  RADMON ENAB
      >> <rec time>  RADIATION (BASED ON RADMON) EXIT
      ```

14. AODSDITH
    - Dither state is **DISA** as of the time of this record
    - **Output**
      ```
      >> <rec time>  DITHER DSDITH
      ```

15. AOENDITH

- Dither state is **ENAB** as of the time of this record
- **Output**

```
>> <rec time>  DITHER ENDITH
```

16. CODISASX with event data field CODISAS1 = 84/90/91/92/97/98
    - SCS XX state is **DISA** as of the time of this record
    - **Output**

```
>> <rec time>  SCSXX  DISA
```

17. COENASX with event data field COENAS1 = 84/90/91/92/97/98
    - SCS XX state is **ENAB** as of the time of this record
    - **Output**

```
>> <rec time>  SCSXX  ENAB
```

Note: only SCS84 and SCS98 are tracked for status printouts. Occurrences of the rest are merely reported.

18. AOMUNLGR
    - **Output**

```
>> <rec time>  MOMENTUM UNLOAD START
```

19. AOFUNCDS with event data field AOPCADSD = 30
    - SPM state is **DISA**
    - **Output**

```
>> <rec time>  SPM  DISA
```

20. AOFUNCEN with event data field AOPCADSE = 30
    - SPM state is **ENAB**
    - **Output**

```
>> <rec time>  SPM  ENAB
```

21. OFMTSEPS
    - **Output**

```
>> <rec time>  COMMAND TO EPS SUB-FORMAT
```

22. OFMTSNRM
    - **Output**

```
>> <rec time>  COMMAND TO NORMAL BACKGROUND SUB-FORMAT
```

23. OFMTSPDG
    - **Output**

```
>> <rec time>  COMMAND TO PCAD DIAGNOSTIC SUB-FORMAT
```

24. OFMTSSSR
    - **Output**

```
>> <rec time>  COMMAND TO SSR SUB-FORMAT
```

Until otherwise noted, Event Type is COMMAND_HW *for the following*. The following commands are selected based on the TLMSID event data field unless otherwise noted.

25. CPAAON or CPABON
    - PA Mode is **HIGH**

26. CTXAON or CTXBON
    - Current transmitter is **A** or **B**, respectively
    - Transmitter on time is the time of this record
    - **In-comm**
    - **Status Printout Cue** Transmitter On (by-comm report only)

27. CTXAOF
    - Transmitter A status is **OFF**

28. CTXBOF and Transmitter A status is **OFF**
    - Transmitter B status is **OFF**

29. CPAAOF and Transmitter A status is **OFF** and Transmitter B status is **OFF**
    - Amp A status is **OFF**

30. CPABOF and Transmitter A status is **OFF** and Transmitter B status is **OFF** and Amp A status is **OFF**

    - Amp B status is **OFF**
    - Cue for **Transmitter Off** box (by-comm report only) with the time of this record
    - Reset all of the following to **UNKNOWN**
      - Transmitter A Status
      - Transmitter B Status
      - Amp A Status
      - Amp B Status
    - **NOT In-Comm**

31. AOFUNCDS with event data field AOPCADSD = 21
    - **Output**

```
>> <rec time>  MOMENTUM UNLOAD TIMEOUT
```

32. TLMSID event data field is CIMODESL
    - IU Mode is all but the first three characters of the MSID Event Data field (e.g. 512X)

33. TLMSID event data field is AFIDP
    - If MSID is AFLCRSET, set all 14 FID lights to OFF
    - Otherwise (**PRESUMABLY OF THE FORM AFLC06D3**): fid light number indicated by characters 5-6 (representative of a number) of the MSID is set to ON

Note: it looks as though an attempt was being made to assemble a string, from the above example: "AFL06D3I" – but the string is not used. (Pattern: begin with "AFL", append the fifth through eighth characters of the MSID, and append an "I".) Eric Martin asked that this construct remain.

34. `TLMSID` matches `CPA?LPM` (`?` represents a single character)
   - PA Mode is **`LOW`**

35. `TLMSID` matches `CSELFMTX` (`X` represents a single character)
   - Telemetry format is **X**
   - **Output:**
     `>> <rec time>  TELEMETRY FORMAT X`

36. `TLMSID` matches `CTUDMON`
   - Dwell Mode is Enabled
   - **Output:**
     `>> <rec time>  DWELL MODE START`

37. `TLMSID` matches `CTUDMOFF`
   - Dwell Mode is Disabled
   - **Output:**
     `>> <rec time>  DWELL MODE STOP`

Until otherwise noted, Event Type is ORBPOINT for the following. The following commands are selected based on the `TYPE` event data field unless otherwise noted.

38. `<any for which TYPE event data field does not contain the string 'DSS'>`
   - **Output**
     `>> <rec time>  <TYPE event data field>`

39. `PEXIT or LSPEXIT` (when preceded by a software command EOESTECN two minutes before the corresponding PENTRY or LSPENTRY) +11 minutes
   - SPM state is **`ENAB`**
   - **Output**
     `>> <rec time>  SPM  ENAB <SCS 33>`

40. `AOSPXDS`
   - AOACISPX state is **`DISA`**
   - **Output**
     `>> <rec time>  SPX FILTER DISA`

41. `AOSPXEN`
   - AOACISPX state is **`ENAB`**
   - **Output**
     `>> <rec time>  SPX FILTER ENAB`

42. `AODPXDS`
   - AOACIDPX state is **`DISA`**

- **Output**
  `>> <rec time>  DPX FILTER DISA`

43. AODPXEN
    - AOACIDPX state is **ENAB**
    - **Output**
      `>> <rec time>  DPX FILTER ENAB`

44. AOMSSDS
    - AOACIMSS state is **DISA**
    - **Output**
      `>> <rec time>  MSS FILTER DISA`

45. AOMSSEN
    - AOACIMSS state is **ENAB**
    - **Output**
      `>> <rec time>  MSS FILTER ENAB`

46. AOIRSDS
    - AOACIIRS state is **DISA**
    - **Output**
      `>> <rec time>  IRS FILTER DISA`

47. AOIRSEN
    - AOACIIRS state is **ENAB**
    - **Output**
      `>> <rec time>  IRS FILTER ENAB`

Completion of backstop file iteration
Status printout cue (by-attitude report only): If at this point a maneuver is determined to be in progress (as of the timestamp of the last backstop record).

Status Printout Formats
Status printouts are driven by the PCAD mode (as tracked through backstop file iteration) when a printout cue is encountered, as well as the type of cue.

PCAD Mode
Regardless of the PCAD mode, for cues other than **Transmitter Off**, the status listing is printed as such:

```
Attitude: Q <Q1>  RA  <RA>
            <Q2>  DEC <DEC>
            <Q3>
            <Q4>
Sun (est): Alpha <alpha>    Beta <beta>
Sim Trans: <SIMTRANS Position> (<Instr>)
    Focus: <SIMFOCUS Position>
  Grating: <GRATING setting: LETG, HETG, or NONE>
   RADMON: <RADMON State: ENAB or DISA> @ <Time this state came into effect>
  Radzone: <Rad Zone state: IN or NOT_IN> (based on RADMON)
```

```
 Dither: <Dither State: DSDITH or ENDITH>
 SCS-84: <SCS 84 State: ENAB or DISA> AOACISPX: <SPX Filter State: ENAB or DISA>
 SCS-98: <SCS 98 State: ENAB or DISA> AOACIDPX: <DPX Filter State: ENAB or DISA>
    SPM: <SPM State: ENAB or DISA>    AOACIMSS: <MSS Filter State: ENAB or DISA>
TLM FMT: <TLM FMT or UNKNOWN>        AOACIIRS: <IRS Filter State: ENAB or DISA>
```

If PCAD mode is **not NMM**, star catalog and fid light diagram are also printed. The following is the mapping of fid light states in the diagram: (`ON -> O; OFF -> .; UNKNOWN -> ?`) `X` is used for this example in all cases.

```
Fidlights:
                 __
                |  | X 5                    11  X ___   X 12
   2  X ____    |  |       7  X      X 9       |   |
       |    |   |A |            /\             |   |
   3  X| AI |   | S| X 6       /H \            |H  |
       |____|   |  |           \ I/             | S |
   1  X         |__|       8  X \/   X 10       |___|
                    X 4                    13  X      X 14
      [======]  <if instr is ACISI>
                [======] <if instr is ACISS>
     <if instr is HRCI>       [======]
     <if instr is HRCS>                       [======]
```

```
Star Cat:
-------------------------------------------------------------------------------
 IDX  IM#  TYPE  IMGSZ    MAG  MAXMAG    YANG    ZANG  DIMDTS  RESTRK  HALFW
-------------------------------------------------------------------------------
[ 1]   0   FID    8x8   7.000   8.000   919.9  -946.9       1       1     25
[ 2]   1   FID    8x8   7.000   8.000 -1828.2   951.2       1       1     25
[ 3]   2   FID    8x8   7.000   8.000   385.9  1595.2       1       1     25
[ 4]   3   BOT    6x6   7.264   8.766   959.1  1504.7      20       1    120
[ 5]   4   BOT    6x6   8.371   9.875  -796.3 -2169.3      20       1    120
[ 6]   5   BOT    6x6   9.331  10.828  1235.6   675.5      20       1    120
[ 7]   6   BOT    6x6   9.050  10.547  -198.3    58.2      20       1    120
[ 8]   7   BOT    6x6   8.479   9.984 -2283.1  -875.5      13       1     85
[ 9]   0   ACQ    6x6   6.542   8.047  1338.9 -1138.8      20       1    120
[10]   1   ACQ    6x6   9.657  11.156   824.8 -2068.4      20       1    120
[11]   2   ACQ    6x6   9.778  11.281  1517.8 -1531.3      20       1    120
```

Cue Type
Three types of cues are discussed above, appearing under various conditions in the two types of reports. The following are the header boxes corresponding to each cue.

Maneuver End

```
*********************************************************************************
* Maneuver Start: <Start Time>                                                  *
* Maneuver End:   <End Time>                                                    *
*                                                                               *
* Obs ID: <OBS ID>                                                              *
*********************************************************************************
```

Transmitter On

```
*********************************************************************************
* Transmitter On <Datetime>                                                     *
* Transmitter: <A/B>                                                            *
* IU Mode: <IU Mode>                                                            *
* PA Mode: <PA Mode>                                                            *
*                                                                               *
* Obs ID: <ObsID>                                                               *
*********************************************************************************
```

Transmitter Off

```
*********************************************************************************
* Transmitter Off <Datetime>                                                    *
*********************************************************************************
```

Formatting of the report for printing: it is desirable for the report to appear neatly on a 2UP nenscript printout. Each column in a 2UP printout has 65 lines, and the Maneuver End and Transmitter On status printouts (beginning with the header box) must appear beginning with the top of a column. Blank lines are inserted in order to affect this.

backstop_spmcheck

History Items
ATTITUDE
SPM

This script verifies that Chandra's attitude is within the Fine Sun Sensor's field of view whenever the Sun Position Monitor (SPM) is enabled.

For each period that the SPM is enabled, the pitch angle must remain within 45 to 139 degrees, exclusive of the boundaries.

Relevant commands:

- SPM Enable:
  - AOFUNCEN command for which AOPCADSE=30
    - or -
  - 11 minutes after ORBPOINT of TYPE=PEXIT or LSPEXIT (assumes SCS 33 run 11 minutes after eclipse penumbra exit) when preceded by a SW_COMMAND of MSID=EOESTECN two minutes before the corresponding ORBPOINT of TYPE=PENTRY or LSPENTRY
- SPM Disable:
  - AOFUNCDS for which AOPCADSD=30

The pitch angle, defined as the angle between the spacecraft's x-axis and the sun vector, is computed using the spacecraft attitude, spacecraft ephemeris, and sun ephemeris. During a maneuver, the attitude follows the profile defined in DM05, "Chandra On-Board Computer Flight Software Requirements Specification" in Section 3.2.3.3.5.1, "Attitude Command Generation".

The check will report a result for each period that the SPM is enabled. If the last SPM enable command is not followed by an SPM disable command, the final period will end at the end of the schedule. During each period, the result will be FAIL if the maximum pitch angle is greater than or equal to 140 degrees or the minimum pitch angle is less than or equal to 44 degrees. If the result is not FAIL during any period, it will be WARN if the maximum pitch angle is greater than 138 degrees or the minimum pitch angle is less than or equal to 46 degrees. If the result is not FAIL or WARN during a report period, the result will be PASS. The overall all result will be the least result (where FAIL is less than WAR is less than PASS) of all periods when the SPM is enabled.

backstop_synccheck

History Items

None

Note: This script is only intended to be run against split loads, though it only relies upon the combined backstop file.

This script takes as input the combined backstop file, and verifies that there are no overlapping SCS pair time spans within the schedule. The time span of an SCS pair is from the earliest command falling in either SCS, to the latest command falling in either SCS. In the context of split loads (vehicle/observing), the SCS pairs are 128/131, 129/132, and 130/133.

The backstop file is ingested and records are processed in sequence. The key to record processing is tracking of expected SCS number(s). Records which match the current tracking expected SCS number(s) are acceptable; all other records (with a nonzero SCS number) raise `WARN`. Records with SCS number 0 are simply skipped.

Tracking of expected SCS numbers is conducted as follows:

- First record establishes two expected SCS numbers (the record's SCS number and its counterpart).
- `COSCSEND` for one of the expected SCS numbers eliminates it, leaving one expected SCS number.
- `COSCSEND` for the remaining SCS number eliminates it, leaving no expected SCS numbers.
- The next record restarts the process (by establishing two expected SCS numbers as was done with the first record).

It is expected that at the end of the backstop file, there will be zero tracking SCS numbers because all expected `COSCSEND` commands will have been encountered. `WARN` is raised if this is not the case.

`FAIL` is raised if a record is encountered without an SCS number.

backstop_timecheck

History Items

TIMEREF

The purpose of the backstop_timecheck script is to check the correlation of VCDU and timestamp for each record in the backstop file.

The TIMEREF history file is used to establish the time reference for the correlation checks. This file contains a series of records with timestamp and VCDU; as with most other history lookups, we take the most recent record prior to the time of the first record in the backstop file. It is assumed that this file is kept up-to-date and each line consists of a timestamp and a VCDU separated by a single pipe ('|').

- For each backstop record, take:
  - $\texttt{time}_{\texttt{rec}}$: record time in seconds since the epoch
  - $\texttt{VCDU}_{\texttt{rec}}$: VCDU from the record
- From the record taken from the `TIMEREF` history file:
  - $\texttt{time}_{\texttt{ref}}$: reference time in seconds since the epoch
  - $\texttt{VCDU}_{\texttt{ref}}$: reference VCDU

For a given backstop record, the expected number of rollovers (between the backstop record and the selected record from the history file) is calculated as follows:

$$roll_{\text{exp}} = \frac{\frac{time_{rec} - time_{ref}}{0.25625} + VCDU_{ref}}{16777216}$$

If this number is 2 or more, `WARN` is raised indicating the need for an update to the `TIMEREF` history file. We incorporate rollovers into the VCDU differential as follows:

$$\Delta VCDU_{\text{exp}} = (VCDU_{rec} + (1677216 \times roll_{\text{exp}})) - VCDU_{ref}$$

The expected time of the backstop record, therefore, is

$$time_{ref} + (0.25625 \times \Delta VCDU_{\text{exp}})$$

Should the actual and expected record times differ by more than 5 seconds, `FAIL`.

Additionally, raise `FAIL` if the current actual record time is less than that of the previous record, or if the current expected record time is less than the previous expected record time.

The script reports the number of record times encountered, along with the maximum, average, and minimum differences between actual and expected record times.

backstop_unloadcheck

History Items
ATTITUDE

The following commands are tracked:
AOMUNLGR

An unload period is defined as the span of time 300 seconds prior to the AOMUNLGR and one hour and 110 seconds following the AOMUNLGR. Overlapping or adjacent unload periods are not permitted (if an overlap is detected, exit **FAIL).**

History Lookup: ATTITUDE.

Initial assumptions for backstop file iteration:

- PCAD Mode: NPM
- Autotransition: Enabled
- PCAD SF 21: Enabled
- PCAD SF 32: Enabled
- Telemetry Format: Not 6
- Dwell Mode: Off

At the start of each unload period, the following are checked:

1. PCAD Mode must be **NPM** at period start (or **WARN**)
2. PCAD SF 21 must be **ENAB** at period start (or **FAIL**)

   Relevant entries:
   Enable:
   a. AOFUNCEN command for which AOPCADSE=21
      -or-
   b. 11 minutes after ORBPOINT of TYPE=PEXIT or LSPEXIT (assumes SCS 33 run 11 minutes after eclipse penumbra exit) when preceded by a SW_COMMAND of MSID=EOESTECN two minutes before the corresponding ORBPOINT of TYPE=PENTRY or LSPENTRY

   Disable:
   a. AOFUNCDS for which AOPCADSD=21

3. PCAD SF 32 must be **ENAB** at period start (or **FAIL**)

   Relevant entries:
   Enable:
   a. AOFUNCEN command for which AOPCADSE=32

   Disable:
   b. AOFUNCDS for which AOPCADSD=32

4. Dwell Mode must be ENAB before AOMUNGLR command
    a. WARN if DISABLED
    b. FAIL if ENABLE occurs greater than 35 seconds or less than 33 seconds prior to AOMUNGLR command
    c. Relevant Entry: CTUDMON command

5. Telemetry Format must be 6 before AOMUNGLR command
    a. WARN if Telemetry Format is not 6
    b. FAIL if ENABLE occurs greater than 35 seconds or less than 33 seconds prior to AOMUNGLR command
    c. Relevant Entry: CSELFMT6 command

6. Dwell Mode must be DISA 10 seconds after Timeout
    a. FAIL if ENABLE occurs greater than 11 seconds or less than 9 seconds prior to timeout
    b. Relevant Entry: CTUDMOFF command
    c. Relevant Entry: Timeout command AOFUNCDS for which AOPCADSD=21

7. Telemetry Format must return to previous value 11 seconds after Timeout
    a. FAIL if telemetry command occurs greater than 12 seconds or less than 10 seconds prior to timeout
    b. WARN if telemetry does not return to previous value
    c. Relevant Entry: CSELFMT1, CSELFMT2, CSELFMT3, CSELFMT4, CSELFMT5 command

8. Dwell Mode must be DISA at the end of the Load
    a. WARN if Dwell Mode is enabled at the end of the load
    b. Relevant Entry: CTUDMOFF command

9. Telemetry Format must not be 6 at the end of the Load
    a. WARN if telemetry is 6 at the end of the load
    b. Relevant Entry: CSELFMT1, CSELFMT2, CSELFMT3, CSELFMT4, CSELFMT5 command

Notes about this check at the start of each period:

- If the period starts at the same time as a maneuver ends, and autotransition is enabled, the transition to NPM will be taken into account for the check.
- Any command or other event at the same time as the period starts will <u>not</u> be taken into account for this check.

None of the following may occur within 300 sec before – 700 sec after any AOMUNLGR, including the boundaries:

1. AONMMODE (or **WARN**)
2. AOMANUVR (or **WARN**)
3. AONPMODE (or **FAIL**)
4. Computed end time of maneuver reached (or **FAIL**)

5. `PCAD SF 32 Disable (AOFUNCDS /AOPCADSD 32)` (or **WARN**)

NOTE: It is known that the preceding two ranges overlap at 300 sec before AOMUNLGR, for maneuver end and AONPMODE. They are checked independently, and in the event either or both checks are not satisfied, a single message will be printed. Observe that the result is **FAIL** regardless of which check is not satisfied.

The following may not occur within 390-4020 seconds after any AOMUNLGR, including the boundaries:

1. PCAD SF 21 Disable `(AOFUNCDS /AOPCADSD 21)` (or **FAIL**)

The time of the last command in the backstop file must be strictly greater than 4920s after any AOMUNLGR (or **WARN**).

backstop_validate

The backstop_validate script was constructed to compare the commands contained in an OFLS timeline report (.tlr) file with those contained in a set of OFLS command load (.cld) files. Commands are compared sequentially based on hex, SCS number, and SCS step number. The timeline report is processed first, followed by the command load files in chronological order.

According to the commentary, the script is largely tailored to "OFLS R8" file formats. Presumably the file formats have not changed since then, at least changes that would affect the behavior of this script. Many of the specifics of CLD file processing are drawn from CM07D and IF1-60 ("Interface Control Documents AXAF to AXAF OCC").

In addition to the name of a TLR file, the `-s` option must be provided in order to indicate that the script is being run against split loads. Script operation (particularly CLD file processing) is dependent on the name of the TLR file, and whether the option is indicated.

In combined mode, all CLD file commands are considered.
In vehicle mode, only CLD file commands from vehicle SCS numbers (128-130) are considered.
In both modes, commands are sorted as described below.

The following table indicates the mode employed based on the pattern of the TLR file name provided and the presence or absence of the `-s` option:

File Name Pattern	-s option	Script operation
-**any**-	No	Vehicle Mode
CR*.tlr (combined TLR)	Yes	Combined Mode
-not- CR*.tlr (presumed vehicle TLR)	Yes	Vehicle Mode

Timeline Report File Processing

After parsing a TLR file, (see TLR parsing spec for more information on specific fields), records are processed individually. Records are skipped based on the following criteria:

- Skip any record where `ORBPT` is not empty (e.g. orbit-point records)
- Skip any record where `MSID` equals `AON` or `AOFF`
- Skip any record where `HEX` is empty

Any record not skipped is retained (in sequence as it was encountered) in part for comparison with records from the command load files. Values for the following fields are taken together as tuples: `HEX`, `SCS`, and `STEP` (line number in the file is carried along for reporting purposes). Exceptions:

- If the `SCS` field is 0 or the empty string (indicates undefined), the SCS number value taken for comparison purposes is the last value established from a defined (not zero and not empty string) `SCS` field, and if none has been established, this will be 0 (initial tracking value).

- If the `STEP` field is 0 or the empty string (indicates undefined), the step value taken for comparison purposes is the last record's value plus one. The "initial" last record's value is 0 (initial tracking value).

Command Load File Processing

Command load files are processed in order, according to the 'CLD File Sorting' algorithm.

Lines are skipped if they match any of the following criteria:

- Begin with `COBCSLDA`, **which is a command specific to OBC-A**;
- Begin with `CBTDNOOP`;
- Blank or consist only of whitespace

All other lines are processed in sequence, and each is split into two pieces: the first 20 hex characters (which are discarded) and the following 7 (which comprise the command hex). For the purposes of this discussion, the seven hex bytes will be treated from left-to-right, bits 27 through 00.

CLD files are comprised of a series of buffers, each of which is assumed to be comprised of exactly one block of commands. The following is the buffer/block structure, not including lines discussed as skipped (above):

- First line: **buffer header**. Number of commands in the buffer indicated by bits 09-00.
- Second line: **block header**. The block SCS number is indicated by bits 15-08; the block starting step number (within the SCS) is indicated by bits 27-16. All commands within the block are assumed to have the block SCS number.
- Commands considered for processing: the buffer header command count **less three** is the number of subsequent lines representing commands that will be processed. The three non-processed lines included in the buffer header command count are the buffer header, block header, and buffer checksum (checksum is the last line in the buffer and is itself skipped).

The following attributes are associated with each command taken for processing: command hex, SCS number (from the block header), and step number (initial value specified in the block header and incremented per command processed). File name and line number are carried along for reporting purposes.

Though the CLD files are opened and parsed in sequence, traversal of a CLD file from start to end will not necessarily result in a time-ordered set of commands in the case of split loads. With split loads, pairs of buffers presented in series within the file contain contemporaneous commands. It is necessary to sort these commands, by using the 23-bit counter value associated with the absolute time delay commands interspersed throughout the file.

The current tracking counter value is assigned to all commands until the next absolute time delay command is encountered, at which point the tracking value is updated and the process continues. The commands comprising ACIS packets, SIM packets, and multi-part commands must be excluded from the search for absolute time delay commands; parts of such commands can match the pattern expected for absolute time delay commands.

The set of commands is iterated in the order initially encountered, and the following algorithm provides for assignment of the counter value to each command.

- ACIS Packet – command hex begins with `D8`
  - All CLD commands comprising the packet are assigned the current tracking counter value, and are skipped for consideration as absolute time delay commands
  - Number of CLD commands occupied by the packet is computed as follows: bits 15-00 of the command represent the number of parts of the ACIS packet; number of CLD commands comprising the ACIS packet is `ceil[(4 * # parts) / 7]`
  - ACIS packets must be isolated first because they do not adhere to the multi-part software command header format and as such one or more of the CLD commands comprising the packet could be mistaken for an absolute time delay command.
- Otherwise – the current command is a single-part command or the start of a multi-part command (hardware or software).
  - If bits 27-23 of the command are `10010` (decimal 18), it is an absolute time delay command. Update the tracking counter value based on bits 22-00 of the command.
  - Otherwise, determine if this is the start of a multi-part **software** command (if bit 27 is `1` and bit 20 is `1`).
    - If so, we have a multi-part software command. All CLD commands comprising the command are assigned the current tracking counter value, and must be skipped for consideration as absolute time delay commands. The number of CLD commands occupied by the multipart command is based on the packing method and reported number of parts.
      - Packing method: bit 00 of the command; 1 indicates densely packed; 0 indicates sparsely packed
      - Number of parts is indicated by 07-00 of the subsequent command.
      - Number of CLD commands: if sparsely packed, simply the number of parts; if densely packed,
        `ceil[(4 * number of parts) / 7]`
    - Otherwise, we have a single-part software command, a single-part hardware command, or any part of a multi-part hardware command. Because no part of any hardware command can (unexpectedly) match the pattern for an absolute time delay command, and we have already eliminated the absolute time delay command (which itself is a single-part software command), all three possibilities here are handled by simply assigning the current tracking counter value to the command.

Having assigned the counter value to all salient commands in the CLD file, the commands are sorted by the following criteria in ascending order: counter value, SCS number, and original order of occurrence based on ordering of CLD files.

Comparison

As discussed above, groups of values are taken together in processing each file as follows:

- TLR file commands (hex, SCS, and step)
- CLD file commands (hex, SCS number, SCS step)

Before comparing the commands from these two files, the TLR file command sequence is changed if commands with the same VCDU have different SCS values and those values are not in ascending order. The commands with the same VCDU are then arranged so that the SCSs for these commands are in ascending order. If the command order is changed, the test result is initialized to WARN and a message is printed indicating which lines of the TLR file contain the commands that have a changed sequence.

These groups of values are compared one-by-one, field-by-field, with any differences reported along with file name and line number in each file (the TLR file and CLD file) at which the difference was found. Furthermore, if consecutive comparisons result in differences, only the first difference is reported, and a message is displayed at the end indicating that one or more error messages were suppressed. This is intended to prevent thousands of error messages from being printed in the case that the TLR is out-of-sync with the CLD files, for example.

Finally, the numbers of commands encountered in the TLR file and CLD files are reported; a difference in these numbers by itself is an error.

ev

The `ev` script establishes the following aliases for the `mission` account on the GRETA network (as it is sourced in that account's `.cshrc` file)

- `_backstop: untar_question; backstop.pl`
- `_history: backstop.pl`
- `_mkdist: backstop_mkdist.pl`
- `_edithist: backstop_interrupt.pl`

The intent of having `_history` run `backstop.pl` is for the user to execute the history update portion of that script only (and exit at the prompt following the history update).

Several other commented-out aliases are present in the file.

rename_files.pl

To be run after untarring a load; changes colons in filename to underscore for compatibility with Windows decompression tools (Winzip).

untar_question

Before beginning the untar, the script identifies where the products have come from based on the number of tarballs within the parent directory. If one tarball exists, the script assumes products came from ORViewer and CMAN (method = orviewer). If two tarballs exist, the script assumes products came from MEXEC and CMAN (method = legacy).

The script then identifies if the user created a sausage directory or a matlab directory. If either directory exists and the user has write permission, the files in the current directory ending in .gz are copied into that directory. If ORViewer products are present, all files not ending in .gz are then tarred and put into the sausage/matlab directory and given a name using the convention `MMMDDYYX_orviewer.tar.gz`.

Upon prompting the user as to whether the untar should proceed, the script runs `gunzip` against any files in the current directory ending in `.gz`.

The following directories are created: `fot, log, output, output/Thermal_Data, mps, & temp`. Under `mps`, the folders `er, or, soe, err, ode, ode/characteristics & ode/constraints` are created. If, after expanding the `Commands` tar (see below) a vehicle-only TLR file is detected (as identified by its name), a directory called `vehicle` is created, with subdirectories `log` and `output`.

The script expects at least one tarball to be present. One tarball is expected for ORViewer loads, and two tarballs are expected for MEXEC-created (legacy) loads. Any tarballs are matched with wildcards:

- `*Sched*tar*` - expected to match `MMMDDYYX_Schedule.tar` (legacy method only)
- `*Com*tar*` - expected to match `MMMDDYYX_Commands.tar` (legacy & ORViewer method)

For legacy loads, each tarball is used as input to `tar xvf` to extract files matching these specific patterns:

- `Schedule` tar: `*.dot, *.sum, *.or, *.er, *.soe, & *.err`
- `Commands` tar: `*.cld, *.tlr, *.fot, *.sum, *.trp, *.err, *CRM*, *CHARACTERIS*, & *CONSTRAINTS*`

After extracting all files matching the above patterns, the above files are now at the top level.
The files are then relocated to the previously-created directories based on file-matching patterns. This is detailed in the ‘Legacy Source Files’ column of the proceeding table.

For ORViewer loads, the tarball is used as input to `tar xvf` to extract files matching these specific patterns:

- `Commands` tar: `*.dot, *.cld, *.tlr, *.fot, *.sum, *.trp, *.err, *CRM*, *CHARACTERIS*, & *CONSTRAINTS*`

After extracting all files matching the above patterns, the above files are now at the top level.
The files are then relocated to the previously-created directories based on file-matching patterns. This is detailed in the ‘OR Viewer Source Files’ column of the proceeding table.

Source Files		Destination Folder
Legacy	OR Viewer	
m[0-3]*/md*.sum m[0-3]*/ms*.sum m[0-3]*/mm*.sum m[0-3]*/mg*.sum m[0-3]*/*.dot C[0-3]*/*.sum	./ms*.sum ./mm*.sum ./mg*.sum C[0-3]*/*.dot C[0-3]*/*.sum	mps
m[0-3]*/*.or	./*or	**mps/or**
m[0-3]*/*.er C[0-3]*/*CRM* Ephemeris/DO[0-9]*.orp	./*CTI_report.txt if sausage sausage/C*.er if matlab matlab/C*.er Ephemeris/DO[0-9]*	**mps/er**
m[0-3]*/*.soe	./*.soe	**mps/soe**
m[0-3]*/*.err C[0-3]*/*.err	C[0-3]*/*.err	**mps/err**
C[0-3]*/*.fot	C[0-3]*/*.fot	**fot**
C[0-3]*/*.cld C[0-3]*/*.tlr C[0-3]*/*.trp	C[0-3]*/*.cld C[0-3]*/*.tlr C[0-3]*/*.trp	**temp**
C[0-3]*/*CHARACTERIS*	C[0-3]*/*CHARACTERIS*	**mps/ode/characteristics**
C[0-3]*/*CONSTRAINTS*	C[0-3]*/*CONSTRAINTS*	**mps/ode/constraints**
if sausage sausage/*dynamical_offsets.txt sausage/*dynamical_offsets.log sausage/*ManErr.txt if matlab matlab/*dynamical_offsets.txt sausage/*dynamical_offsets.log matlab/*ManErr.txt	./*dynamical_offsets.txt ./*ManErr.txt ./*.p	**output**
	*.dot.ref *.log *.mat	**if sausage** **sausage** **if matlab** **matlab**
VR*.tlr	VR*.tlr	**vehicle**

The Orbit Events File name (DO*.orp or DO*) is found in the file ms*.sum.

Note that nothing is placed in the folders `log, vehicle/log, or vehicle/output` (or `output` in the Legacy case).

At this point, anything matching `*.tar` and files matching the pattern `*:*` are removed (and in the case that a directory matches either of those patterns, recursive deletion of that directory is undertaken). At that point, the contents of `temp` are moved to the top level of the current directory, and the `temp` directory is removed.

A separate script (rename_files) is then run such that, for each filename containing one or more colons, the file is renamed, with each colon replaced by an underscore. The current directory is traversed recursively in order to find all such files in the directory tree.