A Satellite Downlink Data Parsing Method (Postprint)

Authors: Zhao Haisheng, Fan Fengxia, Shumei Jia, Li Cheng-Kui

Date: 2019-03-25T00:00:00+00:00

Abstract

Satellite-acquired data is stored in bit- or byte-level formats that are predefined on the ground. Following deformatting and packetization of the downlinked satellite data, the ground segment must parse the data according to these predefined formats to generate readable decimal numbers. Although these formats exhibit substantial variations across different satellites and payloads, they all possess parsability. This paper proposes a hardware-software interface designed to define and store these parsing formats, while concurrently generating software-readable configuration files for data parsing based on these formats. This approach proves time-saving and labor-efficient for processing large volumes of data files, while simultaneously guaranteeing parsing accuracy.

Full Text

Preamble

A Decoding Method for Satellite Telemetry Data

Zhao Haisheng, Fan Fengxia, Jia Shumei, Li Chengkui
Key Laboratory of Particle Astrophysics, Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China

Abstract

Satellite-acquired data is stored in bit- or byte-level formats predefined by ground systems. After telemetry data undergoes deformatting and packetization, ground systems must parse these data according to predefined formats to generate readable decimal values. Although these formats vary significantly across different satellites and payloads, they all share the property of being parsable. This paper designs a software-hardware interface to define and store these parsing formats, and to generate software-readable configuration files for

data decoding. This approach saves considerable time and effort for large data volumes while ensuring parsing accuracy.

Keywords: Satellite Data, Data Packets, Data Decoding, Data Format

0 Introduction

Satellite-acquired data is stored in predefined bit- or byte-level formats, and ground-based decoding strictly follows these specifications. However, during satellite development—particularly for payload instrumentation (the hardware side)—these formats frequently change, creating significant challenges for data parsing (the software side). Inadequate or untimely communication between hardware and software teams can impede project progress. Even when formats remain stable, manual parsing is time-consuming and labor-intensive.

Comparative analysis of these predefined formats reveals consistent underlying rules, making it possible to design a software-hardware interface and decoding methodology that reduces communication costs and enables adaptive parsing. Data parsing typically operates at the packet level. Satellite telemetry packets can be categorized by source: platform-acquired packets, payload engineering packets, and payload science packets [1]. Platform packets generally have simple, fixed formats, such as positioning, attitude, and satellite temperature data. Engineering data is time-driven with diverse data types but small total volume—for example, driver clock status or operation mode packets typically range from tens to hundreds of bytes. Hardware sometimes rotates data collection across different electronics or instruments, resulting in varying formats and lengths that can be treated as separate packets, leading to potential nested structures. Science data volumes are substantial; given limited onboard storage, some physical quantities are stored in two parts. In certain cases, a physical event requires data from multiple threshold-exceeding channels with varying channel counts per trigger, resulting in variable-length packets. Important science packets include physics event packets, noise baseline packets, time carry packets, and GPS pulse packets, typically occupying several to dozens of bytes each.

These raw packets undergo encoding, scrambling, and modulation onboard before transmission to ground via telemetry channels. Ground systems perform demodulation, descrambling, decryption, deformatting, and depacketization, storing packets of the same type in files identified by unique fields. For example, the Hard X-ray Modulation Telescope (HXMT) satellite [2] uses APID (Application Process Identifier) in filenames. This paper focuses on parsing such post-deformatting data.

For both the launched HXMT and the POLAR detector on Tiangong-2 [3], onboard data storage follows big-endian format (most significant bit first). Data streams are processed in first-transmitted, first-stored, first-read order both onboard and on the ground. When handling multi-byte data, ground personnel must account for potential byte order mismatches with computer memory architecture. Some systems may reverse byte order, requiring a flip operation during

parsing.

This paper presents a decoding method and software-hardware interface for satellite telemetry data, integrated into a unified system designed for easy adoption across satellites to reduce communication overhead and accelerate project timelines.

1 System Overview

The system is database-centric and comprises two main modules. [Figure 1: see original paper] illustrates the system architecture. The database stores parsing formats written according to the rules defined in this paper. Module 1 is the interface module, enabling hardware engineers to write parsing formats into the database and retrieve them for display. Module 2 is the data decoding module, which reads formats from the database to generate configuration files and then parses data accordingly.

The database contains two table types: a packet property table (registry) defining attributes for one or multiple packet types, and packet format tables (format tables) that specify parsing formats for individual packets. To decode a packet type, the system first queries the registry for packet properties—including length, start/end keywords, etc.—then retrieves the appropriate format table. Configuration files mirror the database tables in content and naming, effectively serving as local copies. Subsequent descriptions refer to both collectively as the database.

1.1 Registry Table

The registry table characterizes fundamental packet attributes:

- **title:** Packet name, corresponding to the format table name (must be unique)
- **type:** Packet type, the key identifier. A data file may contain one or multiple packet types; assigning identical types enables rapid file-to-packet mapping. For HXMT, physics event and calibration event packets share the same event type. We recommend using APID as the packet type
- **header:** Marks packet start, comprising start position, end position, and value. Two values are supported because some payloads alternate header values to indicate packet loss (e.g., 0x103B followed by 0x183B)
- **mode:** Characterizes packet properties. Same-type packets may have different modes—for example, a header indicating event packets while mode distinguishes science events from calibration events. Comprises start position, end position, and value
- **main:** Primary packet flag. Engineering data files typically contain one primary packet that may include multiple nested packets. Science data files may contain multiple parallel packet types with varying frequencies; the most frequent is designated primary for efficient access. Only one primary packet per type is allowed

- **length:** Defined by start position, end position, and preset value. If positions exist, the value at those positions indicates packet length. Otherwise, the preset value specifies fixed-length packets. Absence of both indicates variable-length packets
- **tail:** End marker, defined by start position, end position, and preset value. For variable-length packets, start/end positions should be absent; hardware should set a preset value (e.g., 0xFFFF)
- **validation:** Special position/value checks, typically two optional (start, end, value) sets
- **crc:** Packet checksum position and length. For variable-length packets, hardware should place this immediately before the tail [4]

For parsing convenience, either **length** or **tail** must be present; **length** may consist solely of a preset value. In nested hardware designs, nested packets must contain either **header** or **mode** and must align to whole bytes. Hardware nesting relates to electronics readout—for instance, when collecting data from different electronic channels with identical structures, or different structures with equal data volumes. Software-defined nesting allows grouping data structures without requiring header/mode, needing only a unique title.

Primary and nested packets with tails must use distinct values. Variable-length structures should be positioned near packet ends, with ideally only one variable-length structure per packet.

Key principles: **type** and **header/mode** uniquely identify a format table; **title** is unique and used for format table lookup; **validation** and **crc** perform integrity checks—packets failing crc verification are typically discarded.

Most common packets require only **title**, **type**, **length**, and **crc**; **header**, **mode**, **tail**, and **validation** are optional. For absent fields, hardware should set attribute values to -1 (e.g., if mode is absent, set its start, end, and value to -1). Positions are relative to packet start in bits or bytes.

1.2 Format Table

The format table records parsing formats for each packet type, corresponding to registry entries:

- **id:** Sequential identifier for parsed variables (starting from 0), enabling variable retrieval by ID
- **title:** Parsed variable name, enabling retrieval by name
- **position:** Variable address, supporting two positions (low-order and high-order bits). Most variables use one position; set the other to -1. If only start position exists with end position = -1, the variable has variable length. A flag indicates absolute (from packet start) or relative (from previous variable) positioning. Variable-length structures must occupy whole bytes, with padding if necessary
- **type:** Variable type: 0 = standard variable or fixed-length continuous

array; 1 = variable-length continuous array (length determined by parsing other information); 2 = discontinuous variable-length array (elements spaced at intervals, typical for multi-channel/multi-cell data acquisition). For type=0, **position** defines the entire parsing range; for types 1-2, **position** specifies only the first element

- **reference:** Used when type=3 (nested packet), containing the referenced packet name (registry title). Empty values allow multiple nested packet types
- **repeat:** For type=0, array dimension (dividing the position-specified range into **repeat** segments). For non-arrays, set to 1. For type=2, **repeat** indicates spacing between elements (e.g., if first element occupies [a,b], next occupies [b+repeat+1, b+repeat+1+b-a])
- **flag:** Parsing unit: BYTE or BIT

This system supports two variable-length structures: continuous arrays and discontinuous arrays (particularly useful for multi-channel acquisition). For parsing simplicity, each packet may contain only one variable-length structure, which must occupy whole bytes with padding as needed. The next variable' s position is calculated from the byte boundary following the variable-length structure.

Software-defined nested packets need not align to byte boundaries but require explicit references. For example, the pattern "ABCABCABC···" can define "ABC" as a nested packet, creating a variable-length array structure.

2 Interface Module

The interface module manages registry and format table records, display, and configuration file generation. This allows hardware teams to modify parsing formats without affecting software decoding. The interface uses web-based design (though QT [5] is also viable). shows personnel roles and permissions. The website hierarchy appears in [Figure 2: see original paper]: under a satellite project, platform packets and payload packets are primary categories. Payload packets subdivide into science and engineering packets, each corresponding to a registry table and multiple format tables.

The backend database uses MySQL [6], where each satellite project corresponds to one database. Registry tables are named: SatelliteName + "" + *PayloadName* + "" + DataCategoryName (e.g., science, engineering). Format tables are named: SatelliteName + "" + *PayloadName* + "" + DataCategoryName + PacketName.

The decoding module reads configuration files rather than the database directly, decoupling the decoder from the database and enabling version control. When payload formats update, configuration files can be saved with timestamps, which is valuable for data traceability.

3 Decoding Module

Data decoding targets both payload test data and satellite telemetry data. While telemetry data is well-organized, it involves multiple organizations requiring interface coordination. Test data is less structured and payload-specific. This section focuses on file identification, registry reading, format table parsing, and special data handling.

Satellite telemetry typically includes data type identifiers—for HXMT, APID marks data files, corresponding to packet types (registry **type**). We recommend including APID or other type keywords in filenames. If using APID naming, establish mappings between APID, packet type, data category, and payload. A simpler approach embeds packet category, data category, and payload keywords directly in filenames.

During parsing, the system extracts packet category, data category, and payload information from the filename to query the registry (matching **type**) and identify required format tables (via registry **title**). Note that one data file may correspond to multiple format tables.

Data acquisition begins by identifying the primary packet from registry query results (using **main** flag). If no primary packet exists, the structure is parallel—all queried format tables may appear, requiring keyword matching to identify the correct format table. With a primary packet, keyword matching is attempted first; if unsuccessful (common when high-frequency packets are artificially designated primary), other query results are tested. Once the format packet is determined, complete packet information is retrieved: fixed-length packets use the **length** keyword, while variable-length packets search for the **tail** keyword.

Keyword matching considers **header**, **mode**, and **validation**. Packets failing **crc** verification can be discarded (if packet length is uncertain, searching for **header**/**tail** patterns in the data is possible).

Parsing variable-length structures requires determining array dimensions. When packet length is known (via **length** or **tail**), dimensions can be calculated since the variable-length start position and tail offset are known. More commonly, variable-length structures correlate with other variables—where a variable's value or bit count indicates length. User knowledge of these relationships simplifies processing.

The system can be file-driven: upon detecting new file transfers, it automatically initiates parsing, provided users predefine output data structures, acquisition interfaces, and formats. The system does not provide direct "file-to-file" translation because: (1) not all parsed quantities need recording—only user-interest variables; (2) files may contain multiple parsing formats and nesting patterns complicating automatic output 编排; (3) one raw file may generate multiple output files. This approach also accommodates design limitations, such as variable correlations.

Data acquisition uses **title** or **id** from format tables to retrieve parsed variables. Output typically uses FITS format [7], though one parsing run may generate multiple output files.

The decoding system is the first step in satellite data product generation. While different satellites and payloads have unique data designs, all follow parsable principles—forming this system's foundation. Decoding is equally critical during ground testing, where formats evolve with payload development. This system adapts to such changes, and its accuracy underpins scientific results. The universal decoding method mentioned in reference [1] (Section 2.1) is essentially a simplified version of this system' s decoding module, successfully applied to POLAR and ME [8] ground data parsing. This universal approach prevents software engineers from manually processing numerous data files, which is crucial for ensuring scientific productivity.

This system handles packet-level parsing only; post-parsing data processing and storage require user-defined structures and formats. Users access parsed data through this system' s interfaces. With these components in place, the system operates automatically and collaborates with downstream software to generate data products.

References

[1] Zhao Haisheng, Ge Mingyu, Li Zhengheng, Nie Jianyin, Song Liming. A Method of Data Preprocessing for Astronomical Satellite[J]. Astronomical Research and Technology, 2017, 14(3): 376-381.

[2] Li Ti-Pei, Wu Mei. The hard X-ray modulation telescope mission[J]. Physics, 2008, 37(09): 648-651.

[3] N. Produit, Barao F, Deluit S, et al. POLAR: a compact detector for gamma-ray bursts photon polarization measurements[C], Nucl. Instr. and Meth. A, 2005, 550(3): 616-625.

[4] Han Jiawei, Zhang Hongqun. Error correction technology based on CCSDS standard for remote sensing satellite[J]. Chin. J. Space Sci., 2009, 29(1): 112-116.

[5] Zhou Xinlei, Wang Wei, Wang Feng, Deng Hui, Liu Cuiyin, Hu Jie, Li Shaoliang, Wang Lulu, Zhou Shiran. Design and Implementation of a Multi-Monitor Display System Based on the Qt for NAOC MUSER Observations[J]. Astronomical Research & Technology, 2015, 12(4): 503-509.

[6] Li Hui-xian, Sang Jian, Wang Sha, Luo A-li. The Database Design of LAMOST Based on MYSQL/LINUX[J], Astronomical Research and Technology, 2006, Vol.3 Issue(1): 56-63.

[7] Wells D C, Greisen E W & Harten R H. FITS: A Flexible Image Transport System[J]. A&AS, 1981, 44: 363-370; A Primer on the FITS Data Format[R/OL]: http://fits.gsfc.nasa.gov/fits_primer.html.

[8] Cao Xuelei, Jiang Weichun, Zhang Wanchang, Meng Bin, Yang Sheng, Luo Tao, Gu Yudong, Tan Ying. Design and verification of Medium Energy Telescope on board HXMT satellite[J], Spacecraft Engineering, 2018, 27(05): 127-133.

Note: Figure translations are in progress. See original paper for figures.