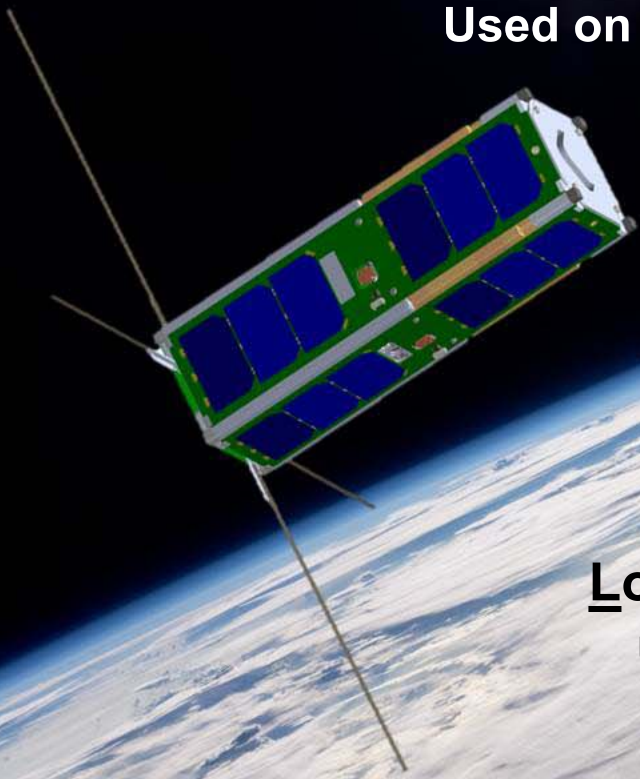
Overview of SSDL's

Telemetry Capture, Storage & Retrieval System (TECSTARS™)

Andrew E. Kalman <aek@stanford.edu>

Used on LMRST-Sat, a joint SSDL-JPL 3U CubeSat mission



LMRST-Sat's purpose: Flight-test the JPL <u>L</u>ow-<u>M</u>ass <u>R</u>adio <u>S</u>cience <u>T</u>ransponder (LMRST) using a CubeSat and the Deep Space Network

Host hardware: 12MHz 16-bit TI MSP430F2618 (116KB Flash, 8KB RAM)

Host software: Salvo RTOS + EFFS-THIN SD Card driver

Nonvolatile storage: 2GB SD Card, FAT16 format

Communications downlink: >=9600bps

Vizon Ground Station

The presence of TECSTARS on LMRST-Sat makes monitoring its health and status and downloading telemetry very simple:

LMRST - SAT
Low Mass Radio Science Transponder Satellite

LMRST Telemetry [-]

Receive Time:	2014-04-12 17:49:08 GMT
Sample Time:	2000-01-01 04:49:53 GMT
Latest Seq #:	2432
LMRST 1:	0.00 V
LMRST 2:	1.21 V
LMRST 3:	1.77 V
LMRST 4:	1.90 V
LMRST 5:	1.94 V
LMRST 6:	1.99 V
LMRST 7:	3.29 V
LMRST 8:	2.49 V

So, how does TECSTARS™ work?

GPS [-]

GPS Time:	22:06:35.00 UTC		
Latitude:	37.430288'	Longitude:	-123.739424'
Altitude:	92.3 m	Seq #:	52172

Configuration Telemetry [-]

Receive Time:	2014-03-25 21:33:49 GMT		
Sample Time:	2014-03-25 22:00:27 GMT		
Latest Seq #:	25821		
TAP ID:	Mode	Interval	Seq #
1-Beacon:	BOTH	5 sec	111987
2-Command Echo:	UART	Infinite	40
3-Bus Telemetry:	SD	30 sec	17091
4-LMRST Telemetry:	SD	20 sec	25755
5-Configuration Telemetry:	BOTH	20 sec	25822
6-I2C Read Telemetry:	SD	Infinite	0
7-EPS Voltage Telemetry:	SD	5 sec	111781
8-EPS Current Telemetry:	SD	5 sec	111770
9-EPS Temperature Telemetry:	SD	5 sec	111730
10-Command Buffer Telemetry:	SD	Infinite	0
11-ADC Read Telemetry:	UART	Infinite	1
12-DIO Read Telemetry:	UART	Infinite	1
13-GPS Telemetry:	BOTH	10 sec	52138

RTC Config:

RTC Delay (s):		RTC Calibration	
RTC Time:	-7fcacadf		

Cmds & Misc Config:

Blink Rate Freq:		HSS Status	255
Cmd Timeout Period:		Cmd Timeout Value:	

LMRST-Sat Telemetry

EPS + BATT:

Voltages, currents and solar panel temperatures
Provided by EPS & BATT modules I2C slaves (MSP430 is I2C Master)
= 22 (voltage), 22 (current) & 18 (temperature) bytes data

LMRST payload:

Eight health-and-status voltages
Provided by MSP320's on-chip multi-channel 12-bit ADC
= 16 bytes data (no packing)

GPS:

Time, elevation, latitude, longitude, #(sats) visible, etc.
Provided by on-board space-grade GPS receiver (NovAtel OEM615V)
= 18 bytes data

Beacon:

Selected status indicators (e.g., charge/discharge, busy/not busy, mode)
Command counters (e.g., received, executed, rejected & queued)
Selected physical parameters (e.g., battery voltages)
On-board time
File counts
= 29 bytes data

Telemetry Requirements

All telemetry must be uniquely identifiable. Therefore:

- **Related telemetry *datapoints* (or *data vectors*) may be grouped into telemetry *datasets***
- **Telemetry datasets are organized into *channels*, each with a *unique ID***
- **Within a channel, each telemetry dataset must:**
  - **have a unique *sequence number***
  - **be *timestamped***

Example:

- **All eight LMRST voltages are grouped together as one dataset, with one ID (4)**
- **The sequence numbers normally start at 0 and increment for every new dataset**
- **The current onboard time is recorded with each new dataset**

Every dataset will be organized in a standardized packet. The packet format is:

Header:	**ID, length, sequence number & timestamp**	**10 bytes**
Carton:	**dataset**	**< 230 bytes**
Footer:	**checksum**	**2 bytes**

Note that the dataset size may vary from one channel to another, based on the characteristics of the channel's telemetry.

Telemetry Application Packet (TAP) Structure

TLM Name	Generic Field Name	Unique Field Name	Value (dec)	Size (Byte)	TAP Offset (Byte)
LMRST_TLM	TAP Header	ID	4	1	0
LMRST_TLM	TAP Header	Tap Length	30	1	1
LMRST_TLM	TAP Header	Sequence Number	TLM	4	2
LMRST_TLM	TAP Header	TimeStamp	TLM	4	6
LMRST_TLM	Data Vector (Time)	Subseconds	TLM	2	10
LMRST_TLM	Data Vector (Data)	MSP430 ADC 1	TLM	2	12
LMRST_TLM	Data Vector (Data)	MSP430 ADC 2	TLM	2	14
LMRST_TLM	Data Vector (Data)	MSP430 ADC 3	TLM	2	16
LMRST_TLM	Data Vector (Data)	MSP430 ADC 4	TLM	2	18
LMRST_TLM	Data Vector (Data)	MSP430 ADC 5	TLM	2	20
LMRST_TLM	Data Vector (Data)	MSP430 ADC 6	TLM	2	22
LMRST_TLM	Data Vector (Data)	MSP430 ADC 7	TLM	2	24
LMRST_TLM	Data Vector (Data)	MSP430 ADC 8	TLM	2	26
LMRST_TLM	TAP Footer	Checksum	CHKSUM	2	28

LMRST-Sat's Telemetry Application Packet (TAP) structure for LMRST telemetry (TAP ID = 4)

The carton in a TAP contains only information associated with the dataset values and (optionally) additional time-related information that may augment the timestamp.

The TAP Packet's Role in TECSTARS

Each time the system *captures* new telemetry, it generates a *new and unique TAP packet* for the given TAP ID, using the channel's next consecutive sequence number and the current on-board timestamp.

When the system *stores* the new telemetry (by broadcasting it and/or saving to SD Card), *no further changes to the TAP are required*. If the TAP is broadcast, it's encapsulated into a Radio Application Packet (RAP). If the TAP is stored to SD Card, it's written to a file as-is, where *the TAP ID and sequence number specify the filename and position in the file*.

When the system is asked to *retrieve* the telemetry, the user *specifies the TAP ID and a range of sequence numbers*, and the system extracts those TAPs and broadcasts them within RAPs.

Flight Software: Structure & Components

Each TAP ID has a TAP structure associated with it:

9 bytes fully define this TAP's actions

```
typedef struct {
  uint8_t action;                       // action(e.g. save to SD card)
  uint8_t size;                         // size of the TAP in bytes (complete, w/header & footer)
  uint8_t interval_idx;                 // capture interval (lookup into table)
  uint8_t (*carton_fill_fp) (void);     // carton fill function (TAP-specific)
  uint32_t seq_num;                     // sequence number
} TAPStruct_t;
```

Each TAP ID has a dedicated carton-filling function:

```
uint8_t carton_LMRST_fill_TAP(void) {
  uint16_t lmr_data[DATA_VECTOR_SIZE_LMRST_TELEM];
  uint8_t i;


  read_LMRST_telem(lmr_data);
  // put in LMRST data -- locally it's little-endian, this converts to big-endian
  for(i=0; i<DATA_VECTOR_SIZE_LMRST_TELEM; i++) {
    tapContents[tap_ptr++] = lmr_data[i] >> 8;     // MSB is stored first
    tapContents[tap_ptr++] = lmr_data[i] & 0x00FF; // LSB is stored last
  } /* for() */

  return 0; // no error
}
```

Flight Software: Usage

Each TAP must be initialized before use:

```
…
TAP_set_action(TAP_ID_LMRST, SEND_TAP_SDCARD);
TAP_set_size(TAP_ID_LMRST, SIZEOF_TAP_ID_LMRST);
TAP_set_interval(TAP_ID_LMRST, TAP_ID_LMRST_INTERVAL_DEFAULT);
TAP_set_carton_fn(TAP_ID_LMRST, carton_LMRST_fill_TAP);
…
```

The *action* controls how the data is stored (broadcast and/or SD Card).
The *size* depends on the data being collected.
The *interval* sets the period between successive telemetry captures.
The *carton function* uniquely captures the telemetry for this TAP.

The action and interval are often redefined while on orbit, based on current requirements.

Flight Software: Runtime

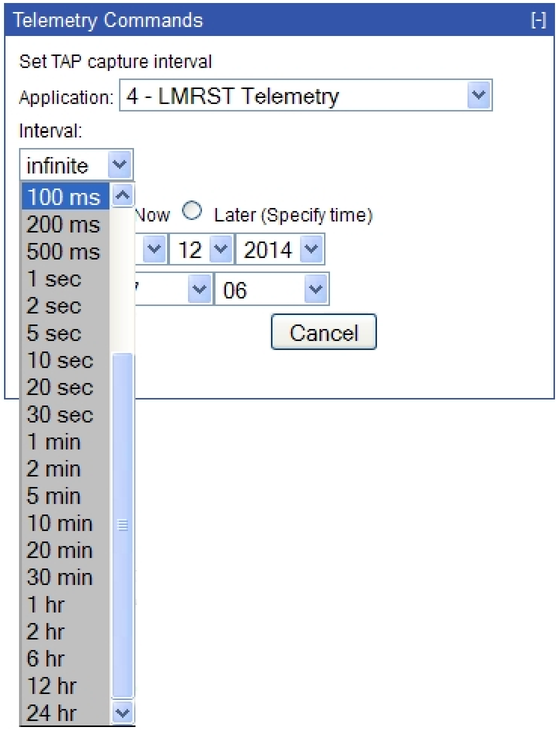
We've chosen to assign a unique *task* (the *Application* in *Telemetry Application Packet*) to each TAP ID in the LMRST-Sat FSW:

```
void task_TAP_LMRST(void) {

  TAP_set_action(TAP_ID_LMRST, SEND_TAP_SDCARD);
  TAP_set_size(TAP_ID_LMRST, SIZEOF_TAP_ID_LMRST);
  TAP_set_interval(TAP_ID_LMRST, TAP_ID_LMRST_INTERVAL_DEFAULT);
  TAP_set_carton_fn(TAP_ID_LMRST, carton_LMRST_fill_TAP);

  while(1) {
    OS_DelayTS(TAP_get_interval(TAP_ID_LMRST));
    TAP_push_TAP(TAP_ID_LMRST);
  } /* while() */

} /* task_TAP_LMRST() */
```

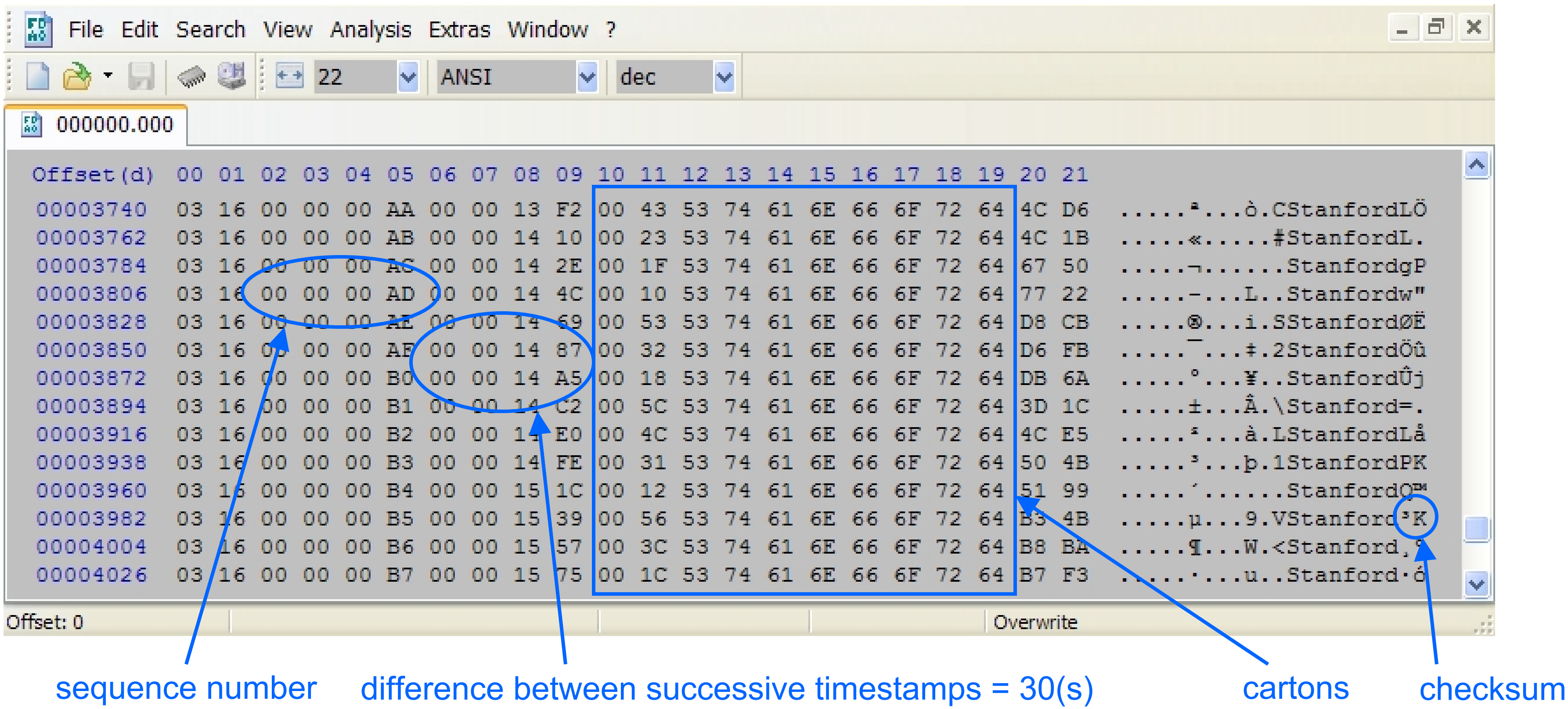


After TAP initialization, the TAP task runs periodically, as per the TAP's capture interval.

Each time the TAP task runs, it creates a (new) TAP packet by:

- *Adding a header with the TAP ID, sequence number and timestamp to the packet;*
- *Capturing the TAP's telemetry via its carton function and putting that data into the packet;*
- *Adding a checksum to the packet; and*
- *Broadcasting the TAP and/or storing it to a file on the SD Card.*

TAP File Structure



Hex view of LMRST-Sat's bus telemetry (TAP ID = 3, capture interval = 30s) as stored in file `\INIT\003\000000.000`.

10 bytes of TAP header, 2+8 bytes of data, 2-byte checksum -> 22 (0x16) bytes per TAP (TAP ID = 3).

Flight Software: File Management

A TAP's one-byte ID and 4-byte sequence number uniquely define where it will be stored on the SD Card (FAT16 format).

The folder where the files are stored is named after the TAP ID.

The filename (DOS 8.3 format) incorporates the most significant three bytes of the timestamp in its name.

The least significant byte of the sequence number indicates the position of the TAP in the file (i.e., the line number, starting with 1).

Examples:

TAP ID 7,	**Sequence Number 0:**	`INIT\007\000000.000`, **line 1**
TAP ID 12,	**Sequence Number 312:**	`INIT\012\000000.001`, **line 57**
TAP ID 4,	**Sequence Number 43,671:**	`INIT\004\000000.170`, **line 152**
TAP ID 5,	**Sequence Number 157,796:**	`INIT\005\000002.104`, **line 101**

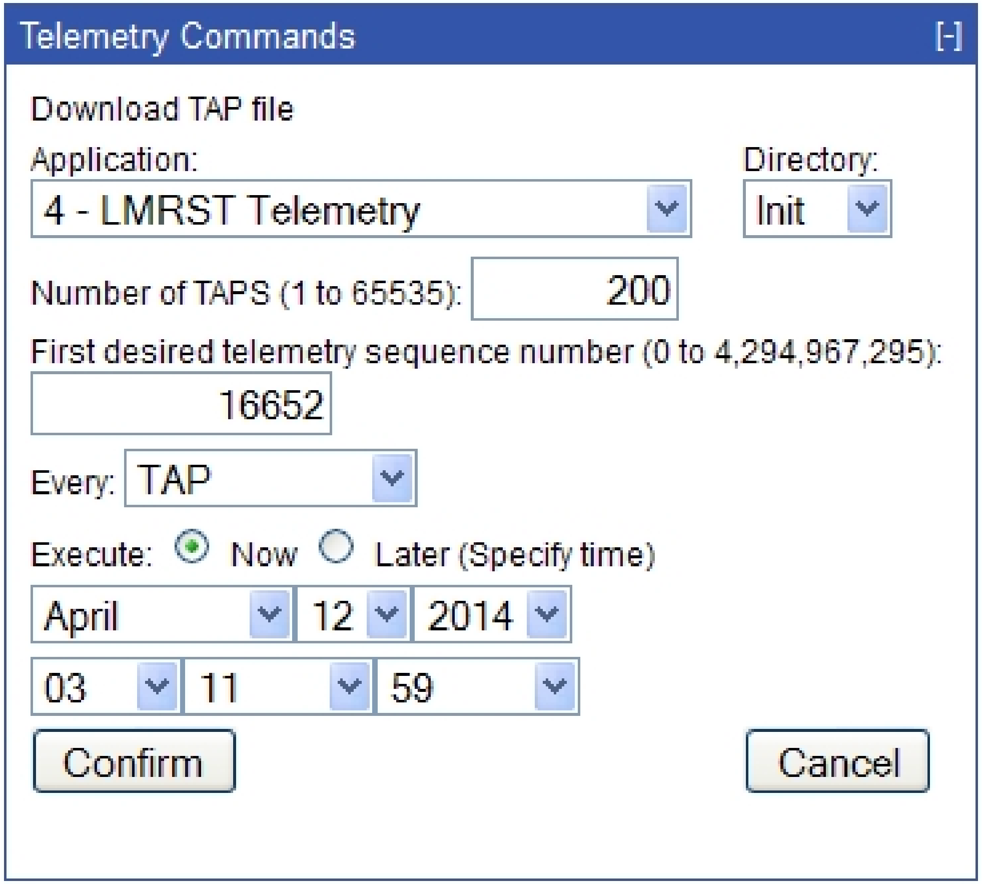
FAT16 is limited to 512 entries in the root folder. FAT16 subfolders are allocated with a linked list structure and can accommodate an ***unlimited*** *number of files.[1]*

Telemetry Retrieval

Telemetry is retrieved based on:
- **TAP ID**
- **Sequence Number**

Downloaded telemetry includes:
- **TAP ID**
- **Sequence Number**
- **(Onboard) Timestamp**

… and can have additional timestamps (e.g., reception time) as well.



Review TECSTARS Features

Size of each dataset is limited only by TAP packet format.

Each dataset is ID'd by its TAP ID, sequence number and timestamp.

TAP capture intervals and actions can be changed on-the-fly.

Auto-incrementing 32-bit sequence numbers for every TAP permit e.g. 50k days of a given TAP at 1 TAP/s.

Recommend that each TAP be associated with its own independent TECSTARS task, repeating at the specified capture interval (infinite = no telemetry capture). The capture interval can be trivially changed on demand.

Each TAP file is limited to 256 unique sequence number entries. No files limit.

Trivial segregation of collected telemetry (e.g., via `INIT` and `OPS` modes and their corresponding folders).

Collected onboard telemetry is easily erased (via file delete operations) and recycled if desired; a TAP's sequence numbers can be reused, esp. given the unique timestamp that accompanies each sequence number.

Telemetry is retrieved based on TAP ID and sequence number.

TECSTARS Performance On LMRST-Sat

10 bytes overhead per TAP packet. Added a 2-byte subsecond timestamp field to cartons, to permit faster than 1Hz telemetry capture (and tagging) rates → 12 bytes total overhead per packet.

Capture interval lookup scheme permits any predefined interval at only 1byte/TAP task: we implemented intervals from 20ms to 1 day.

All TAP tasks at the same priority except for the beacon task (higher). All other internal tasks run *higher* than the TAP tasks. System performance (i.e., telemetry capture at specified rates) degrades gracefully at high capture frequencies. CPU utilization <30% for low-frequency (i.e., > 5s capture interval) TAPs. SCLK = 500kHz.

TAP packets range in size from 22 to 106 bytes (55% to 11% overhead, resp.).

Essentially unlimited storage via 2GB SD Card: each unique TAP consumes ca. 40-120bytes of file storage.

Removable SD Card and hex file viewer make it easy to check TECSTARS operation by auditing the TAP files.

Requires <6KB of code and <1KB of RAM to manage 13 TAPS and their tasks.

TECSTARS Contributors

Andrew Nuttall

Avishai Weiss

Brendan Tseung

Brian Thompson

Bryan Lin

Cyrus Foster

Dawn Wheeler

Nicolas Lee

Randy Lum

Seiya Shimizu

Yonas Tesfaye

Andrew Kalman

REFERENCES

[1] “The "Maximum files per volume" given for FAT is not completely correct. FAT is limited to 512 entries in the <u>root</u> folder only; subfolders are allocated with a linked list structure and can accommodate any number of files. MS made this distinction clear in some old Win95 training materials. The closest convenient reference I can find right now is in the Win2000 ResKit (see "FAT16 vs FAT32", and other places). It doesn't explicitly state that subfolders are unlimited in size, but the ResKit always mentions the root folder when it mentions the 512 file limit. Thanks. “ *http://windowsitpro.com/systems-management/what-are-maximum-volume-sizes-and-maximum-file-sizes-various-windows-file-systems*