

ELS-IDD-15-03561-V4.0

March 7, 2005

**ELS IDFS Design Document
Version 4.0**

For The
**Mars Express
ASPERA-3 Processing
and Archiving Facility
(APAF)**

SwRI Project No. 15-03561

Prepared by:
C.Gonzalez

SOUTHWEST RESEARCH INSTITUTE
Space Science and Engineering Division
Post Office Drawer 25810, 6220 Culebra Road
San Antonio, Texas 78228-0510

ELS-IDD-15-03561-V4.0

March 7, 2005

ELS IDFS DESIGN DOCUMENT

Version 4.0

For The

**Mars Express
ASPERA-3 Processing
and Archiving Facility
(APAF)**

SwRI Project No. 15-03561

Approved By: _____

Dr. J.David Winningham
Principal Investigator

_____ Date

Prepared By: _____

Carrie A. Gonzalez
Lead Software Engineer

_____ Date

Reviewed By: _____

Sandee Jeffers
Software Project Manager

_____ Date

Reviewed By: _____

Dr. Rudy Frahm
Co-Investigator

_____ Date

Revision Log

| Revision | Release Date | Changes to Document |
|-----------------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Version 1.0 | 12/07/01 | <ul style="list-style-type: none"> • Initial Release |
| Version 2.0 | 06/10/03 | <ul style="list-style-type: none"> • Revision Log was added. • Information Modified According to Main Unit Software User's Guide (ME-ASP-MA-005); Issue 2, Rev. 0, dated 2003-01-29 and according to e-mail clarifications based upon this version of the document. • Revised ELSSCIH and ELSSCIL to remove the last step since it is flyback |
| Version 3.0 | 10/18/03 | <ul style="list-style-type: none"> • Information Modified According to Main Unit Software User's Guide (ME-ASP-MA-0005); Issue 3, Rev. 0, dated 2003-06-21. Mainly, updated sections explaining definition of the status byte sw_mode and the ASCII table definitions within the VIDF files for all virtual instruments. • Added status byte to the science virtuals which have summation calibration data. • Modified calibration sets for science virtuals since summations resulted in overflow of 16-bit values. In addition, Time Summation and Step Summation changed from state indicators to true values so no need for ASCII definitions. |
| Version 3.1 | 12/12/03 | <ul style="list-style-type: none"> • Modified tolerance values for -12V Power Line in Section 7.1. |
| Version 3.2 | 01/29/04 | <ul style="list-style-type: none"> • Corrected bit location for els_enable_hv parameter, referenced in the D_QUAL sections. |
| Version 3.3 | 04/02/04 | <ul style="list-style-type: none"> • Updated sections 10.2.7 and 11.2.7 since Energy Summation is saved as Raw value; updated VIDFs in Appendix A to baselined versions. |
| Version 3.4 | 09/02/04 | <ul style="list-style-type: none"> • Updated Tables 10-3 and 11-3, sections 10.1, 10.3.4, 11.1, and 11.3.4 with regards to the Byte location for the compression information from 28 to 29 based upon e-mail confirmation from FMI. • Updated VIDFs in Appendix A to latest baselined versions. |
| Version 3.5 | 03/01/05 | <ul style="list-style-type: none"> • Added reconstruction formulas for Deflection Monitor and Deflection Reference values in sections 8.1 and 9.1. • Updated VIDFs in Appendix A to latest baselined versions. |
| Version 4.0 | 03/07/05 | <ul style="list-style-type: none"> • Added 2 new calibration sets for unit conversion and re-ordered all calibration sets for ELSSCIL and ELSSCIH. • Updated VIDFs in Appendix A to latest baselined versions for ELSSCIL and ELSSCIH. |

TABLE OF CONTENTS

| | | |
|------------|------------------------------------------------------------------|-----------|
| 1.0 | Scope | 1 |
| 1.1 | Project Identification | 1 |
| 1.2 | System Overview..... | 1 |
| 1.3 | Data Flow Overview..... | 2 |
| 1.4 | Document Overview..... | 2 |
| 1.5 | Related Documents..... | 3 |
| 1.6 | Requirements Traceability..... | 3 |
| 2.0 | Packet Header for ELS Science and Housekeeping Data | 4 |
| 3.0 | Notes for ELS Data | 6 |
| 4.0 | Introduction to IDFS | 7 |
| 5.0 | Breakdown of ELS Data into Virtual Instruments..... | 10 |
| 6.0 | Definition for Virtual Instrument ELSENG8 | 11 |
| 6.1 | Notes for ELSENG8 virtual instrument | 13 |
| 6.1.1 | ELS Temperature Monitor..... | 13 |
| 6.1.2 | ELS Bias Monitor..... | 15 |
| 6.1.3 | ELS Bias Reference..... | 15 |
| 6.1.4 | ELS -5V Screen Grid Reference | 16 |
| 6.1.5 | ELS -5V Screen Grid Monitor..... | 16 |
| 6.2 | ELSENG8 Header Record Format | 18 |
| 6.2.1 | Setting of <i>n_sample</i> header record element..... | 18 |
| 6.2.2 | Setting of <i>scan_index</i> header record element | 18 |
| 6.2.3 | Setting of <i>sensor_index</i> header record element | 19 |
| 6.2.4 | Setting of <i>d_qual</i> header record element..... | 19 |
| 6.2.5 | Setting of <i>mode_index</i> header record element..... | 20 |
| 6.3 | ELSENG8 Data Record Format | 21 |
| 6.3.1 | Setting of <i>dr_time</i> data record element..... | 21 |
| 6.3.2 | Setting of <i>spin</i> data record element | 21 |
| 6.3.3 | Setting of <i>sun_sen</i> data record element | 22 |
| 6.3.4 | Setting of <i>data_array</i> data record element | 23 |
| 7.0 | Definition for Virtual Instrument ELSENGS..... | 25 |
| 7.1 | Notes for ELSENGS virtual instrument | 26 |
| 7.2 | ELSENGS Header Record Format | 27 |
| 7.2.1 | Setting of <i>n_sample</i> header record element..... | 27 |
| 7.2.2 | Setting of <i>scan_index</i> header record element | 27 |
| 7.2.3 | Setting of <i>sensor_index</i> header record element | 28 |
| 7.2.4 | Setting of <i>d_qual</i> header record element..... | 28 |
| 7.2.5 | Setting of <i>mode_index</i> header record element..... | 29 |
| 7.3 | ELSENGS Data Record Format | 30 |
| 7.3.1 | Setting of <i>dr_time</i> data record element..... | 30 |
| 7.3.2 | Setting of <i>spin</i> data record element | 30 |
| 7.3.3 | Setting of <i>sun_sen</i> data record element | 31 |
| 7.3.4 | Setting of <i>data_array</i> data record element | 32 |
| 8.0 | Definition for Virtual Instrument ELSSWPL..... | 34 |

| | | |
|-------------|------------------------------------------------------------|-----------|
| 8.1 | Notes for ELSSWPL virtual instrument..... | 36 |
| 8.1.1 | ELS Deflection Voltage Reference | 36 |
| 8.1.2 | ELS Deflection Voltage Monitor | 36 |
| 8.2 | ELSSWPL Header Record Format..... | 37 |
| 8.2.1 | Setting of <i>n_sample</i> header record element..... | 37 |
| 8.2.2 | Setting of <i>scan_index</i> header record element | 38 |
| 8.2.3 | Setting of <i>sensor_index</i> header record element | 38 |
| 8.2.4 | Setting of <i>d_qual</i> header record element..... | 38 |
| 8.2.5 | Setting of <i>mode_index</i> header record element..... | 39 |
| 8.3 | ELSSWPL Data Record Format..... | 40 |
| 8.3.1 | Setting of <i>dr_time</i> data record element..... | 40 |
| 8.3.2 | Setting of <i>spin</i> data record element | 40 |
| 8.3.3 | Setting of <i>sun_sen</i> data record element | 41 |
| 8.3.4 | Setting of <i>data_array</i> data record element..... | 42 |
| 9.0 | Definition for Virtual Instrument ELSSWPH | 45 |
| 9.1 | Notes for ELSSWPH virtual instrument | 47 |
| 9.1.1 | ELS Deflection Voltage Reference | 47 |
| 9.1.2 | ELS Deflection Voltage Monitor | 47 |
| 9.2 | ELSSWPH Header Record Format | 48 |
| 9.2.1 | Setting of <i>n_sample</i> header record element..... | 48 |
| 9.2.2 | Setting of <i>scan_index</i> header record element | 49 |
| 9.2.3 | Setting of <i>sensor_index</i> header record element | 49 |
| 9.2.4 | Setting of <i>d_qual</i> header record element..... | 49 |
| 9.2.5 | Setting of <i>mode_index</i> header record element..... | 50 |
| 9.3 | ELSSWPH Data Record Format | 51 |
| 9.3.1 | Setting of <i>dr_time</i> data record element..... | 51 |
| 9.3.2 | Setting of <i>spin</i> data record element | 51 |
| 9.3.3 | Setting of <i>sun_sen</i> data record element | 52 |
| 9.3.4 | Setting of <i>data_array</i> data record element..... | 53 |
| 10.0 | Definition for Virtual Instrument ELSSCIL..... | 56 |
| 10.1 | Notes for ELSSCIL virtual instrument..... | 60 |
| 10.1.1 | ELS Calibration Data..... | 61 |
| 10.2 | ELSSCIL Header Record Format..... | 63 |
| 10.2.1 | Setting of <i>swp_reset</i> header record element | 63 |
| 10.2.2 | Setting of <i>n_sen</i> header record element..... | 64 |
| 10.2.3 | Setting of <i>n_sample</i> header record element..... | 64 |
| 10.2.4 | Setting of <i>scan_index</i> header record element | 64 |
| 10.2.5 | Setting of <i>sensor_index</i> header record element | 64 |
| 10.2.6 | Setting of <i>d_qual</i> header record element..... | 65 |
| 10.2.7 | Setting of <i>mode_index</i> header record element..... | 66 |
| 10.3 | ELSSCIL Data Record Format..... | 67 |
| 10.3.1 | Setting of <i>dr_time</i> data record element..... | 67 |
| 10.3.2 | Setting of <i>spin</i> data record element | 67 |
| 10.3.3 | Setting of <i>sun_sen</i> data record element | 68 |
| 10.3.4 | Setting of <i>data_array</i> data record element..... | 69 |
| 11.0 | Definition for Virtual Instrument ELSSCIH | 73 |
| 11.1 | Notes for ELSSCIH virtual instrument | 77 |

| | | |
|------------------------------------------------------------------------------|------------------------------------------------------------|-------------|
| 11.1.1 | ELS Calibration Data..... | 78 |
| 11.2 | ELSSCIH Header Record Format | 80 |
| 11.2.1 | Setting of <i>swp_reset</i> header record element | 80 |
| 11.2.2 | Setting of <i>n_sen</i> header record element..... | 81 |
| 11.2.3 | Setting of <i>n_sample</i> header record element..... | 81 |
| 11.2.4 | Setting of <i>scan_index</i> header record element | 81 |
| 11.2.5 | Setting of <i>sensor_index</i> header record element | 81 |
| 11.2.6 | Setting of <i>d_qual</i> header record element..... | 82 |
| 11.2.7 | Setting of <i>mode_index</i> header record element..... | 83 |
| 11.3 | ELSSCIH Data Record Format | 84 |
| 11.3.1 | Setting of <i>dr_time</i> data record element..... | 84 |
| 11.3.2 | Setting of <i>spin</i> data record element | 84 |
| 11.3.3 | Setting of <i>sun_sen</i> data record element | 85 |
| 11.3.4 | Setting of <i>data_array</i> data record element..... | 86 |
| Appendix A - Virtual Instrument Description Files (VIDF) for ELS..... | | A-1 |
| ELSENG8 VIDF File..... | | A-1 |
| ELSENGS VIDF File..... | | A-5 |
| ELSSWPL VIDF File | | A-8 |
| ELSSWPH VIDF File..... | | A-23 |
| ELSSCIL VIDF File | | A-38 |
| ELSSCIH VIDF File..... | | A-67 |
| Appendix B - ELS Instrument Operation Scenarios..... | | B-1 |
| Appendix C – Decompression of ELS Packets..... | | C-1 |

LIST OF FIGURES

| | |
|-------------------------------------------------------------------------|-----------|
| <i>Figure 1 Bit Designation</i> | <i>4</i> |
| <i>Figure 2 IDFS sensor data array.....</i> | <i>9</i> |
| <i>Figure 3 IDFS calibration data array</i> | <i>9</i> |
| <i>Figure 4 Deflection Command Bit Structure</i> | <i>36</i> |
| <i>Figure 5 Data Array for ELSSWPL.....</i> | <i>43</i> |
| <i>Figure 6 Deflection Command Bit Structure</i> | <i>47</i> |
| <i>Figure 7 Data Array for ELSSWPH</i> | <i>54</i> |
| <i>Figure 8 Sector Mask Anode Number / Bit Number Association.....</i> | <i>60</i> |
| <i>Figure 9 Data Array for ELSSCIL.....</i> | <i>71</i> |
| <i>Figure 10 Sector Mask Anode Number / Bit Number Association.....</i> | <i>77</i> |
| <i>Figure 11 Data Array for ELSSCIH</i> | <i>88</i> |

LIST OF TABLES

| | |
|---------------------------------------------------------|-----------|
| <i>Table 2-1 ELS Source Data Packets.....</i> | <i>5</i> |
| <i>Table 6-1 ELSENG8 Sensor Definition</i> | <i>11</i> |
| <i>Table 6-2 ELSENG8 Status Flag Definition.....</i> | <i>12</i> |
| <i>Table 6-3 ELSENG8 Header Record Definition.....</i> | <i>18</i> |
| <i>Table 6-4 ELSENG8 Data Record Definition.....</i> | <i>21</i> |
| <i>Table 7-1 ELSENG8 Sensor Definition</i> | <i>25</i> |
| <i>Table 7-2 ELSENG8 Status Flag Definition.....</i> | <i>25</i> |
| <i>Table 7-3 ELSENG8 Header Record Definition.....</i> | <i>27</i> |
| <i>Table 7-4 ELSENG8 Data Record Definition.....</i> | <i>30</i> |
| <i>Table 8-1 ELSSWPL Sensor Definition</i> | <i>34</i> |
| <i>Table 8-2 ELSSWPL Calibration Definition.....</i> | <i>34</i> |
| <i>Table 8-3 ELSSWPL Status Flag Definition.....</i> | <i>35</i> |
| <i>Table 8-4 ELSSWPL Header Record Definition.....</i> | <i>37</i> |
| <i>Table 8-5 ELSSWPL Data Record Definition.....</i> | <i>40</i> |
| <i>Table 9-1 ELSSWPH Sensor Definition.....</i> | <i>45</i> |
| <i>Table 9-2 ELSSWPH Calibration Definition.....</i> | <i>45</i> |
| <i>Table 9-3 ELSSWPH Status Flag Definition</i> | <i>46</i> |
| <i>Table 9-4 ELSSWPH Header Record Definition</i> | <i>48</i> |
| <i>Table 9-5 ELSSWPH Data Record Definition</i> | <i>51</i> |
| <i>Table 10-1 ELSSCIL Sensor Definition</i> | <i>56</i> |
| <i>Table 10-2 ELSSCIL Calibration Definition</i> | <i>57</i> |
| <i>Table 10-3 ELSSCIL Status Flag Definition.....</i> | <i>58</i> |
| <i>Table 10-4 ELSSCIL Header Record Definition.....</i> | <i>63</i> |
| <i>Table 10-5 ELSSCIL Data Record Definition.....</i> | <i>67</i> |
| <i>Table 11-1 ELSSCIH Sensor Definition</i> | <i>73</i> |
| <i>Table 11-2 ELSSCIH Calibration Definition.....</i> | <i>74</i> |
| <i>Table 11-3 ELSSCIH Status Flag Definition</i> | <i>75</i> |
| <i>Table 11-4 ELSSCIH Header Record Definition.....</i> | <i>80</i> |
| <i>Table 11-5 ELSSCIH Data Record Definition.....</i> | <i>84</i> |

ACRONYMS

| | |
|----------|-----------------------------------------------------------------|
| APAF | ASPERA-3 Processing and Archiving Facility |
| ASPERA-3 | Analyzer of Space Plasma and Energetic Atoms (3rd Version) |
| CCSDS | Consultative Committee for Space Data Systems |
| DPU | Data Processing Unit (of the ASPERA-3 instrument package) |
| ELS | Electron Spectrometer (of the ASPERA-3 instrument package) |
| ESA | European Space Agency |
| ESOC | European Science Operations Center |
| FMI | Finnish Meteorological Institute (Helsinki, Finland) |
| IDFS | Instrument Data File Set or Instrument Description File Set |
| IMA | Ion Mass Analyzer (of the ASPERA-3 instrument package) |
| MEX | Mars Express |
| MU | Main Unit (of the ASPERA-3 instrument package) – same as DPU |
| NASA | National Aeronautics and Space Administration |
| NISN | NASA Integrated Services Network |
| NPD | Neutral Particle Detector (of the ASPERA-3 instrument package) |
| NPI | Neutral Particle Imager (of the ASPERA-3 instrument package) |
| OA | Orbit / Attitude (information from the Mars Express spacecraft) |
| PDS | Planetary Data System |
| PUS | Packet Utilization Standard |
| SCET | Space-Craft Elapsed Time |
| SGICD | Space / Ground Interface Control Document |
| SU | Scanning Unit (of the ASPERA-3 instrument package) |
| SwRI | Southwest Research Institute |
| VIDF | Virtual Instrument Description File |

1.0 Scope

1.1 Project Identification

| | |
|----------------------------------|-------------------------|
| Project Title: | ASPERA for Mars Express |
| Project Number: | 15-02853 / 15-03561 |
| Contract Number: | NASW-99030 / NASW-00003 |
| Principal Investigator: | Winningham, John D. |
| Software Project Manager: | Jeffers, Sandee J. |
| Start Date: | June 14, 1999 |
| End Date: | September 20, 2007 |

1.2 System Overview

The ASPERA-3 instrument package (or ASPERA-3 experiment) will be flown on the Mars Express (MEX) mission of the European Space Agency (ESA) and will be launched in June 2003 according to the current schedule. ASPERA-3 contains a number of different sensors that will measure the particles, neutral atoms, and fields in the near Martian environment. Southwest Research Institute (SwRI) is providing the data system to produce data products in a form suitable for analysis and archiving. These data products will be put into a form known as the Instrument Data File Set (IDFS).

The ASPERA-3 Processing and Archiving Facility (APAF) is a ground data system responsible for processing all of the ASPERA-3 telemetry. The APAF data system acquires the telemetry data via NISN, processes the data into IDFS data sets, distributes the IDFS data sets to the ASPERA-3 team, provides web-based displays for science analysis and public view, stores the telemetry and IDFS data sets on a local SwRI archive, and submits the ASPERA-3 IDFS data sets to PDS for long-term archival.

The first step in defining the IDFS data sets is to identify the physical instruments that make up the ASPERA-3 experiment and any ancillary data necessary for scientific analysis. There are six components of the ASPERA-3 package, plus the orbit and attitude data from the spacecraft:

1. Main Unit (MU)
2. Electron Spectrometer (ELS)
3. Ion Mass Analyzer (IMA)
4. Neutral Particle Detector (NPD)
5. Neutral Particle Imager (NPI)
6. Scanning Unit (SU)
7. Orbit/Attitude (OA)

Each of the physical components will be divided into logical groups (called virtual instruments) in which each logical group will be formatted as an IDFS data set.

1.3 Data Flow Overview

Daily files of ASPERA-3 telemetry and spacecraft OA telemetry will be acquired by SwRI from ESOC (European Science Operations Center) by way of NASA, NISN. The APAF will be used to process all of the ASPERA-3 and spacecraft OA telemetry. It is understood, at this time, that the data records made available from ESOC will be in packet format (refer to Section 2.0 for a detailed explanation of the telemetry source packet structure), time ordered, with duplicates removed and missing packets padded out. Intermediate files of cleaned-up ASPERA-3 and spacecraft OA telemetry shall be generated by the APAF system in the event that cleaned-up telemetry is not provided by ESOC. The APAF will split the ASPERA-3 and spacecraft OA telemetry into the seven logical groups listed above in Section 1.2. The ELS IDFS production code will read the data file generated by the APAF which contains ELS-only packets and generate the IDFS data and header files.

1.4 Document Overview

This document will fully describe all of the data products contained within the IDFS data sets which have been defined for the virtual instruments associated with the ELS instrument flown on the ASPERA-3 instrument package for the Mars Express mission. The document is organized into various sections which

- 1) Describe the telemetry packets that are to be processed by the ELS-specific IDFS production code (Section 2.0).
- 2) Provide some background information explaining how compression / summation techniques used to packetize the data and how operational characteristics of the ELS instrument will be represented within the IDFS data sets created (Section 3.0).
- 3) Provide a brief introduction into the IDFS paradigm, with emphasis placed on the make-up of the data matrix contained in each data record (Section 4.0).
- 4) Provide a listing of the IDFS virtual instruments defined for the ELS instrument, with a brief explanation of the decision process utilized to break up the ELS data into the various streams (Section 5.0).
- 5) Provide a section for each of the IDFS virtual instruments defined for ELS, with each section containing information that describes (Sections 6.0 – 11.0):
 - the sensors contained within each virtual instrument,
 - the telemetry source (housekeeping or science),
 - the packet field name and byte/bit location within the telemetry packet,
 - notes pertinent to the grouping of the ELS telemetry data into the IDFS virtual instrument,
 - data values / data sources pertinent to IDFS header record field elements,

- data values / data sources pertinent to IDFS data record field elements.

It is the intent that this document serve as the basis from which the IDFS production software can be implemented. The actual details of the IDFS production code will be left up to the individual programmer.

1.5 Related Documents

ESA Mars Express Space / Ground Interface Control Document (SGICD); ME-ESC-IF-5001, Issue 2.0, December 20 1999

ASPERA-3 Main Unit Software User's Guide; ME-ASP-MA-0005, Issue 3, released June 21, 2003

SwRI Instrument Description File System Definition: 2.1 Document Release E

APAF Software Requirements Specification: APAF-SRS-15-03561, Version 1.0 released April 24, 2001

1.6 Requirements Traceability

This document serves to satisfy the following requirements for the ELS portion of the ASPERA-3 science and housekeeping data. The first two entries are Functional Requirements (FR) and the last two entries are Input Interface Requirements (II).

| Requirement Identifier | Requirement Description | Source | Page Number |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|-------------|
| APAF-FR-02 | The APAF system shall process all ASPERA-3 science data into IDFS data sets. | Software Requirements Specification | 4 |
| APAF-FR-03 | The APAF system shall process the engineering and ancillary information necessary for calibration and science validation into IDFS data sets. | Software Requirements Specification | 4 |
| APAF-DS-II-2 | ASPERA-3 and Mars Express Orbit/Attitude Telemetry Data | Software Requirements Specification | 5 |
| APAF-DS-II-3 | Virtual Instrument Descriptions for the ASPERA-3 Experiment Data and the Mars Express Orbit/Attitude Data | Software Requirements Specification | 6 |

2.0 Packet Header for ELS Science and Housekeeping Data

The telemetry source packet, which is defined in the Mars Express Space / Ground Interface Control Document (SGICD), is comprised of a Source Packet Header (48 bits in length) followed by the Packet Data Field (which is a variable-length field). The Packet Data Field is comprised of a fixed-length data field header followed by the source data.

There are 2 different types of packets utilized to package the ELS science data and the ELS housekeeping parameters. These two packets are referred to throughout this document as the ELS Science Packet and the Main Unit Housekeeping Packet, respectively. The Main Unit Housekeeping Packet contains housekeeping data not only for ELS, but also for NPI, NPD, along with a few parameters for the Scanning Unit, for IMA and for the Main Unit itself.

Table 2-1 summarizes the information contained in the ELS Science and Main Unit Housekeeping Packets. The byte ordering is sequential within a packet; that is, data is simply laid down contiguously, starting from byte 0 up to byte N. Bytes 0-5 correspond to the 48-bit, fixed length Source Packet Header. Bytes 6-N corresponds to the variable-length Packet Data Field, which is comprised of a Data Field Header (Bytes 6-15) and Source Data (Bytes 16-N). A thick black line is utilized to subdivide Table 2-1 into three distinct blocks - Source Packet Header, Data Field Header and Source Data blocks. The notation **(SCI)** and **(HSKP)** will be used to delineate between values for a Science packet and a Housekeeping packet, respectively.

The following convention shall be used to identify each bit in an N-bit field (see Figure 1), which is opposite to the convention defined in the SGICD, Appendix A1.1:

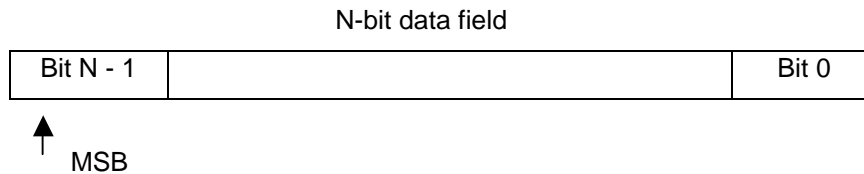


Figure 1 Bit Designation

The overall structure of the packet is the same for the ELS science and the Main Unit housekeeping data; the difference between the two is the value set for the **Packet Type** and **Packet Subtype** fields, the length of the **Source Data** field and the contents of the **Source Data** field.

| Table 2-1 ELS Source Data Packets | | | | |
|------------------------------------------|------|-------|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Telemetry Source Packet Field | Byte | Bit | Value (Decimal) | Description |
| Source Packet Version Number | 0-1 | 15-13 | 0 | Source Packet version number |
| Source Packet Type | 0-1 | 12 | 0 | Source Packet type |
| Data Field Header Flag | 0-1 | 11 | 1 | Flag indicating the presence or absence of a Data Field Header |
| Application Process ID | 0-1 | 10-0 | | ID which uniquely identifies the on board source of the packet. The most significant 7 bits (10-4) form the Process Id, which is the decimal value 61 for ASPERA. The least significant 4 bits (3-0) form the Packet Category. |
| Segmentation Flags | 2-3 | 15-14 | 3 | Segmentation (Grouping) Flag |
| Source Sequence Count | 2-3 | 13-0 | | Counter which corresponds to the order of release of packets by the source |
| Source Packet Length | 4-5 | | Based Upon the Type of Packet Being Processed and Set By the Main Unit | Number of Octets contained in Packet Data Field |
| SCET Time | 6-11 | | | Time that the acquisition of the data in the packet was initiated (CCSDS Unsegmented Time Code) |
| PUS Version | 12 | 7-5 | | Defines the routing/destination of the TM packets |
| Checksum Flag | 12 | 4 | 0 | Not used; always 0 for MEX |
| Spare | 12 | 3-0 | 0 | |
| Packet Type | 13 | | 20 (SCI) 3 (HSKP) | The type to which the telemetry source packet relates |
| Packet Subtype | 14 | | 3 (SCI) 25 (HSKP) | In conjunction with Packet Type, uniquely identifies the nature of the telemetry contained within the telemetry source packet |
| Pad | 15 | | | Additional routing information for the TM packets |
| Source Data | 16-N | | Variable | Start of the ELS Science Data or the Main Unit Housekeeping Data |

3.0 Notes for ELS Data

The reader of this document is referred to Appendix B for a detailed explanation of operational scenarios for the ELS instrument. The scenarios described affect how much science data is generated by ELS.

The ELS science data can be compressed in various ways in order to transmit the data from the satellite to the ground. The following table summarizes how each compression / summation method will be handled and translated into the IDFS data sets:

| Compression / Summation Method | IDFS action |
|---------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Time summation - the summation of multiple sweeps (N) onboard before transmission | The sweep will be written multiple times (N) in the data file, with a time tag that reflects the true accumulation time for each of the sweeps. A status flag will be used to report the number of sweeps that were summated (N) in order to normalize the value when retrieved by the IDFS data access software. |
| Energy summation - the summation of multiple steps (M) within a sweep onboard before transmission | The energy sweep will be handled much like what is done for time summation; that is, the sweep will always be expanded out to 128 steps, with the summed value being repeated M many times within the sweep. A status flag will be used to report the number of steps that were summated (M) in order to normalize the value when retrieved by the IDFS data access software. |
| Bit compression - reducing from 16-bit to 8-bit data | The data will always be written as 16-bit words in IDFS with the IDFS production code decompressing the data before storage. |

In addition to compression and summation, there is the issue of how to deal with the fact that the ELS instrument utilizes a dual range power supply. For each range, there can be a different number of energy steps returned and different reconstruction algorithms that are to be utilized. It has been decided that the easiest and most efficient way to handle this concern is to break up the data into separate virtual instruments according to the sweeping power supply range. There are some existing IDFS display tools that allow for the simultaneous display of data from multiple virtual instruments; therefore, the data from the two ranges can be visually displayed in unison.

It is imperative that an absolute time tag be incorporated into the packets so that correlative science analysis can be performed properly and accurately.

4.0 Introduction to IDFS

Within the IDFS paradigm, for each virtual instrument there are three associated files: (1) the Virtual Instrument Description File (VIDF), (2) the header file, and (3) the data file. A thorough, in-depth explanation of the contents of these 3 files can be found in the Instrument Description File System Definition document, which can be accessed from the <http://www.idfs.org> website. This document will describe the current entries in each file set. There are two types of entry definitions: alpha/numeric, which indicate a value with which the field is to be filled, and DATA, which indicates that the field is filled either with telemetry values or the value for the field is determined by telemetry values. In the VIDF file descriptions, fields which are left blank are unused by the virtual instrument. All alpha/numeric values are subject to review.

The VIDF is a complete description of the virtual instrument. The VIDF file is meant to be easily updated and to contain all of the data that may be periodically updated due to either refinement in the instrument calibration or due to the degradation within the instrument. There must be at least one VIDF file defined for each IDFS virtual instrument. If data within the VIDF changes with time, for example calibration coefficients, additional VIDF files can be defined. The VIDF file provides a general description of the measurements being stored in IDFS format, and contains the data reconstruction parameters that are needed in order to transform the raw data into physical units. In addition, the VIDF file provides information by which data from the data file can be extracted, such as:

- the field DATA_LEN holds the size of the data records contained in the data file
- the field MAX_NSS defines the maximum number of sensor sets defined in each data record
- the field TDW_LEN defines the word length (bits) for each sensor contained within each data record
- the field FILL contains the designated fill data value used in the IDFS data records to indicate missing or fill data

The header files contain data which, for the most part, is slowly varying in time and need not be repeated every data record. Each IDFS data record points to a header record that describes the state of the instrument at that particular point in time. The format of the header record is shown below in the form of a C data structure:

```
struct
{
    2-byte signed int    hdr_len;
    2-byte signed int    year;
    2-byte signed int    day;
    1-byte signed char   time_units;
    1-byte unsigned char i_mode;
    4-byte signed int    data_accum;
    4-byte signed int    data_lat;
```

```

4-byte signed int    swp_reset;
4-byte signed int    sen_reset;
2-byte signed int    n_sen;
2-byte unsigned int  n_sample;
2-byte signed int    scan_index[1 or n_sample];
2-byte signed int    sensor_index [n_sen];
1-byte unsigned char d_qual [n_sen];
1-byte unsigned char mode_index [i_mode];
};

```

D_qual is an index into an array of data quality flags defined in the VIDF file. At a minimum, each VIDF file should define 2 levels of data quality, good and bad data. The **data_accum**, **time_units**, and **data_lat** fields, taken together, define the total time between successive accumulations. Note that for many scalar data sets, the accumulation time (**data_accum**) is set to zero to indicate that the accumulation is instantaneous. In these cases, the **data_lat** field is used to give the time between successive measurements. For those virtual instruments that contain vector sensors, the **swp_reset** field is needed to define the dead time that is needed to reset themselves between successive sweeps. For many particle instruments, this is equivalent to the fly back time.

The vast majority of all of the telemetered data is stored within the data file, which contains the most rapidly varying data. The data records contain the base time tag for the data and raw, unprocessed binary data. Unlike the header record, the data record does not vary in size. The size is specified in the VIDF file. The format of the data record is shown below in the form of a C data structure:

```

struct
{
4-byte signed int    dr_time;
4-byte signed int    spin;
4-byte signed int    sun_sen;
4-byte signed int    hdr_off[max_nss];
4-byte signed int    nss;
1-byte unsigned char data_array[data_size];
};

```

Note that **data_array** is generically assigned as an unsigned character (8 bits). The data may be stored within the field with a base length of 8, 16 or 32 bits. The storage boundary used for individual data within a particular data file is determined from the VIDF file and is used by the IDFS data access software to correctly unpack the data.

Data storage in the IDFS data record is organized along the concept of sensor (primary) data, calibration (secondary) data and sensor sets. Sensor data is the basic, primary measurement and is placed in **data_array** field first. Calibration data is ancillary data that is necessary to interpret the primary data (e.g., automatic gain correction values). Not all virtual instruments have calibration data defined. Calibration data is placed in the **data_array** field after all

sensor data has been written. These two data matrices (sensor and calibration) taken collectively are what is referred to as an IDFS sensor set.

The **data_array** field is a one-dimensional array in which the data is laid down sensor set by sensor set. Within the sensor set, all of the ELS sensor data is laid down sweep by sweep, followed by any calibration data. The following illustration shows the formation of a typical data array. A sweep for any sensor is laid down sequentially. If we label this segment of data as SW_n , then $SW_n =$

| | | | | | |
|---|---|---|-----|-----|------------|
| 0 | 1 | 2 | ... | ... | n_sample-1 |
|---|---|---|-----|-----|------------|

Figure 2 IDFS sensor data array

where n_sample (see Figure 2) is the sweep length.

Corresponding to each sweep may be a set of ancillary (calibration) values such as automatic gain corrections, attenuation values, etc. If we designate these values as $C_n[i]$, then $C_n[i] =$

| | | | | | |
|---|---|---|-----|-----|------------|
| 0 | 1 | 2 | ... | ... | cset_len-1 |
|---|---|---|-----|-----|------------|

Figure 3 IDFS calibration data array

where cset_len (see Figure 3) is the number of elements in the calibration set, i is used to indicate which correction set and n is used to show the association between SW_n and $C_n[i]$.

A simple sensor set consisting of three sensors each, with two calibration sets associated with each sweep is laid down as:

$$[SW_0][SW_1][SW_2][C_0[0]][C_0[1]][C_1[0]][C_1[1]][C_2[0]][C_2[1]]$$

5.0 Breakdown of ELS Data into Virtual Instruments

The ELS Data will be broken down into the following six virtual instruments:

1. **ELSENG8** - ELS 8-Bit Engineering Monitor Data
2. **ELSENGS** - ELS Engineering Status Data
3. **ELSSWPL** - ELS Sweep Low Data
4. **ELSSWPH** - ELS Sweep High Data
5. **ELSSCIL** - ELS Science Low Data
6. **ELSSCIH** - ELS Science High Data

The following sections will describe the data that comprises each of the six virtual instruments defined for ELS data. The data was subdivided into the six virtual instruments based upon data category (Engineering Data vs. Science Data), word size and the portion of the ELS sweep range the data encompassed (low range vs. high range).

In order to derive the values for the **spin** and **sun_sen** fields within the IDFS data records, scanner information that is contained within the telemetry packets is utilized. Due to the actual mechanical operations of the scanner, it is possible that the absolute scanner position could be unattainable and the value returned in telemetry could be invalid. There is no way to flag this data as invalid when examining a single telemetry packet. The IDFS production code should be designed to have alternate ways in which to retrieve the scanner information, perhaps based upon the value of an argument being passed to the production routine. At the present time, alternative methods of determining scanner information have not been defined. It is envisioned that one alternative method may be reading from a pre-defined file which will contain either the scanner information to use for the calculations or will contain pre-calculated values for the **spin** and **sun_sen** fields. For the time being, the value of the argument should be set so that the scanner information is read from the telemetry packet being processed.

6.0 Definition for Virtual Instrument **ELSENG8**

The ELS-specific 8-bit monitor and associated reference values returned in the ELS Science packet that is flagged as containing ELS engineering information (Data subtype = 0) form the virtual instrument **ELSENG8**. The first three characters in the acronym are intended to identify the parent instrument (ELS), the next three characters (ENG) indicate that the data is engineering data and the last character (8) identifies the data as 8-bit data.

The IDFS sensor definitions for the virtual instrument are defined in Table 6-1 below. The status flag definitions for the virtual instrument are defined in Table 6-2. The VIDF file for this virtual instrument can be found in Appendix A.

The Main Unit produces different types of science packets, based upon the data source. The science packet type identifier can be found in the **Data type** parameter, which is defined in byte 19, bits 4-7 in all science data packets generated by the Main Unit, except for the IMA packets. This parameter identifies the instrument (data source) to which the packet is related. For the ELS engineering information data packet, the value for the **Data type** parameter is 1 and the packets are referred to as ELS telemetry packets.

For each instrument defined, the Main Unit can produce multiple packet subtypes. For the ELS telemetry packets, the science packet subtype identifier can be found in the **ELS packet subtype** parameter, which is defined in byte 19, bits 0-1. For the ELS engineering information data packet, the value for the **ELS packet subtype** parameter is 0. In Table 6-1 and Table 6-2, the heading **Data Subtype** refers to the packet subtype.

Table 6-1 ELSENG8 Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|---------------------------|----------------------|------------------|---------------------|---------------------------|-----------------------------|
| 0 | -5V Screen Grid Reference | SCI | 1 | 0 | ELS Screen grid reference | 36 |
| 1 | -5V Screen Grid Monitor | SCI | 1 | 0 | ELS Screen grid monitor | 37 |
| 2 | MCP Bias Reference | SCI | 1 | 0 | ELS MCP reference | 34 |
| 3 | MCP Bias Monitor | SCI | 1 | 0 | ELS MCP monitor | 35 |
| 4 | ELS Temperature Monitor | SCI | 1 | 0 | ELS temperature | 33 |

Table 6-2 ELSENG8 Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|--------------------|-------------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | SW Version – Upper Byte | SCI | 1 | 0 | SW Version | 16 |
| 1 | SW Version – Lower Byte | SCI | 1 | 0 | SW Version | 17 |
| 2 | Software Mode | HSKP | - | - | sw_mode | 106 |

6.1 Notes for ELSENG8 virtual instrument

The data contained within this virtual instrument is scalar in nature and therefore, the virtual instrument itself is defined as a scalar instrument by setting the **smp_id** field in the VIDF file to 2.

The values are taken once per scanner cycle and the assumption is that these values are grabbed instantaneously, so the values in the header record for **data_accum** and **data_lat** are set accordingly in Section 6.2.

6.1.1 ELS Temperature Monitor

This write-up assumes an 8-bit word and that the voltage ranges from 0V to 5V.

For the AD590 temperature transducer:

- Absolute current device
- Range of operation is -55°C to $+150^{\circ}\text{C}$
- Linear current output at $1\ \mu\text{A}$ per $^{\circ}\text{K}$
- Normal current output $298.2\ \mu\text{A}$ at 25°C

Current flows through a $12.1\ \text{K}\Omega$ resistor in ELS. The current is digitized by the DPU such that $0\text{V} = 0\text{x}00$ and $5\text{V} = 0\text{x}FF$ in telemetry. So,

$$0\text{V} = V_L$$

$$5\text{V} = V_H$$

$$\text{Voltage} = M_v * \text{TMON} + b_v$$

where $\text{TMON} = 0\text{x}00$ at V_L and $\text{TMON} = 0\text{x}FF$ at V_H , which allows a solution for the slope (M_v) and intercept (b_v) of the monitor value.

Solving for b_v , we get

$$V_L = M_v * 0\text{x}00 + b_v$$

$$\therefore b_v = V_L$$

now, the slope can be found

$$V_H = M_v * 0\text{x}FF + V_L$$

$$M_v = \frac{(V_H - V_L)}{0\text{x}FF}$$

$$\therefore M_v = \frac{(5 - 0)}{255} = \frac{1}{51}$$

Using the current-voltage relation $V = IR$ (where V is the voltage, I is the current and R is the resistance), the current is

$$I = \frac{V}{R}$$

where the current flow is through the resistor

$$R = 12.1 \times 10^3 \text{ ohm}$$

and the voltage that is produced by the device is

$$V = \frac{(V_H - V_L)}{0xFF} * TMON + V_L$$

For the current, we know that

$$25^\circ\text{C} = M_I (298.2 \times 10^{-6} \text{ A}) + b_I$$

Since the device is linear, we also know that

$$26^\circ\text{C} = M_I (299.2 \times 10^{-6} \text{ A}) + b_I$$

subtracting, we get

$$1^\circ\text{C} = M_I (1 \times 10^{-6} \text{ A})$$

or

$$M_I = \frac{1^\circ\text{C}}{1 \times 10^{-6} \text{ A}}$$

Going back to the original equation,

$$25^\circ\text{C} = \frac{1^\circ\text{C}}{1 \times 10^{-6} \text{ A}} (298.2 \times 10^{-6} \text{ A}) + b_I$$

$$\therefore b_I = 25^\circ\text{C} - 298.2^\circ\text{C}$$

$$b_I = -273.2^\circ\text{C}$$

and temperature

$$T = \frac{1^\circ\text{C}}{1 \times 10^{-6} \text{ A}} * I - 273.2^\circ\text{C}$$

Putting the equations together

$$\text{Temperature} = \frac{1^\circ\text{C}}{1 \times 10^{-6} \text{ A}} * \frac{1}{12.1 \times 10^3 \text{ ohm}} * \left[\frac{(V_H - V_L)}{0xFF} * TMON + V_L \right] - 273.2^\circ\text{C}$$

which is

$$\text{Temperature}[^\circ\text{C}] = 1.620483 * TMON - 273.2$$

6.1.2 ELS Bias Monitor

The ELS Bias Monitor represents the value of the voltage across the MCP bias network which was actually achieved by ELS. This value is represented three different ways: by an 8-bit telemetry number, by a control voltage value, and by the voltage generated across the MCP network.

The ELS Bias Monitor represents a control voltage which ranges from 0V to 4.5V. The voltage is digitized by an Analog to Digital converter to be between 0x00 and 0xff. The converter is linearly divided, so that 0x00 represents 0.000V and 0xff represents 4.500V. Thus, the ELS Bias Monitor is converted to an analog control voltage through the formula:

$$\text{Bias Monitor [V]} = s * \text{BMON}$$

where $s = 4.5/0\text{xff}$ and BMON is the value returned in telemetry.

To convert the Bias Monitor value to the voltage across the MCP network, a Bias Monitor voltage of 0V corresponds to 0V on the bias network. A bias reference of 4.5V corresponds to the full voltage of the ELS bias supply, which is 3000V. So, the ELS Bias Monitor is converted to a voltage across the MCP network through the formula:

$$\text{Bias Monitor [V]} = m * \text{BMON}$$

where $m = 3000/0\text{xff}$ and BMON is the value returned in telemetry.

6.1.3 ELS Bias Reference

The ELS Bias Reference represents the value of the voltage across the MCP bias network to which ELS was commanded. This value is represented three different ways: by an 8-bit telemetry number, by a control voltage value, and by the voltage generated across the MCP network.

The ELS Bias Reference represents a control voltage which ranges from 0V to 5.0V. The voltage is digitized by an Analog to Digital converter to be between 0x00 and 0xff. The converter is linearly divided, so that 0x00 represents 0.000V and 0xff represents 5.000V. Thus, the ELS Bias Reference is converted to an analog control voltage through the formula:

$$\text{Bias Reference [V]} = s * \text{BMON}$$

where $s = 5.0/0\text{xff}$ and BMON is the value returned in telemetry.

To convert the Bias Reference value to the voltage across the MCP network, a Bias Reference voltage of 0V corresponds to 0V on the bias network. A bias reference of 5.0V corresponds to the full voltage of the ELS bias supply, which is 3000V. So, the ELS Bias Reference is converted to a voltage across the MCP network through the formula:

$$\text{Bias Reference [V]} = m * \text{BMON}$$

where $m = 3000/0\text{xff}$ and BMON is the value returned in telemetry.

6.1.4 ELS -5V Screen Grid Reference

The ELS Screen Grid Reference represents the value of the negative voltage placed on the ELS screen grid. The purpose of the screen grid is to reflect low-energy electrons before they reach the MCP. Electrons with energies below the screen grid potential are reflected and do not reach the MCP. The reference value represents the voltage to which the screen grid is commanded. Its value can be represented in two different ways: by an 8-bit telemetry number and by the voltage generated on the ELS screen grid.

The ELS Screen Grid Reference represents a voltage which ranges from 0V to -5.0V. The voltage is determined by a Digital command and converted to an Analog voltage. The command value is an 8-bit word between 0x00 and 0xff. The Digital to Analog converter changes this voltage into a 0 to -5.0V value to be placed on the screen grid. The converter is linearly divided, so that 0x00 represents 0.000V and 0xff represents -5.000V. Thus, the ELS Screen Grid Reference represents the 8-bit code that is sent in the command.

The ELS Screen Grid Reference is converted from the 8-bit code represented by the telemetry value (or commanded value) into the screen grid voltage by conversion to the analog voltage value through the formula:

$$\text{Screen Grid Reference [V]} = s * \text{BMON}$$

where $s = -5.0/0\text{xff}$ and BMON is the telemetry 8-bit value.

6.1.5 ELS -5V Screen Grid Monitor

The ELS Screen Grid Monitor represents the value of the negative voltage that was actually placed on the ELS screen grid. The purpose of the screen grid is to reflect low-energy electrons before they reach the MCP. Electrons with energies below the screen grid potential are reflected and do not reach the MCP. The monitor value represents the voltage actually achieved on the screen grid, not the command. Its value can be represented in two different ways: by an 8-bit telemetry number and by the voltage on the ELS screen grid. The bit length of the telemetered ELS Screen Grid Monitor is 8 bits.

In the Main Unit of ASPERA-3, the voltage from the ELS Screen Grid is inverted so that negative voltages have a positive sign and positive voltages have a negative sign. The ELS Screen Grid inverted voltage is then passed to an Analog to Digital 14-bit signed converter such that bit 14 is the sign of the voltage measured and the device is a linear measure of voltage. Thus, 0x000 is 0.000 V and 0x1ff is 5.000 V with 0x200 representing the voltage sign for the 14-bit word. The maximum voltage which is measured should be -5.000V (inverted to be 5.000V).

Since the 14-bit A/D device within the Main Unit is linear, the device represents $5.0/0\text{x1ff} = 5/511$ volts per digital value. For telemetry, the lower 5 bits are shifted away and only the next 8

bits are telemetered (the sign bit is discarded: $1 + 8 + 5 = 14$ bits). The minimum value of the telemetry word is zero when values are less than 5 bits, but when you are at 6 bits, the minimum voltage is $32 * 5/511 = 0.313112V$. So in telemetry, -0.313112 v represents $0x01$ and -5.000 v represents $0xff$.

To reconstruct the voltage from the ELS Screen Grid Monitor 8-bit telemetry word, use the formula

$$-0.018452317 * \text{BMON} - 0.294659229 = \text{SG volts}$$

where BMON is the screen grid monitor telemetry value and SG is the voltage value of the monitor. Thus, if the monitor TM value for the screen grid is a $0x5b$ (91 decimal) for example, the represented voltage is:

$$-0.018452317 * 91 - 0.294659229 = -1.974 \text{ volts}$$

When you use this formula, a $0x00$ is reconstructed as -0.294659229 volts and represents an upper limit of the screen grid voltage. Numbers between $0x01$ and $0xff$ are represented correctly to the accuracy of -0.018452317 Volts.

6.2 ELSENG8 Header Record Format

The nominal values assigned to the header record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 6-3 ELSENG8 Header Record Definition

| Field | Description | Value |
|--------------|---------------------------------------------------|-------|
| hdr_len | header record size in bytes | DATA |
| year | year as YYYY | DATA |
| day | day of year as DDD | DATA |
| time_units | time base for data_accum | 0 |
| i_mode | no. of status flags returned in mode_index | 3 |
| data_accum | sample accumulation time | 0 |
| data_lat | sample dead time (μ sec) | 31250 |
| swp_reset | sweep dead time (μ sec) | 0 |
| sen_reset | sensor set dead time (μ sec) | 0 |
| n_sen | no. of sensors returned in sensor set | 5 |
| n_sample | no. of data samples per sensor | 1 |
| scan_index | index into scan dependent tables | 0 |
| sensor_index | VIDF sensors returned in sensor set | DATA |
| d_qual | data quality flag per sensor | DATA |
| mode_index | status flag values | DATA |

6.2.1 Setting of *n_sample* header record element

For scalar sensors, **n_sample** is simply the number of successive measurements which are stacked in the sensor set. For this virtual instrument, no stacking is made; therefore, **n_sample** is set to 1.

6.2.2 Setting of *scan_index* header record element

Scan_index is an array of **n_sample** values. The **scan_index** values indicate the sweep step number associated with each of the data samples returned per sensor. For scalar sensors, this field is an array of one. In general, the value for a scalar virtual instrument has no meaning and should be set to zero.

scan_index [0] 0

6.2.3 Setting of *sensor_index* header record element

Sensor_index is an array of **n_sen** values. Since all IDFS sensors will be returned per data record, **n_sen** is set to 5 and the values for **sensor_index** should be set accordingly:

| | |
|------------------|---|
| sensor_index [0] | 0 |
| sensor_index [1] | 1 |
| sensor_index [2] | 2 |
| sensor_index [3] | 3 |
| sensor_index [4] | 4 |

6.2.4 Setting of *d_qual* header record element

D_qual is an array of **n_sen** values. According to the VIDF file outlined to date, there are currently 5 states defined for data quality, as shown in the table below:

| Value | Meaning | Criteria for Setting of Value |
|-------|-------------------|-------------------------------------------------------------------------------------------------|
| 0 | Good Data | default setting |
| 1 | Questionable Data | +30V monitor value is not within tolerance |
| 2 | Invalid Data | +12V, -12V, +5V or -5V monitor value is not within tolerance or the high voltage is not enabled |
| 3 | Bad Data | +30V monitor value and (+12V, -12V, +5V or -5V monitor value) are not within tolerance |
| 4 | Unknown State | Time related Main Unit Housekeeping packet is not available |

The setting of the data quality value to Questionable is dependent upon the state of the high voltage, which is specified in the single-bit parameter **els_enable_hv** (byte 30, bit 2) found within the Main Unit Housekeeping packet. If the value is set to 0 (disabled), the data quality value should be set to 1 (Questionable Data). Since all IDFS sensors will be returned per data record, **n_sen** is set to 5 and the values for **d_qual** should be set accordingly under nominal conditions:

| | |
|------------|---|
| d_qual [0] | 0 |
| d_qual [1] | 0 |
| d_qual [2] | 0 |

d_qual [3] 0
 d_qual [4] 0

6.2.5 Setting of *mode_index* header record element

Mode_index is an array of **i_mode** values. Table 6-2 indicates which telemetry packet fields are to be used to fill this array. The first two status bytes do not reflect the state of the instrument. They are simply used to record the version number of the Main Unit software used to generate the data packets. For status byte 2, a VIDF table will be defined to hold the values that are associated with each possible state value. The following is a brief explanation of the possible values and the meanings for status byte 2:

| Status Byte | 2 Software Mode |
|--------------------------|--------------------------------------------------------------|
| State Definitions | 0 = Off 1 = Booting 2 = Safe 3 = Prom 4 = Normal |

6.3 ELSENG8 Data Record Format

The nominal values assigned to the data record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 6-4 ELSENG8 Data Record Definition

| Field | Description | Value |
|------------|------------------------------------|-------|
| dr_time | initial data time (msec) | DATA |
| spin | azimuthal rate of rotation (msec) | DATA |
| sun_sen | last 0° crossing (msec) | DATA |
| hdr_off | header record offsets in bytes | DATA |
| nss | no. of sensor sets per data record | 1 |
| data_array | data values | DATA |

6.3.1 Setting of *dr_time* data record element

The time tag that will be utilized is the time tag defined in bytes 20-25 (SCET Time) in the ELS Science packet. This value is a 6-byte quantity that represents a CCSDS Unsegmented Time Code.

The finest resolution supported by the IDFS data access software is down to the nanosecond. However, **dr_time** holds the relative beginning time of day in milliseconds for the first data element of the first sensor set in the data record. In order to define the time tag down to the nanosecond precision, the VIDF field **nano_defined** must be set to indicate that a nanosecond time adjustment value is contained within the **data_array** matrix.

6.3.2 Setting of *spin* data record element

There are two parameters which must be considered in order to determine the value to store within the **spin** field in the IDFS data record. These two parameters are spin rate and spin direction. The spin rate is determined by the scanner speed. The scanner speed is reported in the **Scanner speed** parameter, which is byte 30, bits 0-1 in the ELS Science packet. The possible scanner speeds are 0 (non spinning, value = 0), 32 second scan (value = 1), 64 second scan (value = 2), and 128 second scan (value = 3). A scan consists of a turning of 180 degrees. The scanner moves forward and backward through an angle of 180 degrees during normal operation.

The spin direction is determined by the scanner direction. The scanner direction is reported in the **Scanner direction** parameter, which is byte 30, bit 2 in the ELS Science packet. The value for scanner direction will either be 0 to indicate a 0-180 direction or 1 to indicate a 180-0 direction.

The data value of **spin** is given in milliseconds per revolution (a revolution is 360 degrees). When the scanner direction is 0 – 180, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | 64 | 64000 |
| 64 second scan | 128 | 128000 |
| 128 second scan | 256 | 256000 |

When the scanner direction is 180 - 0, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | -64 | -64000 |
| 64 second scan | -128 | -128000 |
| 128 second scan | -256 | -256000 |

6.3.3 Setting of *sun_sen* data record element

There are three parameters which must be considered in order to determine the value to store within the **sun_sen** field in the IDFS data record. These three parameters are scanner speed, scanner direction and scanner position. The scanner position is reported in the **Scanner position** parameter, which is byte 31 in the ELS Science packet. The scanner position is reset to 0 when the scanner is pointing in the direction of 0 degrees or less. There are 0.806 degrees per count under normal conditions. This means that at 180 degrees, the scanner position count is 223.

If the scanner speed is 0 (non spinning), then the scanner position marks where the scanner is pointing. Let's call this position **P**. The **sun_sen** value written in the data record is given as degrees times 100 for the non spinning case. Thus, the value written in the **sun_sen** field of the data record is $180 * 100 * P / 223$.

If the scanner speed is not 0, then the scanner direction will need to be examined in order to project where the 0 offset (or 0 degree) location was or will be located with respect to time.

If the scanner direction is from 0-180, the scanner crossed the 0 degree marker some time in the past. For the value of the scanner position **P**, the value was

$$\text{spin} / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **subtracted** from the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the subtraction is negative, this means that the **sun_sen** time crossed the day boundary. To change this to the next 0 degree marker crossing, **add** the value

$$\text{spin} / 2 * (2 - \mathbf{P} / 223)$$

to the SCET time.

If the scanner direction is from 180-0, then the scanner will cross the 0 degree marker some time in the future. For the value of scanner position **P**, the value will be

$$\text{abs}(\text{spin}) / 2 * \mathbf{P} / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **added** to the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the addition is greater than or equal to 86400 seconds, this means that the **sun_sen** time crossed the day boundary. To change this to the previous 0 degree marker crossing, **subtract** the value

$$\text{abs}(\text{spin}) / 2 * (2 - \mathbf{P} / 223)$$

from the SCET time.

6.3.4 Setting of *data_array* data record element

The first four bytes contained within the **data_array** matrix is a nanosecond time adjustment factor that will be used by the IDFS data access software when computing the time tags associated with the data (refer to **dr_time**). The nanosecond time adjustment factor is interpreted as a signed, 4-byte quantity and therefore, can reach a maximum of 2,147,483,647. However, since this value contains the resolution between nanosecond and millisecond precision, the value should be no larger than 999,999. The remainder of the **data_array** for **ELSENG8** should be laid down according to the layout provided below:

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Comments |
|----------------|-------------|---------------|-----------|--------------|----------------------|----------|
| data_array [4] | 0 | SCI | 1 | 0 | 36 | 1 value |
| data_array [5] | 1 | SCI | 1 | 0 | 37 | 1 value |
| data_array [6] | 2 | SCI | 1 | 0 | 34 | 1 value |

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Comments |
|-----------------------|--------------------|----------------------|------------------|---------------------|-----------------------------|-----------------|
| data_array [7] | 3 | SCI | 1 | 0 | 35 | 1 value |
| data_array [8] | 4 | SCI | 1 | 0 | 33 | 1 value |

7.0 Definition for Virtual Instrument ELSENGs

The ELS-specific instrument status monitors returned in the Main Unit Housekeeping packet form the virtual instrument **ELSENGs**. The first three characters in the acronym are intended to identify the parent instrument (ELS), the next three characters (ENG) indicate that the data is engineering data and the last character (S) indicates that the virtual instrument contains status parameters.

The IDFS sensor definitions for the virtual instrument are defined in Table 7-1 below. The status flag definitions for the virtual instrument are defined in Table 7-2. The VIDF file for this virtual instrument can be found in Appendix A.

Table 7-1 ELSENGs Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Packet Field Name | Packet Byte Position | Packet Bit Position |
|--------------------|---------------|---------------|-------------------|----------------------|---------------------|
| 0 | ELS Enable HV | HSKP | els_enable_hv | 30 | 2 |
| 1 | +30V Enable | HSKP | hk_v_plus_30v | 35 | n/a |
| 2 | -12V Enable | HSKP | hk_v_minus_12v | 37 | n/a |
| 3 | +12V Enable | HSKP | hk_v_plus_12v | 34 | n/a |
| 4 | -5V Enable | HSKP | hk_v_minus_5v | 38 | n/a |
| 5 | +5V Enable | HSKP | hk_v_plus_5v | 36 | n/a |

Table 7-2 ELSENGs Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Packet Field Name | Packet Byte Position |
|-------------|-------------------------|---------------|-------------------|----------------------|
| 0 | SW Version – Upper Byte | HSKP | sw_version | 24 |
| 1 | SW Version – Lower Byte | HSKP | sw_version | 25 |
| 2 | Software Mode | HSKP | sw_mode | 106 |

The **ELSENGs** virtual instrument is defined to hold one-bit status values that are of interest to aid in the understanding of the current operational status of the ELS instrument. However, only one of the IDFS sensors defined above (sensor 0) is a true status bit being returned in telemetry. The other IDFS sensors (sensors 1 through 5) must determine their single-bit values based upon the monitor readings returned in telemetry as identified in the **Packet Field Name** column of the table above.

7.1 Notes for ELSENGS virtual instrument

The data contained within this virtual instrument is scalar in nature and therefore, the virtual instrument itself is defined as a scalar instrument by setting the **smp_id** field in the VIDF file to 2.

The values are taken once per scanner cycle and the assumption is that these values are grabbed instantaneously, so the values in the header record for **data_accum** and **data_lat** are set accordingly in Section 7.2.

The **ELSENGS** virtual instrument is defined to hold one-bit status values that are of interest to aid in the understanding of the current operational status of the ELS instrument. However, only one of the IDFS sensors defined (sensor 0) is a true status bit being returned in telemetry. The other IDFS sensors (sensors 1 through 5) must determine their single-bit values based upon monitor readings returned in telemetry. The following information defines how the status bit settings are to be determined by the IDFS production code for IDFS sensors 1 through 5:

| IDFS Sensor Name | Telemetry Monitor | Determination for Bit Setting |
|------------------|-------------------|----------------------------------------------------------|
| +30V Enable | hk_v_plus_30v | if $+27V \leq \text{monitor} \leq +33V$ value = 1 |
| | | Otherwise, value = 0 |
| -12V Enable | hk_v_minus_12v | if $-13.2V \leq \text{monitor} \leq -10.8V$ value = 1 |
| | | Otherwise, value = 0 |
| +12V Enable | hk_v_plus_12v | if $+11.4V \leq \text{monitor} \leq +12.6V$ value = 1 |
| | | Otherwise, value = 0 |
| -5V Enable | hk_v_minus_5v | if $-5.25V \leq \text{monitor} \leq -4.75V$ value = 1 |
| | | Otherwise, value = 0 |
| +5V Enable | hk_v_plus_5v | if $+4.75V \leq \text{monitor} \leq +5.25V$ value = 1 |
| | | Otherwise, value = 0 |

7.2 ELSENGS Header Record Format

The nominal values assigned to the header record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 7-3 ELSENGS Header Record Definition

| Field | Description | Value |
|--------------|---------------------------------------------------|-------|
| hdr_len | header record size in bytes | DATA |
| year | year as YYYY | DATA |
| day | day of year as DDD | DATA |
| time_units | time base for data_accum | 0 |
| i_mode | no. of status flags returned in mode_index | 3 |
| data_accum | sample accumulation time | 0 |
| data_lat | sample dead time (μ sec) | 31250 |
| swp_reset | sweep dead time (μ sec) | 0 |
| sen_reset | sensor set dead time (μ sec) | 0 |
| n_sen | no. of sensors returned in sensor set | 6 |
| n_sample | no. of data samples per sensor | 1 |
| scan_index | index into scan dependent tables | DATA |
| sensor_index | VIDF sensors returned in sensor set | DATA |
| d_qual | data quality flag per sensor | DATA |
| mode_index | status flag values | DATA |

7.2.1 Setting of *n_sample* header record element

For scalar sensors, **n_sample** is simply the number of successive measurements which are stacked in the sensor set. For this virtual instrument, no stacking is made; therefore, **n_sample** is set to 1.

7.2.2 Setting of *scan_index* header record element

Scan_index is an array of **n_sample** values. The **scan_index** values indicate the sweep step number associated with each of the data samples returned per sensor. For scalar sensors, this field is an array of one. In general, the value for a scalar virtual instrument has no meaning and should be set to zero.

scan_index [0] 0

7.2.3 Setting of *sensor_index* header record element

Sensor_index is an array of **n_sen** values. Since all IDFS sensors will be returned per data record, **n_sen** is set to 6 and the values for **sensor_index** should be set accordingly:

```

sensor_index [0]    0
sensor_index [1]    1
sensor_index [2]    2
sensor_index [3]    3
sensor_index [4]    4
sensor_index [5]    5

```

7.2.4 Setting of *d_qual* header record element

D_qual is an array of **n_sen** values. According to the VIDF file outlined to date, there are currently 4 states defined for data quality, as shown in the table below:

| Value | Meaning | Criteria for Setting of Value |
|-------|-------------------|----------------------------------------------------------------------------------------|
| 0 | Good Data | default setting |
| 1 | Questionable Data | +30V monitor value is not within tolerance |
| 2 | Invalid Data | +12V, -12V, +5V or -5V monitor value is not within tolerance |
| 3 | Bad Data | +30V monitor value and (+12V, -12V, +5V or -5V monitor value) are not within tolerance |

Since all IDFS sensors will be returned per data record, **n_sen** is set to 6 and the values for **d_qual** should be set accordingly under nominal conditions:

```

d_qual [0]    0
d_qual [1]    0
d_qual [2]    0
d_qual [3]    0
d_qual [4]    0
d_qual [5]    0

```

7.2.5 Setting of *mode_index* header record element

Mode_index is an array of **i_mode** values. Table 7-2 indicates which telemetry packet fields are to be used to fill this array. The first two status bytes do not reflect the state of the instrument. They are simply used to record the version number of the Main Unit software used to generate the data packets. For status byte 2, a VIDF table will be defined to hold the values that are associated with each possible state value. The following is a brief explanation of the possible values and the meanings for status byte 2:

| Status Byte | 2 Software Mode |
|--------------------------|--------------------------------------------------------------|
| State Definitions | 0 = Off 1 = Booting 2 = Safe 3 = Prom 4 = Normal |

7.3 ELSENGs Data Record Format

The nominal values assigned to the data record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 7-4 ELSENGs Data Record Definition

| Field | Description | Value |
|--------------|------------------------------------|--------------|
| dr_time | initial data time (msec) | DATA |
| spin | azimuthal rate of rotation (msec) | DATA |
| sun_sen | last 0° crossing (msec) | DATA |
| hdr_off | header record offsets in bytes | DATA |
| nss | no. of sensor sets per data record | 1 |
| data_array | data values | DATA |

7.3.1 Setting of *dr_time* data record element

The time tag that will be utilized is the SCET time tag that is provided in the Data Field Header portion of the variable-length Packet Data Field section of the Main Unit Housekeeping packet. This value is a 6-byte quantity that represents a CCSDS Unsegmented Time Code.

The finest resolution supported by the IDFS data access software is down to the nanosecond. However, **dr_time** holds the relative beginning time of day in milliseconds for the first data element of the first sensor set in the data record. In order to define the time tag down to the nanosecond precision, the VIDF field **nano_defined** must be set to indicate that a nanosecond time adjustment value is contained within the **data_array** matrix.

7.3.2 Setting of *spin* data record element

There are two parameters which must be considered in order to determine the value to store within the **spin** field in the IDFS data record. These two parameters are spin rate and spin direction. The spin rate is determined by the scanner speed. The scanner speed is reported in the **scanner_speed** parameter, which is byte 99, bits 0-1 in the Main Unit housekeeping packet. The possible scanner speeds are 0 (non spinning, value = 0), 32 second scan (value = 1), 64 second scan (value = 2), and 128 second scan (value = 3). A scan consists of a turning of 180 degrees. The scanner moves forward and backward through an angle of 180 degrees during normal operation.

The spin direction is determined by the scanner direction. If the parameter **scanner_setup_mode** (Byte 99, bit 4) is set to 0 (normal mode), the scanner direction is reported in the **scanner_status_direction** parameter, which is byte 98, bit 4 in the Main Unit housekeeping packet. If **scanner_setup_mode** is set to 1 (manual mode), the

scanner direction is reported in the **scanner_setup_direction** parameter, which is byte 99, bit 3 in the Main Unit housekeeping packet. In either case, the value for scanner direction will either be 0 to indicate a 0-180 direction or 1 to indicate a 180-0 direction.

The data value of **spin** is given in milliseconds per revolution (a revolution is 360 degrees). When the scanner direction is 0 – 180, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | 64 | 64000 |
| 64 second scan | 128 | 128000 |
| 128 second scan | 256 | 256000 |

When the scanner direction is 180 - 0, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | -64 | -64000 |
| 64 second scan | -128 | -128000 |
| 128 second scan | -256 | -256000 |

7.3.3 Setting of *sun_sen* data record element

There are three parameters which must be considered in order to determine the value to store within the **sun_sen** field in the IDFS data record. These three parameters are scanner speed, scanner direction and scanner position. The scanner position is reported in the **scanner_position** parameter, which is byte 105 in the Main Unit housekeeping packet. The scanner position is reset to 0 when the scanner is pointing in the direction of 0 degrees or less. There are 0.806 degrees per count under normal conditions. This means that at 180 degrees, the scanner position count is 223.

If the scanner speed is 0 (non spinning), then the scanner position marks where the scanner is pointing. Let's call this position **P**. The **sun_sen** value written in the data record is given as degrees times 100 for the non spinning case. Thus, the value written in the **sun_sen** field of the data record is $180 * 100 * P / 223$.

If the scanner speed is not 0, then the scanner direction will need to be examined in order to project where the 0 offset (or 0 degree) location was or will be located with respect to time.

If the scanner direction is from 0-180, the scanner crossed the 0 degree marker some time in the past. For the value of the scanner position **P**, the value was

$$\text{spin} / 2 * \text{P} / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **subtracted** from the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the subtraction is negative, this means that the **sun_sen** time crossed the day boundary. To change this to the next 0 degree marker crossing, **add** the value

$$\text{spin} / 2 * (2 - \text{P} / 223)$$

to the SCET time.

If the scanner direction is from 180-0, then the scanner will cross the 0 degree marker some time in the future. For the value of scanner position **P**, the value will be

$$\text{abs}(\text{spin}) / 2 * \text{P} / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **added** to the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the addition is greater than or equal to 86400 seconds, this means that the **sun_sen** time crossed the day boundary. To change this to the previous 0 degree marker crossing, **subtract** the value

$$\text{abs}(\text{spin}) / 2 * (2 - \text{P} / 223)$$

from the SCET time.

7.3.4 Setting of *data_array* data record element

The first four bytes contained within the **data_array** matrix is a nanosecond time adjustment factor that will be used by the IDFS data access software when computing the time tags associated with the data (refer to **dr_time**). The nanosecond time adjustment factor is interpreted as a signed, 4-byte quantity and therefore, can reach a maximum of 2,147,483,647. However, since this value contains the resolution between nanosecond and millisecond precision, the value should be no larger than 999,999.

Unlike the other virtual instruments defined for ELS, **ELSENGS** basically consists of single-bit quantities that are packed together into an 8-bit word before being placed into the **data_array** as defined below:

| Array Position | IDFS Sensor | Comments |
|----------------|-------------|--------------------|
| data_array [4] | 0 – 5 | 1 value per sensor |

which is packed as

Byte 4

| | | | | | | | |
|----|----|----|----|----|----|---|---|
| S0 | S1 | S2 | S3 | S4 | S5 | 0 | 0 |
|----|----|----|----|----|----|---|---|

Although it is not necessary, the last two bits of the byte are zero-filled for the sake of clarity.

8.0 Definition for Virtual Instrument **ELSSWPL**

The low range sweep data for the ELS instrument form the virtual instrument **ELSSWPL**. The first three characters in the acronym are intended to identify the parent instrument (ELS), the next three characters (SWP) indicate that the data is from the sweep and the last character (L) indicates that the data is from the low range of the sweep.

The IDFS sensor definitions for the virtual instrument are defined in Table 8-1 below. Some ancillary or calibration data has been stored along with the primary sensor data in the data record and is identified in Table 8-2. The status flag definitions for the virtual instrument are defined in Table 8-3. The VIDF file for this virtual instrument can be found in Appendix A.

The Main Unit produces different types of science packets, based upon the data source. The science packet type identifier can be found in the **Data type** parameter, which is defined in byte 19, bits 4-7 in all science data packets generated by the Main Unit, except for the IMA packets. This parameter identifies the instrument (data source) to which the packet is related. For the ELS engineering information data packet, the value for the **Data type** parameter is 1 and the packets are referred to as ELS telemetry packets.

For each instrument defined, the Main Unit can produce multiple packet subtypes. For the ELS telemetry packets, the science packet subtype identifier can be found in the **ELS packet subtype** parameter, which is defined in byte 19, bits 0-1. For the ELS engineering information data packet, the value for the **ELS packet subtype** parameter is 0. In Table 8-1, Table 8-2 and Table 8-3, the heading **Data Subtype** refers to the packet subtype.

Table 8-1 ELSSWPL Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|----------------------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | ELS Deflection Voltage Reference | SCI | 1 | 0 | ELS Deflection reference | 38-39 |
| 1 | ELS Deflection Voltage Monitor | SCI | 1 | 0 | ELS Deflection monitor | 40-41 |

Table 8-2 ELSSWPL Calibration Definition

| Calibration Set | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|------------------------|-------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | ELS Temperature | SCI | 1 | 0 | ELS temperature | 33 |

Table 8-3 ELSSWPL Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|--------------------|-------------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | SW Version – Upper Byte | SCI | 1 | 0 | SW Version | 16 |
| 1 | SW Version – Lower Byte | SCI | 1 | 0 | SW Version | 17 |
| 2 | Software Mode | HSKP | - | - | sw_mode | 106 |

8.1 Notes for ELSSWPL virtual instrument

The Deflection Command is structured as follows:

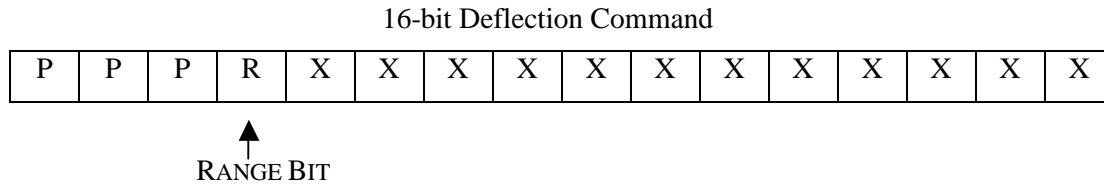


Figure 4 Deflection Command Bit Structure

where P = pad, R = range bit and X = value. The range determines if the instrument is operating in low range or high range. If the bit value is set to 0, the instrument is operating in low range and the data should be included in the **ELSSWPL** IDFS data set. If the bit value is set to 1, the instrument is operating in high range and the data should not be included in the **ELSSWPL** IDFS data set; the data should be included in the **ELSSWPH** IDFS data set. It should be noted that the Deflection Reference (or command) and Deflection Monitor are a pair of values whose interpretation both depend upon the range bit contained within the Deflection command word.

8.1.1 ELS Deflection Voltage Reference

The ELS Deflection Voltage Reference represents the value of the deflection voltage to which ELS was commanded for use in the collimator to sweep the charged particles into the sensor. This value is represented two different ways: by a 12-bit telemetry number and by the voltage across the collimator's deflection plates. The ELS Deflection Voltage Reference is converted to a voltage across the collimator's deflection plates through the formula:

$$\text{Deflection Plate Voltage Reference [V]} = 20.99 * \text{BMON} / 4095$$

where BMON is the value returned in telemetry and 4095 = 0x0fff for the maximum 12-bit value.

8.1.2 ELS Deflection Voltage Monitor

The ELS Deflection Voltage Monitor represents the deflection voltage actually achieved by ELS and used in the collimator to sweep the charged particles into the sensor. This value is represented two different ways: by a 12-bit telemetry number and by the voltage generated across the collimator's deflection plates. The ELS Deflection Voltage Monitor is converted to a voltage across the collimator's deflection plates through the formula:

$$\text{Deflection Plate Voltage Monitor [V]} = 20.99 * (5.0 / 4.5) * \text{BMON} / 4095$$

where BMON is the value returned in telemetry and 4095 = 0x0fff for the maximum 12-bit value.

8.2 ELSSWPL Header Record Format

The nominal values assigned to the header record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 8-4 ELSSWPL Header Record Definition

| Field | Description | Value |
|--------------|---------------------------------------------------|-------|
| hdr_len | header record size in bytes | DATA |
| year | year as YYYY | DATA |
| day | day of year as DDD | DATA |
| time_units | time base for data_accum | -6 |
| i_mode | no. of status flags returned in mode_index | 3 |
| data_accum | sample accumulation time | 28125 |
| data_lat | sample dead time (μ sec) | 3125 |
| swp_reset | sweep dead time (μ sec) | 0 |
| sen_reset | sensor set dead time (μ sec) | 0 |
| n_sen | no. of sensors returned in sensor set | 2 |
| n_sample | no. of data samples per sensor | DATA |
| scan_index | index into scan dependent tables | DATA |
| sensor_index | VIDF sensors returned in sensor set | DATA |
| d_qual | data quality flag per sensor | DATA |
| mode_index | status flag values | DATA |

8.2.1 Setting of *n_sample* header record element

There should be 128 values for the deflection reference (command) contained in a single packet, representing a single ELS sweep. The sweep may be all from the low range, all from the high range or a mixture of low range and high range. Due to this variable nature, **n_sample** cannot be pre-assigned the value of 128. The IDFS production code must look at the range bit for each individual element of the sweep to determine where that element will be recorded. The value for **n_sample** is set to the tally for the number of elements that are associated with the low range.

8.2.2 Setting of *scan_index* header record element

Scan_index is an array of **n_sample** values. The **scan_index** values indicate the sweep step number associated with each of the data samples returned per sensor. For example, if sweep steps 0 – 100 are from the high range and sweep steps 101 – 127 are from the low range, **n_sample** would be set to 27 and there would be 27 **scan_index** values, written as:

```
scan_index [0]      101
scan_index [1]      102
...
scan_index [26]     127
```

8.2.3 Setting of *sensor_index* header record element

Sensor_index is an array of **n_sen** values. Since all IDFS sensors will be returned per data record, **n_sen** is set to 2 and the values for **sensor_index** should be set accordingly:

```
sensor_index [0]    0
sensor_index [1]    1
```

8.2.4 Setting of *d_qual* header record element

D_qual is an array of **n_sen** values. According to the VIDF file outlined to date, there are currently 5 states defined for data quality, as shown in the table below:

| Value | Meaning | Criteria for Setting of Value |
|-------|-------------------|-------------------------------------------------------------------------------------------------|
| 0 | Good Data | default setting |
| 1 | Questionable Data | +30V monitor value is not within tolerance |
| 2 | Invalid Data | +12V, -12V, +5V or -5V monitor value is not within tolerance or the high voltage is not enabled |
| 3 | Bad Data | +30V monitor value and (+12V, -12V, +5V or -5V monitor value) are not within tolerance |
| 4 | Unknown State | Time related Main Unit Housekeeping packet is not available |

The setting of the data quality value to Questionable is dependent upon the state of the high voltage, which is specified in the single-bit parameter **els_enable_hv** (byte 30, bit 2) found within the Main Unit Housekeeping packet. If the value is set to 0 (disabled), the data quality value should be set to 1 (Questionable Data). Since all IDFS sensors will be returned per data record, **n_sen** is set to 2 and the values for **d_qual** should be set accordingly under nominal conditions:

```
d_qual [0]      0
d_qual [1]      0
```

8.2.5 Setting of *mode_index* header record element

Mode_index is an array of **i_mode** values. Table 8-3 indicates which telemetry packet fields are to be used to fill this array. The first two status bytes do not reflect the state of the instrument. They are simply used to record the version number of the Main Unit software used to generate the data packets. For status byte 2, a VIDF table will be defined to hold the values that are associated with each possible state value. The following is a brief explanation of the possible values and the meanings for status byte 2:

| Status Byte | 2 Software Mode |
|--------------------------|--------------------------------------------------------------|
| State Definitions | 0 = Off 1 = Booting 2 = Safe 3 = Prom 4 = Normal |

8.3 ELSSWPL Data Record Format

The nominal values assigned to the data record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 8-5 ELSSWPL Data Record Definition

| Field | Description | Value |
|--------------|------------------------------------|--------------|
| dr_time | initial data time (msec) | DATA |
| spin | azimuthal rate of rotation (msec) | DATA |
| sun_sen | last 0° crossing (msec) | DATA |
| hdr_off | header record offsets in bytes | DATA |
| nss | no. of sensor sets per data record | 1 |
| data_array | data values | DATA |

8.3.1 Setting of *dr_time* data record element

The time tag that will be utilized is the time tag defined in bytes 20-25 (SCET Time) in the ELS Science packet. This value is a 6-byte quantity that represents a CCSDS Unsegmented Time Code.

The finest resolution supported by the IDFS data access software is down to the nanosecond. However, **dr_time** holds the relative beginning time of day in milliseconds for the first data element of the first sensor set in the data record. In order to define the time tag down to the nanosecond precision, the VIDF field **nano_defined** must be set to indicate that a nanosecond time adjustment value is contained within the **data_array** matrix.

8.3.2 Setting of *spin* data record element

There are two parameters which must be considered in order to determine the value to store within the **spin** field in the IDFS data record. These two parameters are spin rate and spin direction. The spin rate is determined by the scanner speed. The scanner speed is reported in the **Scanner speed** parameter, which is byte 30, bits 0-1 in the ELS Science packet. The possible scanner speeds are 0 (non spinning, value = 0), 32 second scan (value = 1), 64 second scan (value = 2), and 128 second scan (value = 3). A scan consists of a turning of 180 degrees. The scanner moves forward and backward through an angle of 180 degrees during normal operation.

The spin direction is determined by the scanner direction. The scanner direction is reported in the **Scanner direction** parameter, which is byte 30, bit 2 in the ELS Science packet. The value for scanner direction will either be 0 to indicate a 0-180 direction or 1 to indicate a 180-0 direction.

The data value of **spin** is given in milliseconds per revolution (a revolution is 360 degrees). When the scanner direction is 0 – 180, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | 64 | 64000 |
| 64 second scan | 128 | 128000 |
| 128 second scan | 256 | 256000 |

When the scanner direction is 180 - 0, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | -64 | -64000 |
| 64 second scan | -128 | -128000 |
| 128 second scan | -256 | -256000 |

8.3.3 Setting of *sun_sen* data record element

There are three parameters which must be considered in order to determine the value to store within the **sun_sen** field in the IDFS data record. These three parameters are scanner speed, scanner direction and scanner position. The scanner position is reported in the **Scanner position** parameter, which is byte 31 in the ELS Science packet. The scanner position is reset to 0 when the scanner is pointing in the direction of 0 degrees or less. There are 0.806 degrees per count under normal conditions. This means that at 180 degrees, the scanner position count is 223.

If the scanner speed is 0 (non spinning), then the scanner position marks where the scanner is pointing. Let's call this position **P**. The **sun_sen** value written in the data record is given as degrees times 100 for the non spinning case. Thus, the value written in the **sun_sen** field of the data record is $180 * 100 * P / 223$.

If the scanner speed is not 0, then the scanner direction will need to be examined in order to project where the 0 offset (or 0 degree) location was or will be located with respect to time.

If the scanner direction is from 0-180, the scanner crossed the 0 degree marker some time in the past. For the value of the scanner position **P**, the value was

$$\text{spin} / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **subtracted** from the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the subtraction is negative, this means that the **sun_sen** time crossed the day boundary. To change this to the next 0 degree marker crossing, **add** the value

$$\text{spin} / 2 * (2 - P / 223)$$

to the SCET time.

If the scanner direction is from 180-0, then the scanner will cross the 0 degree marker some time in the future. For the value of scanner position **P**, the value will be

$$\text{abs}(\text{spin}) / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **added** to the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the addition is greater than or equal to 86400 seconds, this means that the **sun_sen** time crossed the day boundary. To change this to the previous 0 degree marker crossing, **subtract** the value

$$\text{abs}(\text{spin}) / 2 * (2 - P / 223)$$

from the SCET time.

8.3.4 Setting of *data_array* data record element

The first four bytes contained within the **data_array** matrix is a nanosecond time adjustment factor that will be used by the IDFS data access software when computing the time tags associated with the data (refer to **dr_time**). The nanosecond time adjustment factor is interpreted as a signed, 4-byte quantity and therefore, can reach a maximum of 2,147,483,647. However, since this value contains the resolution between nanosecond and millisecond precision, the value should be no larger than 999,999. The remainder of the **data_array** for **ELSSWPL** should contain sensor and calibration data, with the sensor data laid down first according to the layout provided below:

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Packet Bit Position | Comments |
|--------------------------------------|-------------|---------------|-----------|--------------|----------------------|---------------------|----------|
| data_array [4 thru (N*2)-1+4] | 0 | SCI | 1 | 0 | 38-39 | 0-11 | N values |
| data_array [(N*2)+4 thru 2(N*2)-1+4] | 1 | SCI | 1 | 0 | 40-41 | 0-15 | N values |

In the table above and below, the variable **N** represents the value **n_sample**, as defined in the header record. Notice that for the indexing specification for **data_array**, the variable **N** is multiplied by 2 since the data itself is 16-bit data and the **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record..

The calibration data is then placed into **data_array**, at the end of the sensor data. The table below shows how the calibration data is organized within **data_array**. The format **C_i / S_x** identifies the data as calibration (C) data from calibration set **i** that are associated with IDFS sensor (S) number **x**. The variable **M** in the table represents the value $4 + n_{sen} * n_{sample} * 2$.

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Packet Bit Position | Comments |
|-----------------------------------------------|-------------|---------------|-----------|--------------|----------------------|---------------------|------------------------------|
| data_array [M] | C0/S0 | SCI | 1 | 0 | 33 | 0-7 | 1 value |
| data_array [M + 2] | C0/S1 | SCI | 1 | 0 | 33 | 0-7 | 1 value |
| data_array [M + 4 thru (257 * 2) + 4] | n/a | n/a | n/a | n/a | n/a | n/a | 258 – (2N + 2) values, all 0 |

Notice that the index value is increasing by 2 for **data_array**. The reason for this indexing scheme is based upon the fact that the calibration data is stored as a 16-bit value and **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record. While the header record associated with the data record may set **n_sample** to some value other than 128, the data array must be fixed at 128 elements per IDFS sensor since the data records are fixed-length records. For the **ELSSWPL** virtual instrument, this would equate to $128 * n_{sen} + 1 * n_{sen}$ values per data record, for a total of 258 elements. The IDFS production code must pack the active sensor and calibration data, then pad the remainder of the data record with zero-fill as illustrated in Figure 5 below so that the data record's fixed-length is reached.

| | | | | | |
|-------------|---------|---------|-------|-------|-----------|
| 4-Byte Nsec | S0 Data | S1 Data | C0/S0 | C0/S1 | Zero Fill |
|-------------|---------|---------|-------|-------|-----------|

Figure 5 Data Array for ELSSWPL

For IDFS sensor 0, the data always starts at byte 38 of the ELS Science Data packet, with the 16-bit value representing step 1 of the sweep. In order to get to the remaining elements in the sweep, the IDFS production code must add 4 to the base byte offset of 38 for each successive element until all 128 steps have been processed. In other words, the byte offset for step 2 is 42 (38 + 4), the byte offset for step 3 is 46 (38 + 8), etc. until step 128 is processed at byte offset 546 (38 + 508). The same byte offset calculation is

needed for IDFS sensor 1, except that the base byte offset for step 1 of the sweep is byte 40 of the ELS Science Data packet.

9.0 Definition for Virtual Instrument ELSSWPH

The high range sweep data for the ELS instrument form the virtual instrument **ELSSWPH**. The first three characters in the acronym are intended to identify the parent instrument (ELS), the next three characters (SWP) indicate that the data is from the sweep and the last character (H) indicates that the data is from the high range of the sweep.

The IDFS sensor definitions for the virtual instrument are defined in Table 9-1 below. Some ancillary or calibration data has been stored along with the primary sensor data in the data record and is identified in Table 9-2. The status flag definitions for the virtual instrument are defined in Table 9-3. The VIDF file for this virtual instrument can be found in Appendix A.

The Main Unit produces different types of science packets, based upon the data source. The science packet type identifier can be found in the **Data type** parameter, which is defined in byte 19, bits 4-7 in all science data packets generated by the Main Unit, except for the IMA packets. This parameter identifies the instrument (data source) to which the packet is related. For the ELS engineering information data packet, the value for the **Data type** parameter is 1 and the packets are referred to as ELS telemetry packets.

For each instrument defined, the Main Unit can produce multiple packet subtypes. For the ELS telemetry packets, the science packet subtype identifier can be found in the **ELS packet subtype** parameter, which is defined in byte 19, bits 0-1. For the ELS engineering information data packet, the value for the **ELS packet subtype** parameter is 0. In Table 9-1, Table 9-2 and Table 9-3, the heading **Data Subtype** refers to the packet subtype.

Table 9-1 ELSSWPH Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|----------------------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | ELS Deflection Voltage Reference | SCI | 1 | 0 | ELS Deflection reference | 38-39 |
| 1 | ELS Deflection Voltage Monitor | SCI | 1 | 0 | ELS Deflection monitor | 40-41 |

Table 9-2 ELSSWPH Calibration Definition

| Calibration Set | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|------------------------|-------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | ELS Temperature | SCI | 1 | 0 | ELS temperature | 33 |

Table 9-3 ELSSWPH Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|--------------------|-------------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | SW Version – Upper Byte | SCI | 1 | 0 | SW Version | 16 |
| 1 | SW Version – Lower Byte | SCI | 1 | 0 | SW Version | 17 |
| 2 | Software Mode | HSKP | - | - | sw_mode | 106 |

9.1 Notes for ELSSWPH virtual instrument

The Deflection Command is structured as follows:

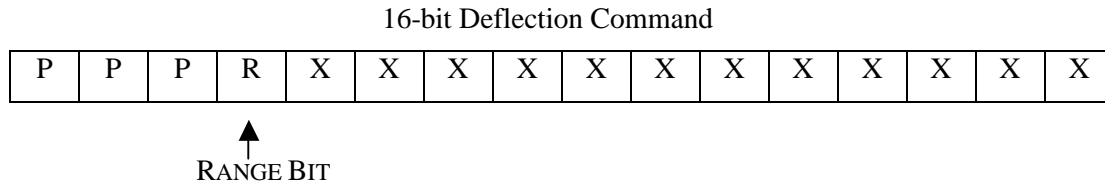


Figure 6 Deflection Command Bit Structure

where P = pad, R = range bit and X = value. The range determines if the instrument is operating in low range or high range. If the bit value is set to 1, the instrument is operating in high range and the data should be included in the **ELSSWPH** IDFS data set. If the bit value is set to 0, the instrument is operating in low range and the data should not be included in the **ELSSWPH** IDFS data set; the data should be included in the **ELSSWPL** IDFS data set. It should be noted that the Deflection Reference (or command) and Deflection Monitor are a pair of values whose interpretation both depend upon the range bit contained within the Deflection command word.

9.1.1 ELS Deflection Voltage Reference

The ELS Deflection Voltage Reference represents the value of the deflection voltage to which ELS was commanded for use in the collimator to sweep the charged particles into the sensor. This value is represented two different ways: by a 12-bit telemetry number and by the voltage across the collimator’s deflection plates. The ELS Deflection Voltage Reference is converted to a voltage across the collimator’s deflection plates through the formula:

$$\text{Deflection Plate Voltage Reference [V]} = 2800 * \text{BMON} / 4095$$

where BMON is the value returned in telemetry and 4095 = 0x0fff for the maximum 12-bit value.

9.1.2 ELS Deflection Voltage Monitor

The ELS Deflection Voltage Monitor represents the deflection voltage actually achieved by ELS and used in the collimator to sweep the charged particles into the sensor. This value is represented two different ways: by a 12-bit telemetry number and by the voltage generated across the collimator’s deflection plates. The ELS Deflection Voltage Monitor is converted to a voltage across the collimator’s deflection plates through the formula:

$$\text{Deflection Plate Voltage Monitor [V]} = 2800 * (5.0 / 4.5) * \text{BMON} / 4095$$

where BMON is the value returned in telemetry and 4095 = 0x0fff for the maximum 12-bit value.

9.2 ELSSWPH Header Record Format

The nominal values assigned to the header record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 9-4 ELSSWPH Header Record Definition

| Field | Description | Value |
|--------------|---------------------------------------------------|-------|
| hdr_len | header record size in bytes | DATA |
| year | year as YYYY | DATA |
| day | day of year as DDD | DATA |
| time_units | time base for data_accum | -6 |
| i_mode | no. of status flags returned in mode_index | 3 |
| data_accum | sample accumulation time | 28125 |
| data_lat | sample dead time (μ sec) | 3125 |
| swp_reset | sweep dead time (μ sec) | 0 |
| sen_reset | sensor set dead time (μ sec) | 0 |
| n_sen | no. of sensors returned in sensor set | 2 |
| n_sample | no. of data samples per sensor | DATA |
| scan_index | index into scan dependent tables | DATA |
| sensor_index | VIDF sensors returned in sensor set | DATA |
| d_qual | data quality flag per sensor | DATA |
| mode_index | status flag values | DATA |

9.2.1 Setting of *n_sample* header record element

There should be 128 values for the deflection reference (command) contained in a single packet, representing a single ELS sweep. The sweep may be all from the low range, all from the high range or a mixture of low range and high range. Due to this variable nature, **n_sample** cannot be pre-assigned the value of 128. The IDFS production code must look at the range bit for each individual element of the sweep to determine where that element will be recorded. The value for **n_sample** is set to the tally for the number of elements that are associated with the high range.

9.2.2 Setting of *scan_index* header record element

Scan_index is an array of **n_sample** values. The **scan_index** values indicate the sweep step number associated with each of the data samples returned per sensor. For example, if sweep steps 0 – 100 are from the high range and sweep steps 101 – 127 are from the low range, **n_sample** would be set to 101 and there would be 101 **scan_index** values, written as:

```
scan_index [0]      0
scan_index [1]      1
...
scan_index [100]    100
```

9.2.3 Setting of *sensor_index* header record element

Sensor_index is an array of **n_sen** values. Since all IDFS sensors will be returned per data record, **n_sen** is set to 2 and the values for **sensor_index** should be set accordingly:

```
sensor_index [0]    0
sensor_index [1]    1
```

9.2.4 Setting of *d_qual* header record element

D_qual is an array of **n_sen** values. According to the VIDF file outlined to date, there are currently 5 states defined for data quality, as shown in the table below:

| Value | Meaning | Criteria for Setting of Value |
|-------|-------------------|-------------------------------------------------------------------------------------------------|
| 0 | Good Data | default setting |
| 1 | Questionable Data | +30V monitor value is not within tolerance |
| 2 | Invalid Data | +12V, -12V, +5V or -5V monitor value is not within tolerance or the high voltage is not enabled |
| 3 | Bad Data | +30V monitor value and (+12V, -12V, +5V or -5V monitor value) are not within tolerance |
| 4 | Unknown State | Time related Main Unit Housekeeping packet is not available |

The setting of the data quality value to Questionable is dependent upon the state of the high voltage, which is specified in the single-bit parameter **els_enable_hv** (byte 30, bit 2) found within the Main Unit Housekeeping packet. If the value is set to 0 (disabled), the data quality value should be set to 1 (Questionable Data). Since all IDFS sensors will be returned per data record, **n_sen** is set to 2 and the values for **d_qual** should be set accordingly under nominal conditions:

```
d_qual [0]      0
d_qual [1]      0
```

9.2.5 Setting of *mode_index* header record element

Mode_index is an array of **i_mode** values. Table 9-3 indicates which telemetry packet fields are to be used to fill this array. The first two status bytes do not reflect the state of the instrument. They are simply used to record the version number of the Main Unit software used to generate the data packets. For status byte 2, a VIDF table will be defined to hold the values that are associated with each possible state value. The following is a brief explanation of the possible values and the meanings for status byte 2:

| Status Byte | 2 Software Mode |
|--------------------------|--------------------------------------------------------------|
| State Definitions | 0 = Off 1 = Booting 2 = Safe 3 = Prom 4 = Normal |

9.3 ELSSWPH Data Record Format

The nominal values assigned to the data record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 9-5 ELSSWPH Data Record Definition

| Field | Description | Value |
|------------|------------------------------------|-------|
| dr_time | initial data time (msec) | DATA |
| spin | azimuthal rate of rotation (msec) | DATA |
| sun_sen | last 0° crossing (msec) | DATA |
| hdr_off | header record offsets in bytes | DATA |
| nss | no. of sensor sets per data record | 1 |
| data_array | data values | DATA |

9.3.1 Setting of *dr_time* data record element

The time tag that will be utilized is the time tag defined in bytes 20-25 (SCET Time) in the ELS Science packet. This value is a 6-byte quantity that represents a CCSDS Unsegmented Time Code.

The finest resolution supported by the IDFS data access software is down to the nanosecond. However, **dr_time** holds the relative beginning time of day in milliseconds for the first data element of the first sensor set in the data record. In order to define the time tag down to the nanosecond precision, the VIDF field **nano_defined** must be set to indicate that a nanosecond time adjustment value is contained within the **data_array** matrix.

9.3.2 Setting of *spin* data record element

There are two parameters which must be considered in order to determine the value to store within the **spin** field in the IDFS data record. These two parameters are spin rate and spin direction. The spin rate is determined by the scanner speed. The scanner speed is reported in the **Scanner speed** parameter, which is byte 30, bits 0-1 in the ELS Science packet. The possible scanner speeds are 0 (non spinning, value = 0), 32 second scan (value = 1), 64 second scan (value = 2), and 128 second scan (value = 3). A scan consists of a turning of 180 degrees. The scanner moves forward and backward through an angle of 180 degrees during normal operation.

The spin direction is determined by the scanner direction. The scanner direction is reported in the **Scanner direction** parameter, which is byte 30, bit 2 in the ELS Science packet. The value for scanner direction will either be 0 to indicate a 0-180 direction or 1 to indicate a 180-0 direction.

The data value of **spin** is given in milliseconds per revolution (a revolution is 360 degrees). When the scanner direction is 0 – 180, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | 64 | 64000 |
| 64 second scan | 128 | 128000 |
| 128 second scan | 256 | 256000 |

When the scanner direction is 180 - 0, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | -64 | -64000 |
| 64 second scan | -128 | -128000 |
| 128 second scan | -256 | -256000 |

9.3.3 Setting of *sun_sen* data record element

There are three parameters which must be considered in order to determine the value to store within the **sun_sen** field in the IDFS data record. These three parameters are scanner speed, scanner direction and scanner position. The scanner position is reported in the **Scanner position** parameter, which is byte 31 in the ELS Science packet. The scanner position is reset to 0 when the scanner is pointing in the direction of 0 degrees or less. There are 0.806 degrees per count under normal conditions. This means that at 180 degrees, the scanner position count is 223.

If the scanner speed is 0 (non spinning), then the scanner position marks where the scanner is pointing. Let's call this position **P**. The **sun_sen** value written in the data record is given as degrees times 100 for the non spinning case. Thus, the value written in the **sun_sen** field of the data record is $180 * 100 * P / 223$.

If the scanner speed is not 0, then the scanner direction will need to be examined in order to project where the 0 offset (or 0 degree) location was or will be located with respect to time.

If the scanner direction is from 0-180, the scanner crossed the 0 degree marker some time in the past. For the value of the scanner position **P**, the value was

$$\text{spin} / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **subtracted** from the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the subtraction is negative, this means that the **sun_sen** time crossed the day boundary. To change this to the next 0 degree marker crossing, **add** the value

$$\text{spin} / 2 * (2 - P / 223)$$

to the SCET time.

If the scanner direction is from 180-0, then the scanner will cross the 0 degree marker some time in the future. For the value of scanner position **P**, the value will be

$$\text{abs}(\text{spin}) / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **added** to the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the addition is greater than or equal to 86400 seconds, this means that the **sun_sen** time crossed the day boundary. To change this to the previous 0 degree marker crossing, **subtract** the value

$$\text{abs}(\text{spin}) / 2 * (2 - P / 223)$$

from the SCET time.

9.3.4 Setting of *data_array* data record element

The first four bytes contained within the **data_array** matrix is a nanosecond time adjustment factor that will be used by the IDFS data access software when computing the time tags associated with the data (refer to **dr_time**). The nanosecond time adjustment factor is interpreted as a signed, 4-byte quantity and therefore, can reach a maximum of 2,147,483,647. However, since this value contains the resolution between nanosecond and millisecond precision, the value should be no larger than 999,999. The remainder of the **data_array** for **ELSSWPH** should contain sensor and calibration data, with the sensor data laid down first according to the layout provided below:

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Packet Bit Position | Comments |
|--------------------------------------|-------------|---------------|-----------|--------------|----------------------|---------------------|----------|
| data_array [4 thru (N*2)-1+4] | 0 | SCI | 1 | 0 | 38-39 | 0-11 | N values |
| data_array [(N*2)+4 thru 2(N*2)-1+4] | 1 | SCI | 1 | 0 | 40-41 | 0-15 | N values |

In the table above and below, the variable **N** represents the value **n_sample**, as defined in the header record. Notice that for the indexing specification for **data_array**, the variable **N** is multiplied by 2 since the data itself is 16-bit data and the **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record..

The calibration data is then placed into **data_array**, at the end of the sensor data. The table below shows how the calibration data is organized within **data_array**. The format **Ci / Sx** identifies the data as calibration (C) data from calibration set **i** that are associated with IDFS sensor (S) number **x**. The variable **M** in the table represents the value $4 + n_{sen} * n_{sample} * 2$.

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Packet Bit Position | Comments |
|--------------------------------------------------|-------------|---------------|-----------|--------------|----------------------|---------------------|--------------------------------|
| data_array [M] | C0/S0 | SCI | 1 | 0 | 33 | 0-7 | 1 value |
| data_array [M + 2] | C0/S1 | SCI | 1 | 0 | 33 | 0-7 | 1 value |
| data_array [M + 4 thru $(257 * 2) + 4$] | n/a | n/a | n/a | n/a | n/a | n/a | $258 - (2N + 2)$ values, all 0 |

Notice that the index value is increasing by 2 for **data_array**. The reason for this indexing scheme is based upon the fact that the calibration data is stored as a 16-bit value and **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record. While the header record associated with the data record may set **n_sample** to some value other than 128, the data array must be fixed at 128 elements per IDFS sensor since the data records are fixed-length records. For the **ELSSWPH** virtual instrument, this would equate to $128 * n_{sen} + 1 * n_{sen}$ values per data record, for a total of 258 elements. The IDFS production code must pack the active sensor and calibration data, then pad the remainder of the data record with zero-fill as illustrated in Figure 7 below so that the data record's fixed-length is reached.

| | | | | | |
|-------------|---------|---------|-------|-------|-----------|
| 4-Byte Nsec | S0 Data | S1 Data | C0/S0 | C0/S1 | Zero Fill |
|-------------|---------|---------|-------|-------|-----------|

Figure 7 Data Array for ELSSWPH

For IDFS sensor 0, the data always starts at byte 38 of the ELS Science Data packet, with the 16-bit value representing step 1 of the sweep. In order to get to the remaining elements in the sweep, the IDFS production code must add 4 to the base byte offset of 38 for each successive element until all 128 steps have been processed. In other words, the byte offset for step 2 is 42 (38 + 4), the byte offset for step 3 is 46 (38 + 8), etc. until step

128 is processed at byte offset 546 ($38 + 508$). The same byte offset calculation is needed for IDFS sensor 1, except that the base byte offset for step 1 of the sweep is byte 40 of the ELS Science Data packet.

10.0 Definition for Virtual Instrument **ELSSCIL**

The science data from the low range portion of the ELS power supply form the virtual instrument **ELSSCIL**. The first three characters in the acronym are intended to identify the parent instrument (ELS), the next three characters (SCI) indicate science data is being returned and the last character (L) indicates that the data is from the low range portion of the ELS power supply.

The IDFS sensor definitions for the virtual instrument are defined in Table 10-1 below. Refer to sections 10.1 and 10.3.4 for a more in-depth explanation of the **Packet Byte Position** column information. Some ancillary or calibration data has been stored along with the primary sensor data in the data record and is identified in Table 10-2. The status flag definitions for the virtual instrument are defined in Table 10-3. The VIDF file for this virtual instrument can be found in Appendix A.

The Main Unit produces different types of science packets, based upon the data source. The science packet type identifier can be found in the **Data type** parameter, which is defined in byte 19, bits 4-7 in all science data packets generated by the Main Unit, except for the IMA packets. This parameter identifies the instrument (data source) to which the packet is related. For the ELS engineering information data packet, the value for the **Data type** parameter is 1 and the packets are referred to as ELS telemetry packets.

For each instrument defined, the Main Unit can produce multiple packet subtypes. For the ELS telemetry packets, the science packet subtype identifier can be found in the **ELS packet subtype** parameter, which is defined in byte 19, bits 0-1. For the ELS science data packet, the value for the **ELS packet subtype** parameter is either 1, 2 or 3. In Table 10-1, Table 10-2 and Table 10-3, the heading **Data Subtype** refers to the packet subtype.

Table 10-1 ELSSCIL Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|-------------------|----------------------|------------------|---------------------|--------------------------|--------------------------------------------|
| 0 | ELS Anode 0 | SCI | 1 | 1, 2 or 3 | Data | 32 |
| 1 | ELS Anode 1 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 2 | ELS Anode 2 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 3 | ELS Anode 3 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 4 | ELS Anode 4 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 5 | ELS Anode 5 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 6 | ELS Anode 6 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |

Table 10-1 ELSSCIL Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|-------------------|----------------------|------------------|---------------------|--------------------------|--------------------------------------------|
| 7 | ELS Anode 7 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 8 | ELS Anode 8 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 9 | ELS Anode 9 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 10 | ELS Anode 10 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 11 | ELS Anode 11 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 12 | ELS Anode 12 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 13 | ELS Anode 13 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 14 | ELS Anode 14 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 15 | ELS Anode 15 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |

Table 10-2 ELSSCIL Calibration Definition

| Calibration Set | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|------------------------|-----------------------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | ELS Temperature | SCI | 1 | 0 | ELS temperature | 33 |
| 1 | ELS MCP Bias Reference | SCI | 1 | 0 | ELS MCP reference | 34 |
| 2 | ELS MCP Bias Monitor | SCI | 1 | 0 | ELS MCP monitor | 35 |
| 3 | Total Range Anode Summation (MSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 4 | Total Range Anode Summation (LSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 5 | Low Range Sweep Summation(MSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |

Table 10-2 ELSSCIL Calibration Definition

| Calibration Set | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|-----------------|-----------------------------------|---------------|-----------|--------------|-------------------|----------------------|
| 6 | Low Range Sweep Summation (LSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 7 | Total Range Sweep Summation (MSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 8 | Total Range Sweep Summation (LSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |

Calibration sets 3 through 8 represent data values that are derived and computed in the IDFS production code as opposed to values directly transferred from the telemetry source packets. The reference to “Last” indicates the byte position taken up by the last sector for the last step included in the packet being processed.

Table 10-3 ELSSCIL Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position | Packet Bit Position |
|-------------|-------------------------|---------------|-----------|--------------|--------------------|----------------------|---------------------|
| 0 | SW Version – Upper Byte | SCI | 1 | 1, 2 or 3 | SW Version | 16 | 0-7 |
| 1 | SW Version – Lower Byte | SCI | 1 | 1, 2 or 3 | SW Version | 17 | 0-7 |
| 2 | Software Mode | HSKP | - | - | sw_mode | 106 | 0-7 |
| 3 | Time Summation | SCI | 1 | 1, 2 or 3 | Time compression | 29 | 0-2 |
| 4 | Energy Summation | SCI | 1 | 1, 2 or 3 | Energy compression | 29 | 3-4 |
| 5 | Log Compression | SCI | 1 | 1, 2 or 3 | Log compression | 29 | 5 |
| 6 | Sector 0 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 0 |
| 7 | Sector 1 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 1 |
| 8 | Sector 2 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 2 |

Table 10-3 ELSSCIL Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position | Packet Bit Position |
|-------------|-------------------------|---------------|-----------|--------------|-------------------|----------------------|---------------------|
| 9 | Sector 3 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 3 |
| 10 | Sector 4 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 4 |
| 11 | Sector 5 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 5 |
| 12 | Sector 6 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 6 |
| 13 | Sector 7 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 7 |
| 14 | Sector 8 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 8 |
| 15 | Sector 9 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 9 |
| 16 | Sector 10 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 10 |
| 17 | Sector 11 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 11 |
| 18 | Sector 12 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 12 |
| 19 | Sector 13 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 13 |
| 20 | Sector 14 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 14 |
| 21 | Sector 15 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 15 |
| 22 | Number Of Active Anodes | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | N/A |

10.1 Notes for ELSSCIL virtual instrument

For the science data, the data that is produced in one sweep for ELS cannot fit into a single ELS Science Packet when uncompressed packets are transmitted. For this scenario, the data will be split into two packets, with steps 0-63 being packed into one packet (subtype = 2) and steps 64-127 being packed into another packet (subtype = 3). If the data is compressed, all steps (0 – 127) may be returned within a single packet (subtype = 1). The mode of transmission is known by examining the **ELS packet subtype** field – byte 19, bits 0-1 – contained within the packet. If the data is split between 2 packets, the IDFS production code will acquire both packets before processing the data into an IDFS data record so that all 128 steps are available. Both packets will have the same time tag on packet bytes 6 – 11; however, the sequence count (bytes 2-3, bits 13-0) will be different for these two packets.

The **ELS sector mask** field contained within the ELS science packet identifies which of the sixteen possible anodes are being returned in the data matrix. This field is a 16-bit word and this field will be set to reflect the subset of anodes that are being returned in the data packet(s). In other words, only those sectors that are being returned will be set to indicate that data is available for those sectors and all other sectors will be set to indicate that no data is available. Figure 8 below shows the ordering of the sector mask, where Anode 15 represents the MSB (bit 15) of the word and Anode 0 represents the LSB (bit 0) of the word.

| | | | | | | | | | | | | | | | |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| Anode 15 | Anode 14 | Anode 13 | Anode 12 | Anode 11 | Anode 10 | Anode 9 | Anode 8 | Anode 7 | Anode 6 | Anode 5 | Anode 4 | Anode 3 | Anode 2 | Anode 1 | Anode 0 |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|

Figure 8 Sector Mask Anode Number / Bit Number Association

For the science data, the data may be transmitted in compressed form. The APAF system will define a software utility library which will contain the decompression algorithms that are to be used to decompress (RICE and Log) the data within the ELS, NPI and NPD Science packets. The parameter **RICE compression** (byte 29, bit 6) needs to be examined in order to determine whether the data has been RICE compressed or not. A value of 1 indicates that RICE compression has been enabled. If the data has been RICE compressed, the data must be uncompressed before any of the other ELS compression / summation schemes are examined. The parameter **Log compression enabled** (byte 29, bit 5) needs to be examined in order to determine whether the 16-bit counts have been converted to 8-bit values. A value of 1 indicates that Log compression has been enabled. The ELS instrument makes use of a summation scheme where the summation of multiple sweeps takes place onboard before the data is transmitted. Refer to Appendix C for an explanation of how the IDFS production code should decompress the data contained within the ELS science packets.

10.1.1 ELS Calibration Data

For calibration sets 0 – 4, there will be one scalar value defined that will be applicable to each of the IDFS sensors that are returned in telemetry. For calibration sets 5 – 8, there will be one scalar value defined for each of the IDFS sensors that are returned in telemetry. Each IDFS sensor corresponds to a specific ELS anode. If all ELS anodes are returned, there will be 16 separate IDFS sensor values to be processed. Calibration sets 3 through 8 represent data values that are derived and computed in the IDFS production code as opposed to values directly transferred from the telemetry source packets. Calibration Sets 5 and 6 – **Low Range Sweep Summation** - represent a summation which corresponds to an IDFS sensor which is computed by adding together the science data for the energy steps where the sweep step is in low range. The algorithm is shown below

$$\text{calibration [sensor]} = \sum_0^{\text{sweep length}} \text{data [sensor]}$$

and the data value to be included in the summation will either be: a) 0 if the step is in high range for the sensor or b) science data if the step is in low range for the sensor. The IDFS production code will need to create this summation before expanding and padding the energy sweep to account for energy summation, but after Rice Decompression and bit decompression have been applied. Since the data is a 16-bit value, there is the chance for overflow of a 16-bit calibration value; therefore, the summation is computed using a 32-bit value as the accumulator and the summation value is written into the IDFS data record as two 16-bit calibration values (MSB and LSB).

Calibration Sets 7 and 8 – **Total Range Sweep Summation** – represent a summation which corresponds to an IDFS sensor which is computed by adding together the science data for every energy step in the sweep. The algorithm is shown below

$$\text{calibration [sensor]} = \sum_0^{\text{sweep length}} \text{data [sensor]}$$

and the data value to be included in the summation will be the science data for that sensor. The IDFS production code will need to create this summation before expanding and padding the energy sweep to account for energy summation, but after Rice Decompression and bit decompression have been applied. Since the data is a 16-bit value, there is the chance for overflow of a 16-bit calibration value; therefore, the summation is computed using a 32-bit value as the accumulator and the summation value is written into the IDFS data record as two 16-bit calibration values (MSB and LSB).

Calibration Sets 3 and 4 – **Total Range Anode Summation** - is formed using the **Total Range Sweep Summation** values for each of the IDFS sensors that are processed. The algorithm is shown below

$$\text{calibration} = \sum_0^{\text{no. of IDFS sensors}} \text{Total Range Sweep Summation [sensor]}$$

and the data to be included in the summation will be the data from Calibration Sets 7 and 8 for the IDFS sensor being processed. This summation results in a single scalar value which needs to be written into the data record once. Since the data is a 16-bit value, there is the chance for overflow of a 16-bit calibration value; therefore, the summation is computed using a 32-bit value as the accumulator and the summation value is written into the IDFS data record as two 16-bit calibration values (MSB and LSB).

10.2 ELSSCIL Header Record Format

The nominal values assigned to the header record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 10-4 ELSSCIL Header Record Definition

| Field | Description | Value |
|--------------|---------------------------------------------------|-------|
| hdr_len | header record size in bytes | DATA |
| year | year as YYYY | DATA |
| day | day of year as DDD | DATA |
| time_units | time base for data_accum | -6 |
| i_mode | no. of status flags returned in mode_index | 23 |
| data_accum | sample accumulation time | 28125 |
| data_lat | sample dead time (μ sec) | 3125 |
| swp_reset | sweep dead time (μ sec) | DATA |
| sen_reset | sensor set dead time (μ sec) | 0 |
| n_sen | no. of sensors returned in sensor set | DATA |
| n_sample | no. of data samples per sensor | DATA |
| scan_index | index into scan dependent tables | DATA |
| sensor_index | VIDF sensors returned in sensor set | DATA |
| d_qual | data quality flag per sensor | DATA |
| mode_index | status flag values | DATA |

10.2.1 Setting of *swp_reset* header record element

The amount of dead time necessary for the ELS instrument to reset itself between successive sweeps is defined in order to account for the missing flyback step. This value is dependent upon the Energy Compression bit returned in the ELS science packets. The Energy Compression bit indicates how many energy steps have been summed over for each value returned. The value of **swp_reset** is given in microsecond :

| Energy Compression value | No. Of Steps Summed | swp_reset value |
|--------------------------|---------------------|-----------------|
| 0 | 1 | 31250 |
| 1 | 2 | 62500 |
| 2 | 4 | 125000 |

10.2.2 Setting of *n_sen* header record element

The maximum number of anode sectors being returned by ELS is 16; however, data for all sectors may not always be returned. The field **ELS sector mask**, bytes 26-27 in the ELS Science Packet, indicates which anode sectors are being returned. This field will hold a 16-bit filter mask selection word that identifies which of the sixteen anode sectors are returned in the science data matrix contained in the packet. The values for each bit should be defined as:

0 = block sector data

1 = transmit sector data

The value for **n_sen** should be set to reflect the actual number of sectors returned out of the 16 possible sectors.

10.2.3 Setting of *n_sample* header record element

There should be 128 values for the deflection reference (command) contained in a single packet, representing a single ELS sweep. The sweep may be all from the low range, all from the high range or a mixture of low range and high range. Due to this variable nature, **n_sample** cannot be pre-assigned the value of 128. The IDFS production code must look at the range bit for each individual element of the sweep to determine where that element will be recorded. The value for **n_sample** is set to the tally for the number of elements that are associated with the low range. The value for **n_sample** should not include the data for the last energy compressed step within the ELS sweep since the last step is the flyback step and the data associated with the flyback step is considered invalid science data. The energy steps to ignore are based upon the setting for energy compression and are indicated in the table below:

| Energy Compression value | Energy Steps to Ignore |
|--------------------------|------------------------|
| 0 | 127 |
| 1 | 126 and 127 |
| 2 | 124, 125, 126 and 127 |

10.2.4 Setting of *scan_index* header record element

Scan_index is an array of **n_sample** values. The **scan_index** values indicate the sweep step number associated with each of the data samples returned per sensor. The values for **scan_index** are to be taken from the ELS Engineering Information packet, utilizing the 128 deflection reference values, extracting only those that are associated with the low range of the ELS sweep. These values indicate the correct step the value was taken at (value ranging from 0 to 4095). Again, the flyback data is not included.

10.2.5 Setting of *sensor_index* header record element

Sensor_index is an array of **n_sen** values. The values should be set to reflect which sectors are contained in the data array. For example, if the sector mask was set to

0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1

this would indicate that data for sectors 0, 2, and 4 were being returned. The value for **n_sen** should be set to 3 and the values for **sensor_index** should be set as follows:

```

sensor_index [0]    0
sensor_index [1]    2
sensor_index [2]    4
    
```

10.2.6 Setting of *d_qual* header record element

D_qual is an array of **n_sen** values. According to the VIDF file outlined to date, there are currently 5 states defined for data quality, as shown in the table below:

| Value | Meaning | Criteria for Setting of Value |
|-------|-------------------|-------------------------------------------------------------------------------------------------|
| 0 | Good Data | default setting |
| 1 | Questionable Data | +30V monitor value is not within tolerance |
| 2 | Invalid Data | +12V, -12V, +5V or -5V monitor value is not within tolerance or the high voltage is not enabled |
| 3 | Bad Data | +30V monitor value and (+12V, -12V, +5V or -5V monitor value) are not within tolerance |
| 4 | Unknown State | Time related Main Unit Housekeeping packet is not available |

The setting of the data quality value to Questionable is dependent upon the state of the high voltage, which is specified in the single-bit parameter **els_enable_hv** (byte 30, bit 2) found within the Main Unit Housekeeping packet. If the value is set to 0 (disabled), the data quality value should be set to 1 (Questionable Data). The values for **d_qual** should be set accordingly under normal conditions:

```

d_qual [0]          0
d_qual [1]          0
...
d_qual [n_sen-1]   0
    
```

10.2.7 Setting of *mode_index* header record element

Mode_index is an array of **i_mode** values. Table 10-3 indicates which telemetry packet fields are to be used to fill this array. The first two status bytes do not reflect the state of the instrument. They are simply used to record the version number of the Main Unit software used to generate the data packets. Status Byte 3 is used to report the number of sweeps that were summed together onboard before transmission in the science packet. For status byte 2 and status bytes 4 through 21, VIDF tables will be defined to hold the values that are associated with each possible state value. The following is a brief explanation of the possible values and the meanings for status byte 2, status byte 4 and status byte 5:

| Status Byte | 2 Software Mode | 4 Energy Summation | 5 Log Compression |
|------------------------------|--------------------------------------------------------------|------------------------------------------|-------------------------|
| State Definitions | 0 = Off 1 = Booting 2 = Safe 3 = Prom 4 = Normal | 0 = 1 step 1 = 2 steps 2 = 4 steps | 0 = off 1 = on |

Status bytes 6 through 21 represent single-bit quantities that are derived from the **ELS sector mask** field (bytes 26-27) within the ELS Science Data Packet. These status bytes indicate which of the sixteen anode sectors were returned in the science data matrix contained in the ELS science data packet. The values for each of the 16 status bytes should be defined as:

- 0 = sector data disabled
- 1 = sector data enabled

Status byte 22 should be set to the same number that is computed for the header record element **n_sen** and reflects the actual number of sectors returned out of the 16 possible sectors. This value is necessary for use in conjunction with some of the calibration data that is returned.

10.3 ELSSCIL Data Record Format

The nominal values assigned to the data record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 10-5 ELSSCIL Data Record Definition

| Field | Description | Value |
|------------|------------------------------------|-------|
| dr_time | initial data time (msec) | DATA |
| spin | azimuthal rate of rotation (msec) | DATA |
| sun_sen | last 0° crossing (msec) | DATA |
| hdr_off | header record offsets in bytes | DATA |
| nss | no. of sensor sets per data record | 1 |
| data_array | data values | DATA |

10.3.1 Setting of *dr_time* data record element

The time tag that will be utilized is the time tag defined in bytes 20-25 (SCET Time) in the ELS Science packet. This value is a 6-byte quantity that represents a CCSDS Unsegmented Time Code. The IDFS production code may need to compute a more accurate timetag based upon the Time Summation scheme (Status Byte 3) being utilized for the current data packet (refer to Section 3.0).

The finest resolution supported by the IDFS data access software is down to the nanosecond. However, **dr_time** holds the relative beginning time of day in milliseconds for the first data element of the first sensor set in the data record. In order to define the time tag down to the nanosecond precision, the VIDF field **nano_defined** must be set to indicate that a nanosecond time adjustment value is contained within the **data_array** matrix.

10.3.2 Setting of *spin* data record element

There are two parameters which must be considered in order to determine the value to store within the **spin** field in the IDFS data record. These two parameters are spin rate and spin direction. The spin rate is determined by the scanner speed. The scanner speed is reported in the **Scanner speed** parameter, which is byte 30, bits 0-1 in the ELS Science packet. The possible scanner speeds are 0 (non spinning, value = 0), 32 second scan (value = 1), 64 second scan (value = 2), and 128 second scan (value = 3). A scan consists of a turning of 180 degrees. The scanner moves forward and backward through an angle of 180 degrees during normal operation.

The spin direction is determined by the scanner direction. The scanner direction is reported in the **Scanner direction** parameter, which is byte 30, bit 2 in the ELS Science packet. The value for scanner direction will either be 0 to indicate a 0-180 direction or 1 to indicate a 180-0 direction.

The data value of **spin** is given in milliseconds per revolution (a revolution is 360 degrees). When the scanner direction is 0 – 180, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | 64 | 64000 |
| 64 second scan | 128 | 128000 |
| 128 second scan | 256 | 256000 |

When the scanner direction is 180 - 0, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | -64 | -64000 |
| 64 second scan | -128 | -128000 |
| 128 second scan | -256 | -256000 |

10.3.3 Setting of *sun_sen* data record element

There are three parameters which must be considered in order to determine the value to store within the **sun_sen** field in the IDFS data record. These three parameters are scanner speed, scanner direction and scanner position. The scanner position is reported in the **Scanner position** parameter, which is byte 31 in the ELS Science packet. The scanner position is reset to 0 when the scanner is pointing in the direction of 0 degrees or less. There are 0.806 degrees per count under normal conditions. This means that at 180 degrees, the scanner position count is 223.

If the scanner speed is 0 (non spinning), then the scanner position marks where the scanner is pointing. Let's call this position **P**. The **sun_sen** value written in the data record is given as degrees times 100 for the non spinning case. Thus, the value written in the **sun_sen** field of the data record is $180 * 100 * P / 223$.

If the scanner speed is not 0, then the scanner direction will need to be examined in order to project where the 0 offset (or 0 degree) location was or will be located with respect to time.

If the scanner direction is from 0-180, the scanner crossed the 0 degree marker some time in the past. For the value of the scanner position **P**, the value was

$$\text{spin} / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **subtracted** from the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the subtraction is negative, this means that the **sun_sen** time crossed the day boundary. To change this to the next 0 degree marker crossing, **add** the value

$$\text{spin} / 2 * (2 - P / 223)$$

to the SCET time.

If the scanner direction is from 180-0, then the scanner will cross the 0 degree marker some time in the future. For the value of scanner position **P**, the value will be

$$\text{abs}(\text{spin}) / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **added** to the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the addition is greater than or equal to 86400 seconds, this means that the **sun_sen** time crossed the day boundary. To change this to the previous 0 degree marker crossing, **subtract** the value

$$\text{abs}(\text{spin}) / 2 * (2 - P / 223)$$

from the SCET time.

10.3.4 Setting of *data_array* data record element

The first four bytes contained within the **data_array** matrix is a nanosecond time adjustment factor that will be used by the IDFS data access software when computing the time tags associated with the data (refer to **dr_time**). The nanosecond time adjustment factor is interpreted as a signed, 4-byte quantity and therefore, can reach a maximum of 2,147,483,647. However, since this value contains the resolution between nanosecond and millisecond precision, the value should be no larger than 999,999. The remainder of the **data_array** for **ELSSCIL** is dependent upon the number of low range sweep steps returned (**n_sample**). Values are packed into the beginning of **data_array** as illustrated below:

| Array Position | IDFS Sensor | Comments |
|-----------------------------------------------------|--------------------|----------|
| data_array [4 thru (N*2)-1+4] | sen_index[0] | N values |
| data_array [(N*2)+4 thru 2(N*2)-1+4] | sen_index[1] | N values |
| ... | ... | ... |
| data_array [(n_sen-1)*(N*2)+4 thru n_sen*(N*2)-1+4] | sen_index[n_sen-1] | N values |

In the table above, the variable **N** represents the value **n_sample**, as defined in header record. Notice that for the indexing specification for **data_array**, the variable **N** is multiplied by 2 since the data itself is 16-bit data and the **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record.

The data for those sectors being returned always starts at byte 32 of the ELS Science Data packet. However, the byte offset from this starting position for each of the anode sectors being returned varies depending upon:

(1) sector mask

The data is laid down starting with the first step for the 1st anode being returned, followed by the first step for the 2nd anode being returned, ..., followed by the first step for the last anode being returned. This pattern is repeated for each of the energy levels (see energy summation below). Data is returned only for those sectors identified within the sector mask, starting with sector 0. For instance, if three even sectors are flagged as being returned – 2, 6, and 10 – sector 2 is considered the 1st sector and sector 10 is considered the last sector within the data matrix.

(2) compression of word size from 16-bits to 8-bits

To determine which word size is being utilized, the data field **Log Compression** (Byte 29, bit 5) must be examined. The IDFS production code must decompress the data before storing it as 16-bit words in the IDFS data record if the indication is that the data has been compressed into 8-bits.

Energy summation must be taken into account when trying to process all 128 ELS steps. The number of energy levels being returned varies from 128 to 32, based upon the energy summation scheme utilized. The IDFS production code will eliminate the energy summation by repeating the value in the appropriate energy step locations.

The calibration data is then placed into **data_array**, at the end of the sector data. The table below shows how the calibration data is organized within **data_array**. The format **C_i** identifies the data as calibration (C) data from calibration set *i*. The value(s) for these calibration sets are written once in the data record and are utilized by all IDFS sensors. The format **C_i / sen_index[X]** identifies the data as calibration (C) data from calibration set *i* that is associated with the IDFS sensor named in **sen_index[X]**. The variable **M** in the table represents the value $4 + n_{sen} * n_{sample} * 2$ and the variable **offset** represents the value $M + 10 + (4 * 2 * k)$, where **k** is the index for the IDFS sensor being processed (**sen_index[k]**), 4 represents the number of calibration values, 2 represents 16-bit values, and 10 represents the number of bytes utilized by the 5 calibration sets that are written only once in the IDFS data record. Notice that the index value is increasing by 2 for **data_array**. The reason for this indexing scheme is based upon the fact that the calibration data is stored as a 16-bit value and **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record.

| Array Position | IDFS Sensor | Comments |
|-----------------------|-------------------------|----------|
| data_array [M] | C0 | 1 value |
| data_array [M+2] | C1 | 1 value |
| data_array [M+4] | C2 | 1 value |
| data_array [M+6] | C3 | 1 value |
| data_array [M+8] | C4 | 1 value |
| data_array [M+10] | C5 / sen_index[0] | 1 value |
| data_array [M+12] | C6 / sen_index[0] | 1 value |
| data_array [M+14] | C7 / sen_index[0] | 1 value |
| data_array [M+16] | C8 / sen_index[0] | 1 value |
| ... | ... | ... |
| data_array [offset] | C5 / sen_index[n_sen-1] | 1 value |
| data_array [offset+2] | C6 / sen_index[n_sen-1] | 1 value |
| data_array [offset+4] | C7 / sen_index[n_sen-1] | 1 value |
| data_array [offset+6] | C8 / sen_index[n_sen-1] | 1 value |

Although the number of sectors and the number of steps returned in telemetry can vary, the data array must be fixed at a maximum size of 16 sectors and 128 steps since the data records are fixed-length records. For the **ELSSCIL** virtual instrument, this would equate to 128 steps * 16 sectors + 4 cal. values * 16 sectors + 5 cal. values per data record, for a total of 2117 elements. This would equate to 4234 bytes since each element is a 16-bit quantity. The IDFS production code must pack the active sensor and calibration data, then pad the remainder of the data record with zero-fill as illustrated in Figure 9 below so that the data record's fixed-length is reached.

| | | | | | | | | |
|-------------|---------|-----|---------------|----|----|-----|---------------|-----------|
| 4-Byte Nsec | S0 Data | ... | Sn_sen-1 Data | C0 | C1 | ... | C8 / Sn_sen-1 | Zero Fill |
|-------------|---------|-----|---------------|----|----|-----|---------------|-----------|

Figure 9 Data Array for ELSSCIL

Calibration set numbers five and six (C5 and C6) – **Low Range Sweep Summation** – equate to a scalar value that represents the summation of the science data over the low range portion of the ELS sweep, one value for each of the sectors being returned. The quantity is actually accumulated in a 32-bit value and then stored as two 16-bit calibration values (MSB and LSB). Calibration set numbers seven and eight (C7 and C8) – **Total Range Sweep Summation** – equate to a scalar value that represents the

summation of the science data over the entire range of the ELS sweep, one value for each of the sectors being returned. The quantity is actually accumulated in a 32-bit value and then stored as two 16-bit calibration values (MSB and LSB). Calibration set numbers three and four (C3 and C4) – **Total Range Anode Summation** – equate to a scalar value which is simply the summation of the **n_sen** values computed for calibration set numbers 7 and 8. The quantity is actually accumulated in a 32-bit value and then stored as two 16-bit calibration values (MSB and LSB).

11.0 Definition for Virtual Instrument **ELSSCIH**

The science data from the high range portion of the ELS power supply form the virtual instrument **ELSSCIH**. The first three characters in the acronym are intended to identify the parent instrument (ELS), the next three characters (SCI) indicate science data is being returned and the last character (H) indicates that the data is from the high range portion of the ELS power supply.

The IDFS sensor definitions for the virtual instrument are defined in Table 11-1 below. Refer to sections 11.1 and 11.3.4 for a more in-depth explanation of the **Packet Byte Position** column information. Some ancillary or calibration data has been stored along with the primary sensor data in the data record and is identified in Table 11-2. The status flag definitions for the virtual instrument are defined in Table 11-3. The VIDF file for this virtual instrument can be found in Appendix A.

The Main Unit produces different types of science packets, based upon the data source. The science packet type identifier can be found in the **Data type** parameter, which is defined in byte 19, bits 4-7 in all science data packets generated by the Main Unit, except for the IMA packets. This parameter identifies the instrument (data source) to which the packet is related. For the ELS engineering information data packet, the value for the **Data type** parameter is 1 and the packets are referred to as ELS telemetry packets.

For each instrument defined, the Main Unit can produce multiple packet subtypes. For the ELS telemetry packets, the science packet subtype identifier can be found in the **ELS packet subtype** parameter, which is defined in byte 19, bits 0-1. For the ELS science data packet, the value for the **ELS packet subtype** parameter is either 1, 2 or 3. In Table 11-1, Table 11-2 and Table 11-3, the heading **Data Subtype** refers to the packet subtype.

Table 11-1 ELSSCIH Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|-------------------|----------------------|------------------|---------------------|--------------------------|--------------------------------------------|
| 0 | ELS Anode 0 | SCI | 1 | 1, 2 or 3 | Data | 32 |
| 1 | ELS Anode 1 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 2 | ELS Anode 2 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 3 | ELS Anode 3 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 4 | ELS Anode 4 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 5 | ELS Anode 5 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 6 | ELS Anode 6 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |

Table 11-1 ELSSCIH Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|-------------------|----------------------|------------------|---------------------|--------------------------|--------------------------------------------|
| 7 | ELS Anode 7 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 8 | ELS Anode 8 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 9 | ELS Anode 9 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 10 | ELS Anode 10 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 11 | ELS Anode 11 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 12 | ELS Anode 12 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 13 | ELS Anode 13 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 14 | ELS Anode 14 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |
| 15 | ELS Anode 15 | SCI | 1 | 1, 2 or 3 | Data | Dependent Upon Setting of Data Sector Mask |

Table 11-2 ELSSCIH Calibration Definition

| Calibration Set | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|------------------------|-----------------------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | ELS Temperature | SCI | 1 | 0 | ELS temperature | 33 |
| 1 | ELS MCP Bias Reference | SCI | 1 | 0 | ELS MCP reference | 34 |
| 2 | ELS MCP Bias Monitor | SCI | 1 | 0 | ELS MCP monitor | 35 |
| 3 | Total Range Anode Summation (MSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 4 | Total Range Anode Summation (LSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 5 | High Range Sweep Summation (MSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 6 | High Range Sweep Summation (LSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |

Table 11-2 ELSSCIH Calibration Definition

| Calibration Set | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|-----------------|-----------------------------------|---------------|-----------|--------------|-------------------|----------------------|
| 7 | Total Range Sweep Summation (MSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |
| 8 | Total Range Sweep Summation (LSB) | SCI | 1 | 1, 2 or 3 | Data | 32 through Last |

Calibration sets 3 through 8 represent data values that are derived and computed in the IDFS production code as opposed to values directly transferred from the telemetry source packets. The reference to “Last” indicates the byte position taken up by the last sector for the last step included in the packet being processed.

Table 11-3 ELSSCIH Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position | Packet Bit Position |
|-------------|-------------------------|---------------|-----------|--------------|--------------------|----------------------|---------------------|
| 0 | SW Version – Upper Byte | SCI | 1 | 1, 2 or 3 | SW Version | 16 | 0-7 |
| 1 | SW Version – Lower Byte | SCI | 1 | 1, 2 or 3 | SW Version | 17 | 0-7 |
| 2 | Software Mode | HSKP | - | - | sw_mode | 106 | 0-7 |
| 3 | Time Summation | SCI | 1 | 1, 2 or 3 | Time compression | 29 | 0-2 |
| 4 | Energy Summation | SCI | 1 | 1, 2 or 3 | Energy compression | 29 | 3-4 |
| 5 | Log Compression | SCI | 1 | 1, 2 or 3 | Log compression | 29 | 5 |
| 6 | Sector 0 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 0 |
| 7 | Sector 1 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 1 |
| 8 | Sector 2 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 2 |
| 9 | Sector 3 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 3 |

Table 11-3 ELSSCIH Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position | Packet Bit Position |
|-------------|-------------------------|---------------|-----------|--------------|-------------------|----------------------|---------------------|
| 10 | Sector 4 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 4 |
| 11 | Sector 5 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 5 |
| 12 | Sector 6 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 6 |
| 13 | Sector 7 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 7 |
| 14 | Sector 8 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 8 |
| 15 | Sector 9 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 9 |
| 16 | Sector 10 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 10 |
| 17 | Sector 11 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 11 |
| 18 | Sector 12 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 12 |
| 19 | Sector 13 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 13 |
| 20 | Sector 14 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 14 |
| 21 | Sector 15 Enable | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | 15 |
| 22 | Number Of Active Anodes | SCI | 1 | 1, 2 or 3 | ELS Sector Mask | 26-27 | N/A |

11.1 Notes for ELSSCIH virtual instrument

For the science data, the data that is produced in one sweep for ELS cannot fit into a single ELS Science Packet when uncompressed packets are transmitted. For this scenario, the data will be split into two packets, with steps 0-63 being packed into one packet (subtype = 2) and steps 64-127 being packed into another packet (subtype = 3). If the data is compressed, all steps (0-127) may be returned within a single packet (subtype = 1). The mode of transmission is known by examining the **ELS packet subtype** field – byte 19, bit 0-1 – contained within the packet. If the data is split between 2 packets, the IDFS production code will acquire both packets before processing the data into an IDFS data record so that all 128 steps are available. Both packets will have the same time tag on packet bytes 6 – 11; however, the sequence count (bytes 2-3, bits 13-0) will be different for these two packets.

The **ELS sector mask** field contained within the ELS science packet identifies which of the sixteen possible anodes are being returned in the data matrix. This field will be a 16-bit word and this field will be set to reflect the subset of anodes that are being returned in the data packet(s). In other words, only those sectors that are being returned will be set to indicate that data is available for those sectors and all other sectors will be set to indicate that no data is available. Figure 10 below shows the ordering of the sector mask, where Anode 15 represents the MSB (bit 15) of the word and Anode 0 represents the LSB (bit 0) of the word.

| | | | | | | | | | | | | | | | |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| Anode 15 | Anode 14 | Anode 13 | Anode 12 | Anode 11 | Anode 10 | Anode 9 | Anode 8 | Anode 7 | Anode 6 | Anode 5 | Anode 4 | Anode 3 | Anode 2 | Anode 1 | Anode 0 |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|

Figure 10 Sector Mask Anode Number / Bit Number Association

For the science data, the data may be transmitted in compressed form. The APAF system will define a software utility library which will contain the decompression algorithms that are to be used to decompress (RICE and Log) the data within the ELS, NPI and NPD Science packets. The parameter **RICE compression** (byte 29, bit 6) needs to be examined in order to determine whether the data has been RICE compressed or not. A value of 1 indicates that RICE compression has been enabled. If the data has been RICE compressed, the data must be uncompressed before any of the other ELS compression / summation schemes are examined. The parameter **Log compression enabled** (byte 29, bit 5) needs to be examined in order to determine whether the 16-bit counts have been converted to 8-bit values. A value of 1 indicates that Log compression has been enabled. The ELS instrument makes use of a summation scheme where the summation of multiple sweeps takes place onboard before the data is transmitted. Refer to Appendix C for an explanation of how the IDFS production code should decompress the data contained within the ELS science packets.

11.1.1 ELS Calibration Data

For calibration sets 0 – 4, there will be one scalar value defined that will be applicable to each of the IDFS sensors that are returned in telemetry. For calibration sets 5 – 8, there will be one scalar value defined for each of the IDFS sensors that are returned in telemetry. Each IDFS sensor corresponds to a specific ELS anode. If all ELS anodes are returned, there will be 16 separate IDFS sensor values to be processed. Calibration sets 3 through 8 represent data values that are derived and computed in the IDFS production code as opposed to values directly transferred from the telemetry source packets. Calibration Sets 5 and 6 – **High Range Sweep Summation** - represent a summation which corresponds to an IDFS sensor which is computed by adding together the science data for the energy steps where the sweep step is in high range. The algorithm is shown below

$$\text{calibration [sensor]} = \sum_0^{\text{sweep length}} \text{data [sensor]}$$

and the data value to be included in the summation will either be: a) 0 if the step is in low range for the sensor or b) science data if the step is in high range for the sensor. The IDFS production code will need to create this summation before expanding and padding the energy sweep to account for energy summation, but after Rice Decompression and bit decompression have been applied. Since the data is a 16-bit value, there is the chance for overflow of a 16-bit calibration value; therefore, the summation is computed using a 32-bit value as the accumulator and the summation value is written into the IDFS data record as two 16-bit calibration values (MSB and LSB).

Calibration Sets 7 and 8 – **Total Range Sweep Summation** - represent a summation which corresponds to an IDFS sensor which is computed by adding together the science data for every energy step in the sweep. The algorithm is shown below

$$\text{calibration [sensor]} = \sum_0^{\text{sweep length}} \text{data [sensor]}$$

and the data value to be included in the summation will be the science data for that sensor. The IDFS production code will need to create this summation before expanding and padding the energy sweep to account for energy summation, but after Rice Decompression and bit decompression have been applied. Since the data is a 16-bit value, there is the chance for overflow of a 16-bit calibration value; therefore, the summation is computed using a 32-bit value as the accumulator and the summation value is written into the IDFS data record as two 16-bit calibration values (MSB and LSB).

Calibration Sets 3 and 4 – **Total Range Anode Summation** - is formed using the **Total Range Sweep Summation** values for each of the IDFS sensors that are processed. The algorithm is shown below

$$\text{calibration} = \sum_0^{\text{no. of IDFS sensors}} \text{Total Range Sweep Summation [sensor]}$$

and the data to be included in the summation will be the data from Calibration Sets 7 and 8 for the IDFS sensor being processed. This summation results in a single scalar value which needs to be written into the data record once. Since the data is a 16-bit value, there is the chance for overflow of a 16-bit calibration value; therefore, the summation is computed using a 32-bit value as the accumulator and the summation value is written into the IDFS data record as two 16-bit calibration values (MSB and LSB).

11.2 ELSSCIH Header Record Format

The nominal values assigned to the header record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 11-4 ELSSCIH Header Record Definition

| Field | Description | Value |
|--------------|---------------------------------------------------|-------|
| hdr_len | header record size in bytes | DATA |
| year | year as YYYY | DATA |
| day | day of year as DDD | DATA |
| time_units | time base for data_accum | -6 |
| i_mode | no. of status flags returned in mode_index | 23 |
| data_accum | sample accumulation time | 28125 |
| data_lat | sample dead time (μ sec) | 3125 |
| swp_reset | sweep dead time (μ sec) | DATA |
| sen_reset | sensor set dead time (μ sec) | 0 |
| n_sen | no. of sensors returned in sensor set | DATA |
| n_sample | no. of data samples per sensor | DATA |
| scan_index | index into scan dependent tables | DATA |
| sensor_index | VIDF sensors returned in sensor set | DATA |
| d_qual | data quality flag per sensor | DATA |
| mode_index | status flag values | DATA |

11.2.1 Setting of *swp_reset* header record element

The amount of dead time necessary for the ELS instrument to reset itself between successive sweeps is defined in order to account for the missing flyback step. This value is dependent upon the Energy Compression bit returned in the ELS science packets. The Energy Compression bit indicates how many energy steps have been summed over for each value returned. The value of **swp_reset** is given in microsecond :

| Energy Compression value | No. Of Steps Summed | swp_reset value |
|--------------------------|---------------------|-----------------|
| 0 | 1 | 31250 |
| 1 | 2 | 62500 |
| 2 | 4 | 125000 |

11.2.2 Setting of *n_sen* header record element

The maximum number of anode sectors being returned by ELS is 16; however, data for all sectors may not always be returned. The field **ELS sector mask**, bytes 26-27, within the ELS Science Data Packet indicates which anode sectors are being returned. This field will hold a 16-bit filter mask selection word that identifies which of the sixteen anode sectors are returned in the science data matrix contained in the packet. The values for each bit should be defined as:

0 = block sector data

1 = transmit sector data

The value for **n_sen** should be set to reflect the actual number of sectors returned out of the 16 possible sectors.

11.2.3 Setting of *n_sample* header record element

There should be 128 values for the deflection reference (command) contained in a single packet, representing a single ELS sweep. The sweep may be all from the low range, all from the high range or a mixture of low range and high range. Due to this variable nature, **n_sample** cannot be pre-assigned the value of 128. The IDFS production code must look at the range bit for each individual element of the sweep to determine where that element will be recorded. The value for **n_sample** is set to the tally for the number of elements that are associated with the high range. The value for **n_sample** should not include the data for the last energy compressed step within the ELS sweep since the last step is the flyback step and the data associated with the flyback step is considered invalid science data. The energy steps to ignore are based upon the setting for energy compression and are indicated in the table below:

| Energy Compression value | Energy Steps to Ignore |
|--------------------------|------------------------|
| 0 | 127 |
| 1 | 126 and 127 |
| 2 | 124, 125, 126 and 127 |

11.2.4 Setting of *scan_index* header record element

Scan_index is an array of **n_sample** values. The **scan_index** values indicate the sweep step number associated with each of the data samples returned per sensor. The values for **scan_index** are to be taken from the ELS Engineering Information packet, utilizing the 128 deflection reference values, extracting only those that are associated with the high range of the ELS sweep. These values indicate the correct step the value was taken at (value ranging from 0 to 4095). Again, the flyback data is not included.

11.2.5 Setting of *sensor_index* header record element

Sensor_index is an array of **n_sen** values. The values should be set to reflect which sectors are contained in the data array. For example, if the sector mask was set to

0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1

this would indicate that data for sectors 0, 2, and 4 were being returned. The value for **n_sen** should be set to 3 and the values for **sensor_index** should be set as follows:

```

sensor_index [0]    0
sensor_index [1]    2
sensor_index [2]    4
    
```

11.2.6 Setting of *d_qual* header record element

D_qual is an array of **n_sen** values. According to the VIDF file outlined to date, there are currently 5 states defined for data quality, as shown in the table below:

| Value | Meaning | Criteria for Setting of Value |
|-------|-------------------|-------------------------------------------------------------------------------------------------|
| 0 | Good Data | default setting |
| 1 | Questionable Data | +30V monitor value is not within tolerance |
| 2 | Invalid Data | +12V, -12V, +5V or -5V monitor value is not within tolerance or the high voltage is not enabled |
| 3 | Bad Data | +30V monitor value and (+12V, -12V, +5V or -5V monitor value) are not within tolerance |
| 4 | Unknown State | Time related Main Unit Housekeeping packet is not available |

The setting of the data quality value to Questionable is dependent upon the state of the high voltage, which is specified in the single-bit parameter **els_enable_hv** (byte 30, bit 2) found within the Main Unit Housekeeping packet. If the value is set to 0 (disabled), the data quality value should be set to 1 (Questionable Data). The values for **d_qual** should be set accordingly under normal conditions:

```

d_qual [0]          0
d_qual [1]          0
...
d_qual [n_sen-1]    0
    
```

11.2.7 Setting of *mode_index* header record element

Mode_index is an array of **i_mode** values. Table 11-3 indicates which telemetry packet fields are to be used to fill this array. The first two status bytes do not reflect the state of the instrument. They are simply used to record the version number of the Main Unit software used to generate the data packets. Status Byte 3 is used to report the number of sweeps that were summed together onboard before transmission in the science packet. For status byte 2 and status bytes 4 through 21, VIDF tables will be defined to hold the values that are associated with each possible state value. The following is a brief explanation of the possible values and the meanings for status byte 2, status byte 4 and status byte 5:

| Status Byte | 2 Software Mode | 4 Energy Summation | 5 Log Compression |
|------------------------------|--------------------------------------------------------------|------------------------------------------|-------------------------|
| State Definitions | 0 = Off 1 = Booting 2 = Safe 3 = Prom 4 = Normal | 0 = 1 step 1 = 2 steps 2 = 4 steps | 0 = off 1 = on |

Status bytes 6 through 21 represent single-bit quantities that are derived from the **ELS sector mask** field (bytes 26-27) within the ELS Science Data Packet. These status bytes indicate which of the sixteen anode sectors were returned in the science data matrix contained in the ELS science data packet. The values for each of the 16 status bytes should be defined as:

- 0 = sector data disabled
- 1 = sector data enabled

Status byte 22 should be set to the same number that is computed for the header record element **n_sen** and reflects the actual number of sectors returned out of the 16 possible sectors. This value is necessary for use in conjunction with some of the calibration data that is returned.

11.3 ELSSCIH Data Record Format

The nominal values assigned to the data record elements that can be pre-assigned, that is, which are not dynamically changing during flight, are indicated in the table below.

Table 11-5 ELSSCIH Data Record Definition

| Field | Description | Value |
|------------|------------------------------------|-------|
| dr_time | initial data time (msec) | DATA |
| spin | azimuthal rate of rotation (msec) | DATA |
| sun_sen | last 0° crossing (msec) | DATA |
| hdr_off | header record offsets in bytes | DATA |
| nss | no. of sensor sets per data record | 1 |
| data_array | data values | DATA |

11.3.1 Setting of *dr_time* data record element

The time tag that will be utilized is the time tag defined in bytes 20-25 (SCET Time) in the ELS Science packet. This value is a 6-byte quantity that represents a CCSDS Unsegmented Time Code. The IDFS production code may need to compute a more accurate timetag based upon the Time Summation scheme (Status Byte 3) being utilized for the current data packet (refer to Section 3.0).

The finest resolution supported by the IDFS data access software is down to the nanosecond. However, **dr_time** holds the relative beginning time of day in milliseconds for the first data element of the first sensor set in the data record. In order to define the time tag down to the nanosecond precision, the VIDF field **nano_defined** must be set to indicate that a nanosecond time adjustment value is contained within the **data_array** matrix.

11.3.2 Setting of *spin* data record element

There are two parameters which must be considered in order to determine the value to store within the **spin** field in the IDFS data record. These two parameters are spin rate and spin direction. The spin rate is determined by the scanner speed. The scanner speed is reported in the **Scanner speed** parameter, which is byte 30, bits 0-1 in the ELS Science packet. The possible scanner speeds are 0 (non spinning, value = 0), 32 second scan (value = 1), 64 second scan (value = 2), and 128 second scan (value = 3). A scan consists of a turning of 180 degrees. The scanner moves forward and backward through an angle of 180 degrees during normal operation.

The spin direction is determined by the scanner direction. The scanner direction is reported in the **Scanner direction** parameter, which is byte 30, bit 2 in the ELS Science packet. The value for scanner direction will either be 0 to indicate a 0-180 direction or 1 to indicate a 180-0 direction.

The data value of **spin** is given in milliseconds per revolution (a revolution is 360 degrees). When the scanner direction is 0 – 180, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | 64 | 64000 |
| 64 second scan | 128 | 128000 |
| 128 second scan | 256 | 256000 |

When the scanner direction is 180 - 0, the following values should be used:

| Scanner Speed | Seconds per Revolution | Milliseconds per Revolution |
|------------------|------------------------|-----------------------------|
| 0 (non spinning) | 0 | 0 |
| 32 second scan | -64 | -64000 |
| 64 second scan | -128 | -128000 |
| 128 second scan | -256 | -256000 |

11.3.3 Setting of *sun_sen* data record element

There are three parameters which must be considered in order to determine the value to store within the **sun_sen** field in the IDFS data record. These three parameters are scanner speed, scanner direction and scanner position. The scanner position is reported in the **Scanner position** parameter, which is byte 31 in the ELS Science packet. The scanner position is reset to 0 when the scanner is pointing in the direction of 0 degrees or less. There are 0.806 degrees per count under normal conditions. This means that at 180 degrees, the scanner position count is 223.

If the scanner speed is 0 (non spinning), then the scanner position marks where the scanner is pointing. Let's call this position **P**. The **sun_sen** value written in the data record is given as degrees times 100 for the non spinning case. Thus, the value written in the **sun_sen** field of the data record is $180 * 100 * P / 223$.

If the scanner speed is not 0, then the scanner direction will need to be examined in order to project where the 0 offset (or 0 degree) location was or will be located with respect to time.

If the scanner direction is from 0-180, the scanner crossed the 0 degree marker some time in the past. For the value of the scanner position **P**, the value was

$$\text{spin} / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **subtracted** from the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the subtraction is negative, this means that the **sun_sen** time crossed the day boundary. To change this to the next 0 degree marker crossing, **add** the value

$$\text{spin} / 2 * (2 - P / 223)$$

to the SCET time.

If the scanner direction is from 180-0, then the scanner will cross the 0 degree marker some time in the future. For the value of scanner position **P**, the value will be

$$\text{abs}(\text{spin}) / 2 * P / 223$$

where **spin** is the value computed for the spin rate, as explained in the previous section. Since the spin rate is given in terms of milliseconds per revolution, the result of the equation above gives the number of milliseconds that should be **added** to the SCET time for the start of the sweep in this scan. The SCET time tag represents the **dr_time** in the data record. If the time that is computed based upon the addition is greater than or equal to 86400 seconds, this means that the **sun_sen** time crossed the day boundary. To change this to the previous 0 degree marker crossing, **subtract** the value

$$\text{abs}(\text{spin}) / 2 * (2 - P / 223)$$

from the SCET time.

11.3.4 Setting of *data_array* data record element

The first four bytes contained within the **data_array** matrix is a nanosecond time adjustment factor that will be used by the IDFS data access software when computing the time tags associated with the data (refer to **dr_time**). The nanosecond time adjustment factor is interpreted as a signed, 4-byte quantity and therefore, can reach a maximum of 2,147,483,647. However, since this value contains the resolution between nanosecond and millisecond precision, the value should be no larger than 999,999. The remainder of the **data_array** for **ELSSCIH** is dependent upon the number of high range sweep steps returned (**n_sample**). Values are packed into the beginning of **data_array** as illustrated below:

| Array Position | IDFS Sensor | Comments |
|-----------------------------------------------------|--------------------|----------|
| data_array [4 thru (N*2)-1+4] | sen_index[0] | N values |
| data_array [(N*2)+4 thru 2(N*2)-1+4] | sen_index[1] | N values |
| ... | ... | ... |
| data_array [(n_sen-1)*(N*2)+4 thru n_sen*(N*2)-1+4] | sen_index[n_sen-1] | N values |

In the table above, the variable **N** represents the value **n_sample**, as defined in header record. Notice that for the indexing specification for **data_array**, the variable **N** is multiplied by 2 since the data itself is 16-bit data and the **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record.

The data for those sectors being returned always starts at byte 32 of the ELS Science Data packet. However, the byte offset from this starting position for each of the anode sectors being returned varies depending upon:

(1) sector mask

The data is laid down starting with the first step for the 1st anode being returned, followed by the first step for the 2nd anode being returned, ..., followed by the first step for the last anode being returned. This pattern is repeated for each of the energy levels (see energy summation below). Data is returned only for those sectors identified within the sector mask, starting with sector 0. For instance, if three even sectors are flagged as being returned – 2, 6, and 10 – sector 2 is considered the 1st sector and sector 10 is considered the last sector within the data matrix.

(2) compression of word size from 16-bits to 8-bits

To determine which word size is being utilized, the data field **Log Compression** (Byte 29, bit 5) must be examined. The IDFS production code must decompress the data before storing it as 16-bit words in the IDFS data record if the indication is that the data has been compressed into 8-bits.

Energy summation must be taken into account when trying to process all 128 ELS steps. The number of energy levels being returned varies from 128 to 32, based upon the energy summation scheme utilized. The IDFS production code will eliminate the energy summation by repeating the value in the appropriate energy step locations.

The calibration data is then placed into **data_array**, at the end of the sector data. The table below shows how the calibration data is organized within **data_array**. The format **C_i** identifies the data as calibration (C) data from calibration set *i*. The value(s) for these calibration sets are written once in the data record and are utilized by all IDFS sensors. The format **C_i / sen_index[X]** identifies the data as calibration (C) data from calibration set *i* that is associated with the IDFS sensor named in **sen_index[X]**. The variable **M** in the table represents the value $4 + n_{sen} * n_{sample} * 2$ and the variable **offset** represents the value $M + 10 + (4 * 2 * k)$, where **k** is the index for the IDFS sensor being processed (**sen_index[k]**), 4 represents the number of calibration values, 2 represents 16-bit values, and 10 represents the number of bytes utilized by the 5 calibration sets that are written only once in the IDFS data record. Notice that the index value is increasing by 2 for **data_array**. The reason for this indexing scheme is based upon the fact that the calibration data is stored as a 16-bit value and **data_array** is generically assigned as an unsigned character (8 bits) within the IDFS data record.

| Array Position | IDFS Sensor | Comments |
|-----------------------|-------------------------|----------|
| data_array [M] | C0 | 1 value |
| data_array [M+2] | C1 | 1 value |
| data_array [M+4] | C2 | 1 value |
| data_array [M+6] | C3 | 1 value |
| data_array [M+8] | C4 | 1 value |
| data_array [M+10] | C5 / sen_index[0] | 1 value |
| data_array [M+12] | C6 / sen_index[0] | 1 value |
| data_array [M+14] | C7 / sen_index[0] | 1 value |
| data_array [M+16] | C8 / sen_index[0] | 1 value |
| ... | ... | ... |
| data_array [offset] | C5/ sen_index[n_sen-1] | 1 value |
| data_array [offset+2] | C6 / sen_index[n_sen-1] | 1 value |
| data_array [offset+4] | C7 / sen_index[n_sen-1] | 1 value |
| data_array [offset+6] | C8 / sen_index[n_sen-1] | 1 value |

Although the number of sectors and the number of steps returned in telemetry can vary, the data array must be fixed at a maximum size of 16 sectors and 128 steps since the data records are fixed-length records. For the **ELSSCIH** virtual instrument, this would equate to 128 steps * 16 sectors + 4 cal. values * 16 sectors + 5 cal. values per data record, for a total of 2117 elements. This would equate to 4234 bytes since each element is a 16-bit quantity. The IDFS production code must pack the active sensor and calibration data, then pad the remainder of the data record with zero-fill as illustrated in Figure 11 below so that the data record's fixed-length is reached.

| | | | | | | | | |
|-------------|---------|-----|---------------|----|----|-----|---------------|-----------|
| 4-Byte Nsec | S0 Data | ... | Sn_sen-1 Data | C0 | C1 | ... | C8 / Sn_sen-1 | Zero Fill |
|-------------|---------|-----|---------------|----|----|-----|---------------|-----------|

Figure 11 Data Array for ELSSCIH

Calibration set numbers five and six (C5 and C6) – **High Range Sweep Summation** – equate to a scalar value that represents the summation of the science data over the high range portion of the ELS sweep, one value for each of the sectors being returned. The quantity is actually accumulated in a 32-bit value and then stored as two 16-bit calibration values (MSB and LSB). Calibration set numbers seven and eight (C7 and C8) – **Total Range Sweep Summation** – equate to a scalar value that represents the summation of the science data over the entire range of the ELS sweep, one value for each

of the sectors being returned. The quantity is actually accumulated in a 32-bit value and then stored as two 16-bit calibration values (MSB and LSB). Calibration set numbers three and four (C3 and C4) – **Total Range Anode Summation** – equate to a scalar value which is simply the summation of the **n_sen** values computed for calibration set numbers 7 and 8. The quantity is actually accumulated in a 32-bit value and then stored as two 16-bit calibration values (MSB and LSB).

Appendix A - Virtual Instrument Description Files (VIDF) for ELS

ELSENG8 VIDF File

```

vidf v3_ELSENG8 {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft    */
    string experiment = "ASPERA-3";     /* exp_desc     */
    string instrument = "ELS";          /* inst_desc    */
    string contact = "Dr. Rickard Lundin";
    string contact = "Swedish Institute of Space Physics";
    string contact = "Box 812 S-981 28.Kiruna";
    string contact = "Sweden";
    string contact = "rickard@irf.se";

/*
* This VIDF contains 8-bit engineering monitor and associated
* reference values for ELS. For the flight unit, the maximum
* MCP voltage is 3000V.
*
* The following is a list of tables which are in this vidf
* TABLE 0: Polynomial coefficients for rescaling a 5V max value
*           to the 4.5V monitor or a 5V value
* TABLE 1: Polynomial coefficients which converts to degC or
*           volts.
* TABLE 2: Polynomial coefficients which convert to current
* TABLE 3: ASCII definitions of status states
*
* The following are units which can be derived from the tables.
* The format is to give the tables applied followed by the
* operations and unit definition
*
* DATA TYPE  TABLES          OPERS          UNIT
* Sensor      0                0            Voltage
* Sensor      2                0            current (microAmp)
* Sensor      1                0            Temperature/voltage
*
* Data_len is set using the equation
* 20 + 5 sensors + 4 (nanosecond timing)
*/
    int s_year = 2003;                /* ds_year      */
    int s_day = 1;                    /* ds_day       */
    int s_msec = 0;                   /* ds_msec      */
    int s_usec = 0;                   /* ds_usec      */
    int e_year = 2010;                /* de_year      */
    int e_day = 1;                    /* de_day       */
    int e_msec = 0;                   /* de_msec      */
    int e_usec = 0;                   /* de_usec      */
    int smp_id = 2;                   /* smp_id       */
    int sen_mode = 2;                 /* sen_mode     */
    int n_qual = 5;                   /* n_qual       */
    int n_cal_sets = 0;               /* cal_sets     */
    int n_tbls = 4;                   /* num_tbls     */

```

```

int n_consts = 0;          /* num_consts */
int n_status = 3;        /* status */
int n_sensors = 5;       /* sen */
int swp_len = 1;         /* swp_len */
int max_nss = 1;         /* max_nss */
int data_len = 29;       /* data_len */
int fill_flg = 0;        /* fill_flg */
int da_method = 0;       /* da_method */
int nano_defined = 1;    /* nsec timetag */
struct Status0 {
    string name = "Software Version - Upper Byte"; /* name */
    int state = 255; /* state */
};
struct Status1 {
    string name = "Software Version - Lower Byte"; /* name */
    int state = 255; /* state */
};
struct Status2 {
    string name = "Software Mode"; /* name */
    int state = 5; /* state */
};
string qual_names = "Good Data"; /* name */
string qual_names = "Questionable Data"; /* name */
string qual_names = "Invalid Data"; /* name */
string qual_names = "Bad Data"; /* name */
string qual_names = "Unknown State"; /* name */
struct Sensor0 {
    string name = "-5V Screen Grid Reference"; /* name */
    int d_type = 0; /* d_type */
    int status = 1; /* status */
    int tdw_len = 8; /* tdw_len */
    int time_offset = 0; /* time_offset */
};
struct Sensor1 {
    string name = "-5V Screen Grid Monitor"; /* name */
    int d_type = 0; /* d_type */
    int status = 1; /* status */
    int tdw_len = 8; /* tdw_len */
    int time_offset = 0; /* time_offset */
};
struct Sensor2 {
    string name = "MCP Bias Reference"; /* name */
    int d_type = 0; /* d_type */
    int status = 1; /* status */
    int tdw_len = 8; /* tdw_len */
    int time_offset = 0; /* time_offset */
};
struct Sensor3 {
    string name = "MCP Bias Monitor"; /* name */
    int d_type = 0; /* d_type */
    int status = 1; /* status */
    int tdw_len = 8; /* tdw_len */
    int time_offset = 0; /* time_offset */
};
struct Sensor4 {

```

```

        string name = "ELS Temperature Monitor";           /* name           */
        int d_type = 0;                                   /* d_type          */
        int status = 1;                                  /* status          */
        int tdw_len = 8;                                 /* tdw_len         */
        int time_offset = 0;                             /* time_offset     */
    };
    struct Table0 {
        int tbl_sca_sz = 10;                             /* tbl_sca_sz     */
        int tbl_ele_sz = 10;                             /* tbl_ele_sz     */
        int tbl_type = 0;                                /* tbl_type       */
    };
/*
* Table 0
* This table contains the sets of polynomial coefficients which
* converts sensor (4) telemetry to voltage and sensor (0 - 3)
* telemetry to the analog control voltage.
*/
        int tbl_var = 0;                                 /* tbl_var        */
        int tbl_expand = 0;                             /* tbl_expand     */
        int crit_act_sz = 0;                            /* crit_act_sz    */
        int format [5] = {2, 2, 2, 2, 2};
        int offset [5] = {6, 8, 2, 4, 0};
        int scale [10] = {                               /* scale factor */
            0, -8, 0, -8, 0, -8, 0, -8, -9, -9};        /* 0000 - 0009 */
        int values [10] = {                             /* values        */
            0, 1960784, 0, 1960784, 0, 1764706, 0,    /* 0000 - 0006 */
            -1960784, -294659229, -18452317};         /* 0007 - 0009 */
    };
    struct Table1 {
        int tbl_sca_sz = 10;                             /* tbl_sca_sz     */
        int tbl_ele_sz = 10;                             /* tbl_ele_sz     */
        int tbl_type = 0;                                /* tbl_type       */
    };
/*
* Table 1
* This table contains the sets of polynomial coefficients which
* converts sensor (4) telemetry to degrees C and sensor (0-3)
* telemetry to volts.
*/
        int tbl_var = 0;                                 /* tbl_var        */
        int tbl_expand = 0;                             /* tbl_expand     */
        int crit_act_sz = 0;                            /* crit_act_sz    */
        int format [5] = {2, 2, 2, 2, 2};
        int offset [5] = {6, 8, 2, 4, 0};
        int scale [10] = {                               /* scale factor */
            -1, -6, 0, -4, 0, -4, 0, -8, -9, -9};     /* 0000 - 0009 */
        int values [10] = {                             /* values        */
            -2732, 1620483, 0, 117647, 0, 117647, 0, /* 0000 - 0006 */
            -1960784, -294659229, -18452317};        /* 0007 - 0009 */
    };
    struct Table2 {
        int tbl_sca_sz = 2;                             /* tbl_sca_sz     */
        int tbl_ele_sz = 2;                             /* tbl_ele_sz     */
        int tbl_type = 0;                                /* tbl_type       */
    };
/*
* Table 2

```

```

* This table contains the sets of polynomial coefficients which
* converts sensor (4) telemetry to microamp.
*/
    int tbl_var = 0; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [5] = {-1, -1, -1, -1, 2};
    int offset [5] = {-1, -1, -1, -1, 0};
    int scale [2] = {0, -6}; /* scale factor */
    int values [2] = {0, 1620483}; /* values */
};
struct Table3 {
    int tbl_sca_sz = 0; /* tbl_sca_sz */
    int tbl_ele_sz = 5; /* tbl_ele_sz */
    int tbl_type = 1; /* tbl_type */
};
/*
* Table 3
* ASCII definitions of the status states
*/
    int tbl_var = 4; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_ele */
    int format [3] = {-1, -1, 0}; /* format */
    int offset [3] = {-1, -1, 0}; /* offsets */
    string values [5] = { /* values */
        "Undefined", "Booting", "Safe", /* 000 - 002 */
        "Prom", "Normal" /* 003 - 004 */
    };
};
}

```

ELSENGS VIDF File

```

vidf v3_ELSENGS {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft   */
    string experiment = "ASPERA-3";    /* exp_desc    */
    string instrument = "ELS";         /* inst_desc   */
    string contact = "Dr. Rickard Lundin";
    string contact = "Swedish Institute of Space Physics";
    string contact = "Box 812 S-981 28.Kiruna";
    string contact = "Sweden";
    string contact = "rickard@irf.se";

/*
 * This VIDF contains the 1-bit status indicators that are
 * of interest to aid in the understanding of the current
 * operational status of ELS.
 *
 * The following is a list of tables which are in this vidf
 *   TABLE 0: ASCII definitions of sensor states
 *   TABLE 1: ASCII definitions of status states
 *
 * Data_len is set using the equation
 *   20 + (6 sensors / 8 sensors per byte) + 4 (nanosecond timing)
 *
 */
    int s_year = 2003;                  /* ds_year      */
    int s_day = 1;                      /* ds_day       */
    int s_msec = 0;                     /* ds_msec     */
    int s_usec = 0;                     /* ds_usec     */
    int e_year = 2010;                  /* de_year     */
    int e_day = 1;                      /* de_day      */
    int e_msec = 0;                     /* de_msec     */
    int e_usec = 0;                     /* de_usec     */
    int smp_id = 2;                     /* smp_id      */
    int sen_mode = 2;                   /* sen_mode    */
    int n_qual = 4;                     /* n_qual      */
    int n_cal_sets = 0;                  /* cal_sets    */
    int n_tbls = 2;                      /* num_tbls    */
    int n_consts = 0;                   /* num_consts  */
    int n_status = 3;                   /* status      */
    int n_sensors = 6;                  /* sen         */
    int swp_len = 1;                    /* swp_len     */
    int max_nss = 1;                    /* max_nss     */
    int data_len = 25;                  /* data_len    */
    int fill_flag = 0;                  /* fill_flg    */
    int da_method = 0;                  /* da_method   */
    int nano_defined = 1;               /* nsec timetag */
    struct Status0 {
        string name = "Software Version - Upper Byte"; /* name      */
        int state = 255;                    /* state     */
    };
    struct Status1 {
        string name = "Software Version - Lower Byte"; /* name      */

```

```

        int state = 255;                                /* state */
};
struct Status2 {
    string name = "Software Mode";                     /* name */
    int state = 5;                                     /* state */
};
string qual_names = "Good Data";                       /* name */
string qual_names = "Questionable Data";              /* name */
string qual_names = "Invalid Data";                   /* name */
string qual_names = "Bad Data";                       /* name */
struct Sensor0 {
    string name = "ELS Enable High Voltage";           /* name */
    int d_type = 0;                                    /* d_type */
    int status = 1;                                    /* status */
    int tdw_len = 1;                                   /* tdw_len */
    int time_offset = 0;                               /* time_offset */
};
struct Sensor1 {
    string name = "+30V Enable";                       /* name */
    int d_type = 0;                                    /* d_type */
    int status = 1;                                    /* status */
    int tdw_len = 1;                                   /* tdw_len */
    int time_offset = 0;                               /* time_offset */
};
struct Sensor2 {
    string name = "-12V Enable";                      /* name */
    int d_type = 0;                                    /* d_type */
    int status = 1;                                    /* status */
    int tdw_len = 1;                                   /* tdw_len */
    int time_offset = 0;                               /* time_offset */
};
struct Sensor3 {
    string name = "+12V Enable";                      /* name */
    int d_type = 0;                                    /* d_type */
    int status = 1;                                    /* status */
    int tdw_len = 1;                                   /* tdw_len */
    int time_offset = 0;                               /* time_offset */
};
struct Sensor4 {
    string name = "-5V Enable";                       /* name */
    int d_type = 0;                                    /* d_type */
    int status = 1;                                    /* status */
    int tdw_len = 1;                                   /* tdw_len */
    int time_offset = 0;                               /* time_offset */
};
struct Sensor5 {
    string name = "+5V Enable";                       /* name */
    int d_type = 0;                                    /* d_type */
    int status = 1;                                    /* status */
    int tdw_len = 1;                                   /* tdw_len */
    int time_offset = 0;                               /* time_offset */
};
struct Table0 {
    int tbl_sca_sz = 0;                                /* tbl_sca_sz */
    int tbl_ele_sz = 2;                                /* tbl_ele_sz */
};

```

```

        int tbl_type = 1;                                /* tbl_type    */
/*
* Table 0
* This table is an ASCII look-up table which indicates what
* the settings of the bit status monitors represents.
*/
        int tbl_var = 0;                                /* tbl_var     */
        int tbl_expand = 0;                             /* tbl_expand  */
        int crit_act_sz = 0;                             /* crit_act_sz */
        int format [6] = {
            0, 0, 0, 0, 0, 0                             /* format     */
        };                                               /* 000 - 005  */
        int offset [6] = {
            0, 0, 0, 0, 0, 0                             /* offsets    */
        };                                               /* 000 - 005  */
        string values [2] = {"Disabled", "Enabled"};    /* values     */
};
struct Table1 {
        int tbl_sca_sz = 0;                              /* tbl_sca_sz  */
        int tbl_ele_sz = 5;                              /* tbl_ele_sz  */
        int tbl_type = 1;                                /* tbl_type    */
/*
* Table 1
* ASCII definitions of the status states
*/
        int tbl_var = 4;                                /* tbl_var     */
        int tbl_expand = 0;                             /* tbl_expand  */
        int crit_act_sz = 0;                             /* crit_act_ele */
        int format [3] = {-1, -1, 0};                  /* format     */
        int offset [3] = {-1, -1, 0};                  /* offsets    */
        string values [5] = {
            "Undefined", "Booting", "Safe",
            "Prom", "Normal"                             /* 000 - 002  */
        };                                               /* 003 - 004  */
};
}

```

ELSSWPL VIDF File

```

vidf v3_ELSSWPL {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft   */
    string experiment = "ASPERA-3";     /* exp_desc    */
    string instrument = "ELS";          /* inst_desc   */
    string contact = "Dr. Rickard Lundin";
    string contact = "Swedish Institute of Space Physics";
    string contact = "Box 812 S-981 28.Kiruna";
    string contact = "Sweden";
    string contact = "rickard@irf.se";
/*
* This virtual contains low range sweep data for ELS. The data
* values stored are the power supply reference and monitor of that
* setting, as returned in the ELS Science packets. The reference
* value is the plate voltage that is being requested. The monitor
* value is the voltage the deflection plates achieved. The low
* range voltage values are stored in the correct location within
* the sweep. The sweep length is 128 values.
*
* Units for each of the unique anodes of ELS require specific
* sensitivity and resolution values. These values are used to
* translate the control voltage or monitor voltage into the energy
* unit since calibration to energy is dependent of the individual
* anode characteristics.
*
* The following is a list of tables which are in this vidf
* TABLE 0: Polynomial coefficients which places the data into
* math buffer.
* TABLE 1: Polynomial coefficients which converts data into
* fraction of voltage range.
* TABLE 2: Maximum control input voltage [volts].
* TABLE 3: Maximum deflection Low range voltage [volts].
* TABLE 4: Anode 00 K-factor [eV/Volt].
* TABLE 5: Anode 01 K-factor [eV/Volt].
* TABLE 6: Anode 02 K-factor [eV/Volt].
* TABLE 7: Anode 03 K-factor [eV/Volt].
* TABLE 8: Anode 04 K-factor [eV/Volt].
* TABLE 9: Anode 05 K-factor [eV/Volt].
* TABLE 10: Anode 06 K-factor [eV/Volt].
* TABLE 11: Anode 07 K-factor [eV/Volt].
* TABLE 12: Anode 08 K-factor [eV/Volt].
* TABLE 13: Anode 09 K-factor [eV/Volt].
* TABLE 14: Anode 10 K-factor [eV/Volt].
* TABLE 15: Anode 11 K-factor [eV/Volt].
* TABLE 16: Anode 12 K-factor [eV/Volt].
* TABLE 17: Anode 13 K-factor [eV/Volt].
* TABLE 18: Anode 14 K-factor [eV/Volt].
* TABLE 19: Anode 15 K-factor [eV/Volt].
* TABLE 20: Anode 00 Detector Resolution [fractionan number]
* TABLE 21: Anode 01 Detector Resolution [fractionan number]

```

```

* TABLE 22: Anode 02 Detector Resolution [fractionan number]
* TABLE 23: Anode 03 Detector Resolution [fractionan number]
* TABLE 24: Anode 04 Detector Resolution [fractionan number]
* TABLE 25: Anode 05 Detector Resolution [fractionan number]
* TABLE 26: Anode 06 Detector Resolution [fractionan number]
* TABLE 27: Anode 07 Detector Resolution [fractionan number]
* TABLE 28: Anode 08 Detector Resolution [fractionan number]
* TABLE 29: Anode 09 Detector Resolution [fractionan number]
* TABLE 30: Anode 10 Detector Resolution [fractionan number]
* TABLE 31: Anode 11 Detector Resolution [fractionan number]
* TABLE 32: Anode 12 Detector Resolution [fractionan number]
* TABLE 33: Anode 13 Detector Resolution [fractionan number]
* TABLE 34: Anode 14 Detector Resolution [fractionan number]
* TABLE 35: Anode 15 Detector Resolution [fractionan number]
* TABLE 36: Polynomial coefficients which converts data
*             into monitor fraction of voltage range.
* TABLE 37: ASCII definitions of status states
*

```

```

* The following are units which can be derived from the tables.
* The format is to give the tables applied followed by the
* operations and unit definition
*

```

| DATA TYPE | TABLES | OPERS | UNIT |
|---------------|----------|---------|--------------------|
| Sensors (0-1) | 0 | 0 | deflection index |
| Sensors (0-1) | 1 | 0 | fraction-unitless |
| Sensors (0-1) | 1,2 | 0,3 | control voltage |
| Sensors (1) | 1,36 | 0,3 | monitor voltage |
| Sensors (0-1) | 1,3 | 0,3 | deflection volt |
| Sensors (0-1) | 1,3,4 | 0,3,3 | Anode 00 Energy eV |
| Sensors (0-1) | 1,3,4,20 | 0,3,3,3 | Anode 00 Width eV |
| Sensors (0-1) | 1,3,5 | 0,3,3 | Anode 01 Energy eV |
| Sensors (0-1) | 1,3,4,21 | 0,3,3,3 | Anode 01 Width eV |
| Sensors (0-1) | 1,3,6 | 0,3,3 | Anode 02 Energy eV |
| Sensors (0-1) | 1,3,4,22 | 0,3,3,3 | Anode 02 Width eV |
| Sensors (0-1) | 1,3,7 | 0,3,3 | Anode 03 Energy eV |
| Sensors (0-1) | 1,3,4,23 | 0,3,3,3 | Anode 03 Width eV |
| Sensors (0-1) | 1,3,8 | 0,3,3 | Anode 04 Energy eV |
| Sensors (0-1) | 1,3,4,24 | 0,3,3,3 | Anode 04 Width eV |
| Sensors (0-1) | 1,3,9 | 0,3,3 | Anode 05 Energy eV |
| Sensors (0-1) | 1,3,4,25 | 0,3,3,3 | Anode 05 Width eV |
| Sensors (0-1) | 1,3,10 | 0,3,3 | Anode 06 Energy eV |
| Sensors (0-1) | 1,3,4,26 | 0,3,3,3 | Anode 06 Width eV |
| Sensors (0-1) | 1,3,11 | 0,3,3 | Anode 07 Energy eV |
| Sensors (0-1) | 1,3,4,27 | 0,3,3,3 | Anode 07 Width eV |
| Sensors (0-1) | 1,3,12 | 0,3,3 | Anode 08 Energy eV |
| Sensors (0-1) | 1,3,4,28 | 0,3,3,3 | Anode 08 Width eV |
| Sensors (0-1) | 1,3,13 | 0,3,3 | Anode 09 Energy eV |
| Sensors (0-1) | 1,3,4,29 | 0,3,3,3 | Anode 09 Width eV |
| Sensors (0-1) | 1,3,14 | 0,3,3 | Anode 10 Energy eV |
| Sensors (0-1) | 1,3,4,30 | 0,3,3,3 | Anode 10 Width eV |
| Sensors (0-1) | 1,3,15 | 0,3,3 | Anode 11 Energy eV |
| Sensors (0-1) | 1,3,4,31 | 0,3,3,3 | Anode 11 Width eV |
| Sensors (0-1) | 1,3,16 | 0,3,3 | Anode 12 Energy eV |
| Sensors (0-1) | 1,3,4,32 | 0,3,3,3 | Anode 12 Width eV |
| Sensors (0-1) | 1,3,17 | 0,3,3 | Anode 13 Energy eV |

```

* Sensors (0-1)    1,3,4,33      0,3,3,3      Anode 13 Width eV
* Sensors (0-1)    1,3,18       0,3,3        Anode 14 Energy eV
* Sensors (0-1)    1,3,4,34      0,3,3,3      Anode 14 Width eV
* Sensors (0-1)    1,3,19       0,3,3        Anode 15 Energy eV
* Sensors (0-1)    1,3,4,35      0,3,3,3      Anode 15 Width eV
*
* Data_len is evaluated according to the equation:
* [(128 steps * 2 sensors) + (1 cal. * 2 sensors)] * 2 bytes + 4
* (nanosecond timing) + 20 bytes for the rest of the elements in
* the data record.
*
*/
int s_year = 2003;          /* ds_year      */
int s_day = 1;             /* ds_day       */
int s_msec = 0;           /* ds_msec      */
int s_usec = 0;           /* ds_usec      */
int e_year = 2010;        /* de_year      */
int e_day = 1;            /* de_day       */
int e_msec = 0;           /* de_msec      */
int e_usec = 0;           /* de_usec      */
int smp_id = 1;           /* smp_id       */
int sen_mode = 2;         /* sen_mode     */
int n_qual = 5;           /* n_qual       */
int n_cal_sets = 1;       /* cal_sets     */
int n_tbls = 38;          /* num_tbls     */
int n_consts = 0;         /* num_consts   */
int n_status = 3;         /* status       */
int n_sensors = 2;        /* sen          */
int swp_len = 128;        /* swp_len      */
int max_nss = 1;         /* max_nss      */
int data_len = 540;       /* data_len     */
int fill_flag = 0;        /* fill_flg     */
int da_method = 0;        /* da_method    */
int nano_defined = 1;     /* nsec timetag */
struct Status0 {
    string name = "Software Version - Upper Byte"; /* name      */
    int state = 255; /* state     */
};
struct Status1 {
    string name = "Software Version - Lower Byte"; /* name      */
    int state = 255; /* state     */
};
struct Status2 {
    string name = "Software Mode"; /* name      */
    int state = 5; /* state     */
};
string qual_names = "Good Data"; /* name */
string qual_names = "Questionable Data"; /* name */
string qual_names = "Invalid Data"; /* name */
string qual_names = "Bad Data"; /* name */
string qual_names = "Unknown State"; /* name */
struct Sensor0 {
    string name = "ELS Deflection Voltage Reference"; /* name      */
    int d_type = 0; /* d_type    */
    int status = 1; /* status    */
}

```

```

        int tdw_len = 16;                /* tdw_len      */
        int time_offset = 0;            /* time_offset  */
};
struct Sensor1 {
    string name = "ELS Deflection Voltage Monitor"; /* name        */
    int d_type = 0;                       /* d_type      */
    int status = 1;                       /* status      */
    int tdw_len = 16;                    /* tdw_len     */
    int time_offset = 0;                 /* time_offset */
};
struct CalSet0 {
    string name = "ELS Temperature";      /* name        */
    int use = 0;                          /* use         */
    int word_len = 8;                    /* word length */
    int target = 0;                      /* target     */
};
struct Table0 {
    int tbl_sca_sz = 2;                  /* tbl_sca_sz */
    int tbl_ele_sz = 2;                  /* tbl_ele_sz */
    int tbl_type = 0;                   /* tbl_type   */
};
/*
 * Table 0
 * This table contains the sets of polynomial coefficients which
 * put the data value into the working math buffer.
 */
    int tbl_var = 0;                    /* tbl_var     */
    int tbl_expand = 0;                 /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {2, 2};           /* format     */
    int offset [2] = {0, 0};           /* offsets    */
    int scale [2] = {0, 0};            /* scale factor */
    int values [2] = {0, 1};           /* values     */
};
struct Table1 {
    int tbl_sca_sz = 4;                  /* tbl_sca_sz */
    int tbl_ele_sz = 4;                  /* tbl_ele_sz */
    int tbl_type = 0;                   /* tbl_type   */
};
/*
 * Table 1
 * This table contains the 1/index maximum value of the voltage DAC.
 * Dividing the control index by this number gives the fractional
 * amount of the voltage range used in deflecting the electron.
 */
    int tbl_var = 0;                    /* tbl_var     */
    int tbl_expand = 0;                 /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {2, 2};           /* format     */
    int offset [2] = {0, 2};           /* offsets    */
    int scale [4] = {-11, -12, 0, -12}; /* scale factor */
    int values [4] = {-352000000, 244200244, 0, 271333605}; /* values */
};
struct Table2 {
    int tbl_sca_sz = 2;                  /* tbl_sca_sz */
    int tbl_ele_sz = 2;                  /* tbl_ele_sz */
    int tbl_type = 0;                   /* tbl_type   */
};

```

```

/*
 * Table 2
 * This table contains the maximum control input voltage [volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 1}; /* offsets */
    int scale [2] = {0, -7}; /* scale factor */
    int values [2] = {5, 55555556}; /* values */
};
struct Table3 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 3
 * This table contains the maximum low range deflection voltage
 * [volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 1}; /* offsets */
    int scale [2] = {-2, -2}; /* scale factor */
    int values [2] = {2099, 2099}; /* values */
};
struct Table4 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 4
 * This table contains the k-factor for anode 00 [eV/volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-2}; /* scale factor */
    int values [1] = {728}; /* values */
};
struct Table5 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 5
 * This table contains the k-factor for anode 01 [eV/volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */

```

```

        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};         /* offsets     */
        int scale [1] = {-2};            /* scale factor */
        int values [1] = {722};          /* values      */
};
struct Table6 {
    int tbl_sca_sz = 1;                  /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                  /* tbl_ele_sz  */
    int tbl_type = 0;                    /* tbl_type    */
};
/*
* Table 6
* This table contains the k-factor for anode 02 [eV/volt].
*/
    int tbl_var = 1;                    /* tbl_var     */
    int tbl_expand = 0;                  /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {1, 1};          /* format      */
    int offset [2] = {0, 0};         /* offsets     */
    int scale [1] = {-2};            /* scale factor */
    int values [1] = {722};          /* values      */
};
struct Table7 {
    int tbl_sca_sz = 1;                  /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                  /* tbl_ele_sz  */
    int tbl_type = 0;                    /* tbl_type    */
};
/*
* Table 7
* This table contains the k-factor for anode 03 [eV/volt].
*/
    int tbl_var = 1;                    /* tbl_var     */
    int tbl_expand = 0;                  /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {1, 1};          /* format      */
    int offset [2] = {0, 0};         /* offsets     */
    int scale [1] = {-2};            /* scale factor */
    int values [1] = {722};          /* values      */
};
struct Table8 {
    int tbl_sca_sz = 1;                  /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                  /* tbl_ele_sz  */
    int tbl_type = 0;                    /* tbl_type    */
};
/*
* Table 8
* This table contains the k-factor for anode 04 [eV/volt].
*/
    int tbl_var = 1;                    /* tbl_var     */
    int tbl_expand = 0;                  /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {1, 1};          /* format      */
    int offset [2] = {0, 0};         /* offsets     */
    int scale [1] = {-2};            /* scale factor */
    int values [1] = {728};          /* values      */
};
struct Table9 {

```

```

        int tbl_sca_sz = 1;                /* tbl_sca_sz */
        int tbl_ele_sz = 1;                /* tbl_ele_sz */
        int tbl_type = 0;                  /* tbl_type */
/*
* Table 9
* This table contains the k-factor for anode 05 [eV/volt].
*/
        int tbl_var = 1;                  /* tbl_var */
        int tbl_expand = 0;                /* tbl_expand */
        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};           /* format */
        int offset [2] = {0, 0};           /* offsets */
        int scale [1] = {-2};              /* scale factor */
        int values [1] = {734};            /* values */
};
struct Table10 {
        int tbl_sca_sz = 1;                /* tbl_sca_sz */
        int tbl_ele_sz = 1;                /* tbl_ele_sz */
        int tbl_type = 0;                  /* tbl_type */
/*
* Table 10
* This table contains the k-factor for anode 06 [eV/volt].
*/
        int tbl_var = 1;                  /* tbl_var */
        int tbl_expand = 0;                /* tbl_expand */
        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};           /* format */
        int offset [2] = {0, 0};           /* offsets */
        int scale [1] = {-2};              /* scale factor */
        int values [1] = {734};            /* values */
};
struct Table11 {
        int tbl_sca_sz = 1;                /* tbl_sca_sz */
        int tbl_ele_sz = 1;                /* tbl_ele_sz */
        int tbl_type = 0;                  /* tbl_type */
/*
* Table 11
* This table contains the k-factor for anode 07 [eV/volt].
*/
        int tbl_var = 1;                  /* tbl_var */
        int tbl_expand = 0;                /* tbl_expand */
        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};           /* format */
        int offset [2] = {0, 0};           /* offsets */
        int scale [1] = {-2};              /* scale factor */
        int values [1] = {734};            /* values */
};
struct Table12 {
        int tbl_sca_sz = 1;                /* tbl_sca_sz */
        int tbl_ele_sz = 1;                /* tbl_ele_sz */
        int tbl_type = 0;                  /* tbl_type */
/*
* Table 12
* This table contains the k-factor for anode 08 [eV/volt].
*/

```

```

        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};      /* format       */
        int offset [2] = {0, 0};      /* offsets      */
        int scale [1] = {-2};         /* scale factor */
        int values [1] = {734};       /* values       */
};
struct Table13 {
    int tbl_sca_sz = 1;                /* tbl_sca_sz   */
    int tbl_ele_sz = 1;                /* tbl_ele_sz   */
    int tbl_type = 0;                  /* tbl_type     */
};
/*
 * Table 13
 * This table contains the k-factor for anode 09 [eV/volt].
 */
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};      /* format       */
        int offset [2] = {0, 0};      /* offsets      */
        int scale [1] = {-2};         /* scale factor */
        int values [1] = {734};       /* values       */
};
struct Table14 {
    int tbl_sca_sz = 1;                /* tbl_sca_sz   */
    int tbl_ele_sz = 1;                /* tbl_ele_sz   */
    int tbl_type = 0;                  /* tbl_type     */
};
/*
 * Table 14
 * This table contains the k-factor for anode 10 [eV/volt].
 */
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};      /* format       */
        int offset [2] = {0, 0};      /* offsets      */
        int scale [1] = {-2};         /* scale factor */
        int values [1] = {734};       /* values       */
};
struct Table15 {
    int tbl_sca_sz = 1;                /* tbl_sca_sz   */
    int tbl_ele_sz = 1;                /* tbl_ele_sz   */
    int tbl_type = 0;                  /* tbl_type     */
};
/*
 * Table 15
 * This table contains the k-factor for anode 11 [eV/volt].
 */
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};      /* format       */
        int offset [2] = {0, 0};      /* offsets      */
        int scale [1] = {-2};         /* scale factor */
        int values [1] = {734};       /* values       */

```

```

};
struct Table16 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 16
* This table contains the k-factor for anode 12 [eV/volt].
*/
    int tbl_var = 1;           /* tbl_var */
    int tbl_expand = 0;        /* tbl_expand */
    int crit_act_sz = 0;       /* crit_act_sz */
    int format [2] = {1, 1};   /* format */
    int offset [2] = {0, 0};   /* offsets */
    int scale [1] = {-2};      /* scale factor */
    int values [1] = {734};    /* values */
};
struct Table17 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 17
* This table contains the k-factor for anode 13 [eV/volt].
*/
    int tbl_var = 1;           /* tbl_var */
    int tbl_expand = 0;        /* tbl_expand */
    int crit_act_sz = 0;       /* crit_act_sz */
    int format [2] = {1, 1};   /* format */
    int offset [2] = {0, 0};   /* offsets */
    int scale [1] = {-2};      /* scale factor */
    int values [1] = {734};    /* values */
};
struct Table18 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 18
* This table contains the k-factor for anode 14 [eV/volt].
*/
    int tbl_var = 1;           /* tbl_var */
    int tbl_expand = 0;        /* tbl_expand */
    int crit_act_sz = 0;       /* crit_act_sz */
    int format [2] = {1, 1};   /* format */
    int offset [2] = {0, 0};   /* offsets */
    int scale [1] = {-2};      /* scale factor */
    int values [1] = {734};    /* values */
};
struct Table19 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 19

```

```

* This table contains the k-factor for anode 15 [eV/volt].
*/
    int tbl_var = 1;          /* tbl_var      */
    int tbl_expand = 0;      /* tbl_expand   */
    int crit_act_sz = 0;     /* crit_act_sz  */
    int format [2] = {1, 1}; /* format       */
    int offset [2] = {0, 0}; /* offsets      */
    int scale [1] = {-2};    /* scale factor */
    int values [1] = {728};  /* values       */
};
struct Table20 {
    int tbl_sca_sz = 1;      /* tbl_sca_sz   */
    int tbl_ele_sz = 1;      /* tbl_ele_sz   */
    int tbl_type = 0;        /* tbl_type     */
};
/*
* Table 20
* This table contains the detector resolution as a fractional
* number for anode 00.
*/
    int tbl_var = 1;          /* tbl_var      */
    int tbl_expand = 0;      /* tbl_expand   */
    int crit_act_sz = 0;     /* crit_act_sz  */
    int format [2] = {1, 1}; /* format       */
    int offset [2] = {0, 0}; /* offsets      */
    int scale [1] = {-4};    /* scale factor */
    int values [1] = {866};  /* values       */
};
struct Table21 {
    int tbl_sca_sz = 1;      /* tbl_sca_sz   */
    int tbl_ele_sz = 1;      /* tbl_ele_sz   */
    int tbl_type = 0;        /* tbl_type     */
};
/*
* Table 21
* This table contains the detector resolution as a fractional
* number for anode 01.
*/
    int tbl_var = 1;          /* tbl_var      */
    int tbl_expand = 0;      /* tbl_expand   */
    int crit_act_sz = 0;     /* crit_act_sz  */
    int format [2] = {1, 1}; /* format       */
    int offset [2] = {0, 0}; /* offsets      */
    int scale [1] = {-4};    /* scale factor */
    int values [1] = {834};  /* values       */
};
struct Table22 {
    int tbl_sca_sz = 1;      /* tbl_sca_sz   */
    int tbl_ele_sz = 1;      /* tbl_ele_sz   */
    int tbl_type = 0;        /* tbl_type     */
};
/*
* Table 22
* This table contains the detector resolution as a fractional
* number for anode 02.
*/
    int tbl_var = 1;          /* tbl_var      */
    int tbl_expand = 0;      /* tbl_expand   */

```

```

        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};          /* offsets     */
        int scale [1] = {-4};             /* scale factor*/
        int values [1] = {830};           /* values      */
};
struct Table23 {
    int tbl_sca_sz = 1;                   /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                   /* tbl_ele_sz  */
    int tbl_type = 0;                     /* tbl_type    */
};
/*
* Table 23
* This table contains the detector resolution as a fractional
* number for anode 03.
*/
        int tbl_var = 1;                   /* tbl_var     */
        int tbl_expand = 0;                /* tbl_expand  */
        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};          /* offsets     */
        int scale [1] = {-4};             /* scale factor*/
        int values [1] = {849};           /* values      */
};
struct Table24 {
    int tbl_sca_sz = 1;                   /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                   /* tbl_ele_sz  */
    int tbl_type = 0;                     /* tbl_type    */
};
/*
* Table 24
* This table contains the detector resolution as a fractional
* number for anode 04.
*/
        int tbl_var = 1;                   /* tbl_var     */
        int tbl_expand = 0;                /* tbl_expand  */
        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};          /* offsets     */
        int scale [1] = {-4};             /* scale factor*/
        int values [1] = {810};           /* values      */
};
struct Table25 {
    int tbl_sca_sz = 1;                   /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                   /* tbl_ele_sz  */
    int tbl_type = 0;                     /* tbl_type    */
};
/*
* Table 25
* This table contains the detector resolution as a fractional
* number for anode 05.
*/
        int tbl_var = 1;                   /* tbl_var     */
        int tbl_expand = 0;                /* tbl_expand  */
        int crit_act_sz = 0;                /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};          /* offsets     */
        int scale [1] = {-4};             /* scale factor*/

```

```

        int values [1] = {854};                /* values      */
    };
    struct Table26 {
        int tbl_sca_sz = 1;                    /* tbl_sca_sz   */
        int tbl_ele_sz = 1;                    /* tbl_ele_sz   */
        int tbl_type = 0;                       /* tbl_type     */
/*
* Table 26
* This table contains the detector resolution as a fractional
* number for anode 06.
*/
        int tbl_var = 1;                       /* tbl_var      */
        int tbl_expand = 0;                     /* tbl_expand   */
        int crit_act_sz = 0;                    /* crit_act_sz  */
        int format [2] = {1, 1};                /* format       */
        int offset [2] = {0, 0};                /* offsets      */
        int scale [1] = {-4};                    /* scale factor */
        int values [1] = {828};                 /* values       */
    };
    struct Table27 {
        int tbl_sca_sz = 1;                    /* tbl_sca_sz   */
        int tbl_ele_sz = 1;                    /* tbl_ele_sz   */
        int tbl_type = 0;                       /* tbl_type     */
/*
* Table 27
* This table contains the detector resolution as a fractional
* number for anode 07.
*/
        int tbl_var = 1;                       /* tbl_var      */
        int tbl_expand = 0;                     /* tbl_expand   */
        int crit_act_sz = 0;                    /* crit_act_sz  */
        int format [2] = {1, 1};                /* format       */
        int offset [2] = {0, 0};                /* offsets      */
        int scale [1] = {-4};                    /* scale factor */
        int values [1] = {767};                 /* values       */
    };
    struct Table28 {
        int tbl_sca_sz = 1;                    /* tbl_sca_sz   */
        int tbl_ele_sz = 1;                    /* tbl_ele_sz   */
        int tbl_type = 0;                       /* tbl_type     */
/*
* Table 28
* This table contains the detector resolution as a fractional
* number for anode 08.
*/
        int tbl_var = 1;                       /* tbl_var      */
        int tbl_expand = 0;                     /* tbl_expand   */
        int crit_act_sz = 0;                    /* crit_act_sz  */
        int format [2] = {1, 1};                /* format       */
        int offset [2] = {0, 0};                /* offsets      */
        int scale [1] = {-4};                    /* scale factor */
        int values [1] = {776};                 /* values       */
    };
    struct Table29 {
        int tbl_sca_sz = 1;                    /* tbl_sca_sz   */

```

```

        int tbl_ele_sz = 1;                /* tbl_ele_sz */
        int tbl_type = 0;                 /* tbl_type   */
/*
* Table 29
* This table contains the detector resolution as a fractional
* number for anode 09.
*/
        int tbl_var = 1;                  /* tbl_var     */
        int tbl_expand = 0;               /* tbl_expand  */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};         /* format      */
        int offset [2] = {0, 0};         /* offsets     */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {799};          /* values      */
};
struct Table30 {
        int tbl_sca_sz = 1;               /* tbl_sca_sz  */
        int tbl_ele_sz = 1;               /* tbl_ele_sz  */
        int tbl_type = 0;                 /* tbl_type    */
/*
* Table 30
* This table contains the detector resolution as a fractional
* number for anode 10.
*/
        int tbl_var = 1;                  /* tbl_var     */
        int tbl_expand = 0;               /* tbl_expand  */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};         /* format      */
        int offset [2] = {0, 0};         /* offsets     */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {797};          /* values      */
};
struct Table31 {
        int tbl_sca_sz = 1;               /* tbl_sca_sz  */
        int tbl_ele_sz = 1;               /* tbl_ele_sz  */
        int tbl_type = 0;                 /* tbl_type    */
/*
* Table 31
* This table contains the detector resolution as a fractional
* number for anode 11.
*/
        int tbl_var = 1;                  /* tbl_var     */
        int tbl_expand = 0;               /* tbl_expand  */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};         /* format      */
        int offset [2] = {0, 0};         /* offsets     */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {777};          /* values      */
};
struct Table32 {
        int tbl_sca_sz = 1;               /* tbl_sca_sz  */
        int tbl_ele_sz = 1;               /* tbl_ele_sz  */
        int tbl_type = 0;                 /* tbl_type    */
/*
* Table 32

```

```

* This table contains the detector resolution as a fractional
* number for anode 12.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-4}; /* scale factor */
    int values [1] = {800}; /* values */
};
struct Table33 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 33
* This table contains the detector resolution as a fractional
* number for anode 13.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-4}; /* scale factor */
    int values [1] = {837}; /* values */
};
struct Table34 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 34
* This table contains the detector resolution as a fractional
* number for anode 14.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-4}; /* scale factor */
    int values [1] = {838}; /* values */
};
struct Table35 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 35
* This table contains the detector resolution as a fractional
* number for anode 15.
*/
    int tbl_var = 1; /* tbl_var */

```

```

        int tbl_expand = 0;                /* tbl_expand */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};         /* format */
        int offset [2] = {0, 0};         /* offsets */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {812};          /* values */
};
struct Table36 {
    int tbl_sca_sz = 1;                  /* tbl_sca_sz */
    int tbl_ele_sz = 1;                  /* tbl_ele_sz */
    int tbl_type = 0;                    /* tbl_type */
};
/*
 * Table 36
 * This table contains the maximum monitor voltage [volt].
 */
    int tbl_var = 1;                    /* tbl_var */
    int tbl_expand = 0;                  /* tbl_expand */
    int crit_act_sz = 0;                  /* crit_act_sz */
    int format [2] = {-1, 1};           /* format */
    int offset [2] = {-1, 0};           /* offsets */
    int scale [1] = {0};                 /* scale factor */
    int values [1] = {5};                /* values */
};
struct Table37 {
    int tbl_sca_sz = 0;                  /* tbl_sca_sz */
    int tbl_ele_sz = 5;                  /* tbl_ele_sz */
    int tbl_type = 1;                    /* tbl_type */
};
/*
 * Table 37
 * ASCII definitions of the status states
 */
    int tbl_var = 4;                    /* tbl_var */
    int tbl_expand = 0;                  /* tbl_expand */
    int crit_act_sz = 0;                  /* crit_act_ele */
    int format [3] = {-1, -1, 0};       /* format */
    int offset [3] = {-1, -1, 0};       /* offsets */
    string values [5] = {                /* values */
        "Undefined", "Booting", "Safe", /* 000 - 002 */
        "Prom", "Normal"                /* 003 - 004 */
    };
};
}

```

ELSSWPH VIDF File

```

vidf v3_ELSSWPH {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft    */
    string experiment = "ASPERA-3";     /* exp_desc     */
    string instrument = "ELS";          /* inst_desc    */
    string contact = "Dr. Rickard Lundin";
    string contact = "Swedish Institute of Space Physics";
    string contact = "Box 812 S-981 28.Kiruna";
    string contact = "Sweden";
    string contact = "rickard@irf.se";
/*
* This virtual contains high range sweep data for ELS. The data
* values stored are the power supply reference and monitor of that
* setting, as returned in the ELS Science packets. The reference
* value is the plate voltage that is being requested. The monitor
* value is the voltage the deflection plates achieved. The high
* range voltage values are stored in the correct location within
* the sweep. The sweep length is 128 values.
*
* Units for each of the unique anodes of ELS require specific
* sensitivity and resolution values. These values are used to
* translate the control voltage or monitor voltage into the energy
* unit since calibration to energy is dependent of the individual
* anode characteristics.
*
* The following is a list of tables which are in this vidf
* TABLE 0: Polynomial coefficients which places the data into
* math buffer.
* TABLE 1: Polynomial coefficients which converts data into
* fraction of voltage range.
* TABLE 2: Maximum control input voltage [volts].
* TABLE 3: Maximum deflection high range voltage [volts].
* TABLE 4: Anode 00 K-factor [eV/Volt].
* TABLE 5: Anode 01 K-factor [eV/Volt].
* TABLE 6: Anode 02 K-factor [eV/Volt].
* TABLE 7: Anode 03 K-factor [eV/Volt].
* TABLE 8: Anode 04 K-factor [eV/Volt].
* TABLE 9: Anode 05 K-factor [eV/Volt].
* TABLE 10: Anode 06 K-factor [eV/Volt].
* TABLE 11: Anode 07 K-factor [eV/Volt].
* TABLE 12: Anode 08 K-factor [eV/Volt].
* TABLE 13: Anode 09 K-factor [eV/Volt].
* TABLE 14: Anode 10 K-factor [eV/Volt].
* TABLE 15: Anode 11 K-factor [eV/Volt].
* TABLE 16: Anode 12 K-factor [eV/Volt].
* TABLE 17: Anode 13 K-factor [eV/Volt].
* TABLE 18: Anode 14 K-factor [eV/Volt].
* TABLE 19: Anode 15 K-factor [eV/Volt].
* TABLE 20: Anode 00 Detector Resolution [fractionan number]
* TABLE 21: Anode 01 Detector Resolution [fractionan number]

```

```

* TABLE 22: Anode 02 Detector Resolution [fractionan number]
* TABLE 23: Anode 03 Detector Resolution [fractionan number]
* TABLE 24: Anode 04 Detector Resolution [fractionan number]
* TABLE 25: Anode 05 Detector Resolution [fractionan number]
* TABLE 26: Anode 06 Detector Resolution [fractionan number]
* TABLE 27: Anode 07 Detector Resolution [fractionan number]
* TABLE 28: Anode 08 Detector Resolution [fractionan number]
* TABLE 29: Anode 09 Detector Resolution [fractionan number]
* TABLE 30: Anode 10 Detector Resolution [fractionan number]
* TABLE 31: Anode 11 Detector Resolution [fractionan number]
* TABLE 32: Anode 12 Detector Resolution [fractionan number]
* TABLE 33: Anode 13 Detector Resolution [fractionan number]
* TABLE 34: Anode 14 Detector Resolution [fractionan number]
* TABLE 35: Anode 15 Detector Resolution [fractionan number]
* TABLE 36: Polynomial coefficients which converts data
*             into monitor fraction of voltage range.
* TABLE 37: ASCII definitions of status states
*

```

```

* The following are units which can be derived from the tables.
* The format is to give the tables applied followed by the
* operations and unit definition
*

```

| DATA TYPE | TABLES | OPERS | UNIT |
|---------------|----------|---------|--------------------|
| Sensors (0-1) | 0 | 0 | deflection index |
| Sensors (0-1) | 1 | 0 | fraction-unitless |
| Sensors (0-1) | 1,2 | 0,3 | control voltage |
| Sensors (1) | 1,36 | 0,3 | monitor voltage |
| Sensors (0-1) | 1,3 | 0,3 | deflection volt |
| Sensors (0-1) | 1,3,4 | 0,3,3 | Anode 00 Energy eV |
| Sensors (0-1) | 1,3,4,20 | 0,3,3,3 | Anode 00 Width eV |
| Sensors (0-1) | 1,3,5 | 0,3,3 | Anode 01 Energy eV |
| Sensors (0-1) | 1,3,4,21 | 0,3,3,3 | Anode 01 Width eV |
| Sensors (0-1) | 1,3,6 | 0,3,3 | Anode 02 Energy eV |
| Sensors (0-1) | 1,3,4,22 | 0,3,3,3 | Anode 02 Width eV |
| Sensors (0-1) | 1,3,7 | 0,3,3 | Anode 03 Energy eV |
| Sensors (0-1) | 1,3,4,23 | 0,3,3,3 | Anode 03 Width eV |
| Sensors (0-1) | 1,3,8 | 0,3,3 | Anode 04 Energy eV |
| Sensors (0-1) | 1,3,4,24 | 0,3,3,3 | Anode 04 Width eV |
| Sensors (0-1) | 1,3,9 | 0,3,3 | Anode 05 Energy eV |
| Sensors (0-1) | 1,3,4,25 | 0,3,3,3 | Anode 05 Width eV |
| Sensors (0-1) | 1,3,10 | 0,3,3 | Anode 06 Energy eV |
| Sensors (0-1) | 1,3,4,26 | 0,3,3,3 | Anode 06 Width eV |
| Sensors (0-1) | 1,3,11 | 0,3,3 | Anode 07 Energy eV |
| Sensors (0-1) | 1,3,4,27 | 0,3,3,3 | Anode 07 Width eV |
| Sensors (0-1) | 1,3,12 | 0,3,3 | Anode 08 Energy eV |
| Sensors (0-1) | 1,3,4,28 | 0,3,3,3 | Anode 08 Width eV |
| Sensors (0-1) | 1,3,13 | 0,3,3 | Anode 09 Energy eV |
| Sensors (0-1) | 1,3,4,29 | 0,3,3,3 | Anode 09 Width eV |
| Sensors (0-1) | 1,3,14 | 0,3,3 | Anode 10 Energy eV |
| Sensors (0-1) | 1,3,4,30 | 0,3,3,3 | Anode 10 Width eV |
| Sensors (0-1) | 1,3,15 | 0,3,3 | Anode 11 Energy eV |
| Sensors (0-1) | 1,3,4,31 | 0,3,3,3 | Anode 11 Width eV |
| Sensors (0-1) | 1,3,16 | 0,3,3 | Anode 12 Energy eV |
| Sensors (0-1) | 1,3,4,32 | 0,3,3,3 | Anode 12 Width eV |
| Sensors (0-1) | 1,3,17 | 0,3,3 | Anode 13 Energy eV |

```

* Sensors (0-1)    1,3,4,33      0,3,3,3      Anode 13 Width eV
* Sensors (0-1)    1,3,18       0,3,3        Anode 14 Energy eV
* Sensors (0-1)    1,3,4,34     0,3,3,3     Anode 14 Width eV
* Sensors (0-1)    1,3,19       0,3,3        Anode 15 Energy eV
* Sensors (0-1)    1,3,4,35     0,3,3,3     Anode 15 Width eV
*
* Data_len is evaluated according to the equation:
* [(128 steps * 2 sensors) + (1 cal. * 2 sensors)] * 2 bytes + 4
* (nanosecond timing) + 20 bytes for the rest of the elements in
* the data record.
*
*/
int s_year = 2003;          /* ds_year      */
int s_day = 1;             /* ds_day       */
int s_msec = 0;           /* ds_msec      */
int s_usec = 0;           /* ds_usec      */
int e_year = 2010;        /* de_year      */
int e_day = 1;            /* de_day       */
int e_msec = 0;           /* de_msec      */
int e_usec = 0;           /* de_usec      */
int smp_id = 1;           /* smp_id       */
int sen_mode = 2;         /* sen_mode     */
int n_qual = 5;           /* n_qual       */
int n_cal_sets = 1;       /* cal_sets     */
int n_tbls = 38;          /* num_tbls     */
int n_consts = 0;         /* num_consts   */
int n_status = 3;         /* status       */
int n_sensors = 2;        /* sen          */
int swp_len = 128;        /* swp_len      */
int max_nss = 1;         /* max_nss      */
int data_len = 540;       /* data_len     */
int fill_flag = 0;        /* fill_flg     */
int da_method = 0;        /* da_method    */
int nano_defined = 1;     /* nsec timetag */
struct Status0 {
    string name = "Software Version - Upper Byte"; /* name      */
    int state = 255;      /* state     */
};
struct Status1 {
    string name = "Software Version - Lower Byte"; /* name      */
    int state = 255;      /* state     */
};
struct Status2 {
    string name = "Software Mode"; /* name      */
    int state = 5;          /* state     */
};
string qual_names = "Good Data"; /* name */
string qual_names = "Questionable Data"; /* name */
string qual_names = "Invalid Data"; /* name */
string qual_names = "Bad Data"; /* name */
string qual_names = "Unknown State"; /* name */
struct Sensor0 {
    string name = "ELS Deflection Voltage Reference"; /* name      */
    int d_type = 0; /* d_type     */
    int status = 1; /* status     */
};

```

```

        int tdw_len = 16;                /* tdw_len      */
        int time_offset = 0;            /* time_offset  */
};
struct Sensor1 {
    string name = "ELS Deflection Voltage Monitor"; /* name      */
    int d_type = 0;                       /* d_type     */
    int status = 1;                       /* status     */
    int tdw_len = 16;                     /* tdw_len    */
    int time_offset = 0;                  /* time_offset */
};
struct CalSet0 {
    string name = "ELS Temperature";      /* name      */
    int use = 0;                          /* use       */
    int word_len = 8;                     /* word length */
    int target = 0;                       /* target    */
};
struct Table0 {
    int tbl_sca_sz = 2;                   /* tbl_sca_sz */
    int tbl_ele_sz = 2;                   /* tbl_ele_sz */
    int tbl_type = 0;                     /* tbl_type   */
};
/*
 * Table 0
 * This table contains the sets of polynomial coefficients which
 * put the data value into the working math buffer.
 */
    int tbl_var = 0;                      /* tbl_var    */
    int tbl_expand = 0;                   /* tbl_expand */
    int crit_act_sz = 0;                  /* crit_act_sz */
    int format [2] = {2, 2};              /* format     */
    int offset [2] = {0, 0};              /* offsets    */
    int scale [2] = {0, 0};                /* scale factor */
    int values [2] = {0, 1};              /* values     */
};
struct Table1 {
    int tbl_sca_sz = 4;                   /* tbl_sca_sz */
    int tbl_ele_sz = 4;                   /* tbl_ele_sz */
    int tbl_type = 0;                     /* tbl_type   */
};
/*
 * Table 1
 * This table contains the 1/index maximum value of the voltage DAC.
 * Dividing the control index by this number gives the fractional
 * amount of the voltage range used in deflecting the electron.
 */
    int tbl_var = 0;                      /* tbl_var    */
    int tbl_expand = 0;                   /* tbl_expand */
    int crit_act_sz = 0;                  /* crit_act_sz */
    int format [2] = {2, 2};              /* format     */
    int offset [2] = {0, 2};              /* offsets    */
    int scale [4] = {-11, -12, 0, -12};   /* scale factor */
    int values [4] = {-352000000, 244200244, 0, 271333605}; /* values */
/* this int scale [4] = {0, -12, 0, -12}; */ /* scale factor */
/* this int values [4] = {0, 244200244, 0, 271333605}; */ /* values */
    /* int scale [4] = {0, -12, 0, -10}; */ /* scale factor */
    /* int values [4] = {0, 244200244, 0, 39215686}; */ /* values */
};

```

```

    struct Table2 {
        int tbl_sca_sz = 2;           /* tbl_sca_sz */
        int tbl_ele_sz = 2;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type   */
/*
* Table 2
* This table contains the maximum control input voltage [volt].
*/
        int tbl_var = 1;           /* tbl_var     */
        int tbl_expand = 0;        /* tbl_expand  */
        int crit_act_sz = 0;       /* crit_act_sz */
        int format [2] = {1, 1};   /* format      */
        int offset [2] = {0, 1};   /* offsets     */
        int scale [2] = {0, -1};    /* scale factor */
        int values [2] = {5, 45};   /* values      */
/*
        int values [2] = {5, 55555556}; /* values      */
    };
    struct Table3 {
        int tbl_sca_sz = 2;           /* tbl_sca_sz */
        int tbl_ele_sz = 2;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type   */
/*
* Table 3
* This table contains the maximum high range deflection voltage
* [volt].
*/
        int tbl_var = 1;           /* tbl_var     */
        int tbl_expand = 0;        /* tbl_expand  */
        int crit_act_sz = 0;       /* crit_act_sz */
        int format [2] = {1, 1};   /* format      */
        int offset [2] = {0, 1};   /* offsets     */
        int scale [2] = {0, 0};    /* scale factor */
        int values [2] = {2800, 2800}; /* values      */
/*
        int values [2] = {2800, 31111111}; /* values      */
    };
    struct Table4 {
        int tbl_sca_sz = 1;         /* tbl_sca_sz */
        int tbl_ele_sz = 1;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type   */
/*
* Table 4
* This table contains the k-factor for anode 00 [eV/volt].
*/
        int tbl_var = 1;           /* tbl_var     */
        int tbl_expand = 0;        /* tbl_expand  */
        int crit_act_sz = 0;       /* crit_act_sz */
        int format [2] = {1, 1};   /* format      */
        int offset [2] = {0, 0};   /* offsets     */
        int scale [1] = {-2};      /* scale factor */
        int values [1] = {728};    /* values      */
    };
    struct Table5 {
        int tbl_sca_sz = 1;         /* tbl_sca_sz */
        int tbl_ele_sz = 1;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type   */

```

```

/*
 * Table 5
 * This table contains the k-factor for anode 01 [eV/volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-2}; /* scale factor */
    int values [1] = {722}; /* values */
};
struct Table6 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
/*
 * Table 6
 * This table contains the k-factor for anode 02 [eV/volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-2}; /* scale factor */
    int values [1] = {722}; /* values */
};
struct Table7 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
/*
 * Table 7
 * This table contains the k-factor for anode 03 [eV/volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-2}; /* scale factor */
    int values [1] = {722}; /* values */
};
struct Table8 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
/*
 * Table 8
 * This table contains the k-factor for anode 04 [eV/volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */

```

```

        int format [2] = {1, 1};           /* format      */
        int offset [2] = {0, 0};         /* offsets     */
        int scale [1] = {-2};           /* scale factor */
        int values [1] = {728};         /* values      */
};
struct Table9 {
    int tbl_sca_sz = 1;                 /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                 /* tbl_ele_sz  */
    int tbl_type = 0;                   /* tbl_type    */
};
/*
* Table 9
* This table contains the k-factor for anode 05 [eV/volt].
*/
    int tbl_var = 1;                   /* tbl_var     */
    int tbl_expand = 0;                 /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {1, 1};           /* format      */
    int offset [2] = {0, 0};         /* offsets     */
    int scale [1] = {-2};           /* scale factor */
    int values [1] = {734};         /* values      */
};
struct Table10 {
    int tbl_sca_sz = 1;                 /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                 /* tbl_ele_sz  */
    int tbl_type = 0;                   /* tbl_type    */
};
/*
* Table 10
* This table contains the k-factor for anode 06 [eV/volt].
*/
    int tbl_var = 1;                   /* tbl_var     */
    int tbl_expand = 0;                 /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {1, 1};           /* format      */
    int offset [2] = {0, 0};         /* offsets     */
    int scale [1] = {-2};           /* scale factor */
    int values [1] = {734};         /* values      */
};
struct Table11 {
    int tbl_sca_sz = 1;                 /* tbl_sca_sz  */
    int tbl_ele_sz = 1;                 /* tbl_ele_sz  */
    int tbl_type = 0;                   /* tbl_type    */
};
/*
* Table 11
* This table contains the k-factor for anode 07 [eV/volt].
*/
    int tbl_var = 1;                   /* tbl_var     */
    int tbl_expand = 0;                 /* tbl_expand  */
    int crit_act_sz = 0;                /* crit_act_sz */
    int format [2] = {1, 1};           /* format      */
    int offset [2] = {0, 0};         /* offsets     */
    int scale [1] = {-2};           /* scale factor */
    int values [1] = {734};         /* values      */
};
struct Table12 {
    int tbl_sca_sz = 1;                 /* tbl_sca_sz  */

```

```

        int tbl_ele_sz = 1;                /* tbl_ele_sz */
        int tbl_type = 0;                 /* tbl_type   */
/*
* Table 12
* This table contains the k-factor for anode 08 [eV/volt].
*/
        int tbl_var = 1;                  /* tbl_var     */
        int tbl_expand = 0;               /* tbl_expand  */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};          /* offsets     */
        int scale [1] = {-2};             /* scale factor */
        int values [1] = {734};           /* values      */
};
struct Table13 {
        int tbl_sca_sz = 1;               /* tbl_sca_sz  */
        int tbl_ele_sz = 1;               /* tbl_ele_sz  */
        int tbl_type = 0;                 /* tbl_type    */
/*
* Table 13
* This table contains the k-factor for anode 09 [eV/volt].
*/
        int tbl_var = 1;                  /* tbl_var     */
        int tbl_expand = 0;               /* tbl_expand  */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};          /* offsets     */
        int scale [1] = {-2};             /* scale factor */
        int values [1] = {734};           /* values      */
};
struct Table14 {
        int tbl_sca_sz = 1;               /* tbl_sca_sz  */
        int tbl_ele_sz = 1;               /* tbl_ele_sz  */
        int tbl_type = 0;                 /* tbl_type    */
/*
* Table 14
* This table contains the k-factor for anode 10 [eV/volt].
*/
        int tbl_var = 1;                  /* tbl_var     */
        int tbl_expand = 0;               /* tbl_expand  */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};          /* format      */
        int offset [2] = {0, 0};          /* offsets     */
        int scale [1] = {-2};             /* scale factor */
        int values [1] = {734};           /* values      */
};
struct Table15 {
        int tbl_sca_sz = 1;               /* tbl_sca_sz  */
        int tbl_ele_sz = 1;               /* tbl_ele_sz  */
        int tbl_type = 0;                 /* tbl_type    */
/*
* Table 15
* This table contains the k-factor for anode 11 [eV/volt].
*/
        int tbl_var = 1;                  /* tbl_var     */

```

```

        int tbl_expand = 0;                /* tbl_expand */
        int crit_act_sz = 0;              /* crit_act_sz */
        int format [2] = {1, 1};         /* format */
        int offset [2] = {0, 0};         /* offsets */
        int scale [1] = {-2};            /* scale factor */
        int values [1] = {734};          /* values */
};
struct Table16 {
    int tbl_sca_sz = 1;                  /* tbl_sca_sz */
    int tbl_ele_sz = 1;                  /* tbl_ele_sz */
    int tbl_type = 0;                    /* tbl_type */
};
/*
* Table 16
* This table contains the k-factor for anode 12 [eV/volt].
*/
        int tbl_var = 1;                 /* tbl_var */
        int tbl_expand = 0;              /* tbl_expand */
        int crit_act_sz = 0;            /* crit_act_sz */
        int format [2] = {1, 1};         /* format */
        int offset [2] = {0, 0};         /* offsets */
        int scale [1] = {-2};            /* scale factor */
        int values [1] = {734};          /* values */
};
struct Table17 {
    int tbl_sca_sz = 1;                  /* tbl_sca_sz */
    int tbl_ele_sz = 1;                  /* tbl_ele_sz */
    int tbl_type = 0;                    /* tbl_type */
};
/*
* Table 17
* This table contains the k-factor for anode 13 [eV/volt].
*/
        int tbl_var = 1;                 /* tbl_var */
        int tbl_expand = 0;              /* tbl_expand */
        int crit_act_sz = 0;            /* crit_act_sz */
        int format [2] = {1, 1};         /* format */
        int offset [2] = {0, 0};         /* offsets */
        int scale [1] = {-2};            /* scale factor */
        int values [1] = {734};          /* values */
};
struct Table18 {
    int tbl_sca_sz = 1;                  /* tbl_sca_sz */
    int tbl_ele_sz = 1;                  /* tbl_ele_sz */
    int tbl_type = 0;                    /* tbl_type */
};
/*
* Table 18
* This table contains the k-factor for anode 14 [eV/volt].
*/
        int tbl_var = 1;                 /* tbl_var */
        int tbl_expand = 0;              /* tbl_expand */
        int crit_act_sz = 0;            /* crit_act_sz */
        int format [2] = {1, 1};         /* format */
        int offset [2] = {0, 0};         /* offsets */
        int scale [1] = {-2};            /* scale factor */
        int values [1] = {734};          /* values */
};

```

```

    struct Table19 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type */
/*
* Table 19
* This table contains the k-factor for anode 15 [eV/volt].
*/
        int tbl_var = 1;           /* tbl_var */
        int tbl_expand = 0;        /* tbl_expand */
        int crit_act_sz = 0;       /* crit_act_sz */
        int format [2] = {1, 1};   /* format */
        int offset [2] = {0, 0};   /* offsets */
        int scale [1] = {-2};      /* scale factor */
        int values [1] = {728};    /* values */
    };
    struct Table20 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type */
/*
* Table 20
* This table contains the detector resolution as a fractional
* number for anode 00.
*/
        int tbl_var = 1;           /* tbl_var */
        int tbl_expand = 0;        /* tbl_expand */
        int crit_act_sz = 0;       /* crit_act_sz */
        int format [2] = {1, 1};   /* format */
        int offset [2] = {0, 0};   /* offsets */
        int scale [1] = {-4};      /* scale factor */
        int values [1] = {866};    /* values */
    };
    struct Table21 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type */
/*
* Table 21
* This table contains the detector resolution as a fractional
* number for anode 01.
*/
        int tbl_var = 1;           /* tbl_var */
        int tbl_expand = 0;        /* tbl_expand */
        int crit_act_sz = 0;       /* crit_act_sz */
        int format [2] = {1, 1};   /* format */
        int offset [2] = {0, 0};   /* offsets */
        int scale [1] = {-4};      /* scale factor */
        int values [1] = {834};    /* values */
    };
    struct Table22 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;         /* tbl_ele_sz */
        int tbl_type = 0;           /* tbl_type */
/*

```

```

* Table 22
* This table contains the detector resolution as a fractional
* number for anode 02.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-4}; /* scale factor */
    int values [1] = {830}; /* values */
};
struct Table23 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 23
* This table contains the detector resolution as a fractional
* number for anode 03.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-4}; /* scale factor */
    int values [1] = {849}; /* values */
};
struct Table24 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 24
* This table contains the detector resolution as a fractional
* number for anode 04.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-4}; /* scale factor */
    int values [1] = {810}; /* values */
};
struct Table25 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 25
* This table contains the detector resolution as a fractional
* number for anode 05.
*/

```

```

        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};     /* format       */
        int offset [2] = {0, 0};     /* offsets      */
        int scale [1] = {-4};        /* scale factor */
        int values [1] = {854};      /* values       */
};
struct Table26 {
    int tbl_sca_sz = 1;              /* tbl_sca_sz   */
    int tbl_ele_sz = 1;              /* tbl_ele_sz   */
    int tbl_type = 0;                /* tbl_type     */
};
/*
 * Table 26
 * This table contains the detector resolution as a fractional
 * number for anode 06.
 */
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};     /* format       */
        int offset [2] = {0, 0};     /* offsets      */
        int scale [1] = {-4};        /* scale factor */
        int values [1] = {828};      /* values       */
};
struct Table27 {
    int tbl_sca_sz = 1;              /* tbl_sca_sz   */
    int tbl_ele_sz = 1;              /* tbl_ele_sz   */
    int tbl_type = 0;                /* tbl_type     */
};
/*
 * Table 27
 * This table contains the detector resolution as a fractional
 * number for anode 07.
 */
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};     /* format       */
        int offset [2] = {0, 0};     /* offsets      */
        int scale [1] = {-4};        /* scale factor */
        int values [1] = {767};      /* values       */
};
struct Table28 {
    int tbl_sca_sz = 1;              /* tbl_sca_sz   */
    int tbl_ele_sz = 1;              /* tbl_ele_sz   */
    int tbl_type = 0;                /* tbl_type     */
};
/*
 * Table 28
 * This table contains the detector resolution as a fractional
 * number for anode 08.
 */
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;            /* tbl_expand   */
        int crit_act_sz = 0;          /* crit_act_sz  */
        int format [2] = {1, 1};     /* format       */

```

```

        int offset [2] = {0, 0};           /* offsets      */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {776};         /* values       */
    };
    struct Table29 {
        int tbl_sca_sz = 1;              /* tbl_sca_sz   */
        int tbl_ele_sz = 1;              /* tbl_ele_sz   */
        int tbl_type = 0;                /* tbl_type     */
    };
/*
* Table 29
* This table contains the detector resolution as a fractional
* number for anode 09.
*/
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;              /* tbl_expand   */
        int crit_act_sz = 0;             /* crit_act_sz  */
        int format [2] = {1, 1};         /* format       */
        int offset [2] = {0, 0};         /* offsets      */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {799};         /* values       */
    };
    struct Table30 {
        int tbl_sca_sz = 1;              /* tbl_sca_sz   */
        int tbl_ele_sz = 1;              /* tbl_ele_sz   */
        int tbl_type = 0;                /* tbl_type     */
    };
/*
* Table 30
* This table contains the detector resolution as a fractional
* number for anode 10.
*/
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;              /* tbl_expand   */
        int crit_act_sz = 0;             /* crit_act_sz  */
        int format [2] = {1, 1};         /* format       */
        int offset [2] = {0, 0};         /* offsets      */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {797};         /* values       */
    };
    struct Table31 {
        int tbl_sca_sz = 1;              /* tbl_sca_sz   */
        int tbl_ele_sz = 1;              /* tbl_ele_sz   */
        int tbl_type = 0;                /* tbl_type     */
    };
/*
* Table 31
* This table contains the detector resolution as a fractional
* number for anode 11.
*/
        int tbl_var = 1;                /* tbl_var      */
        int tbl_expand = 0;              /* tbl_expand   */
        int crit_act_sz = 0;             /* crit_act_sz  */
        int format [2] = {1, 1};         /* format       */
        int offset [2] = {0, 0};         /* offsets      */
        int scale [1] = {-4};            /* scale factor */
        int values [1] = {777};         /* values       */
    };
};

```

```

    struct Table32 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;          /* tbl_ele_sz */
        int tbl_type = 0;            /* tbl_type   */
/*
* Table 32
* This table contains the detector resolution as a fractional
* number for anode 12.
*/
        int tbl_var = 1;             /* tbl_var     */
        int tbl_expand = 0;          /* tbl_expand  */
        int crit_act_sz = 0;         /* crit_act_sz */
        int format [2] = {1, 1};    /* format     */
        int offset [2] = {0, 0};    /* offsets    */
        int scale [1] = {-4};        /* scale factor */
        int values [1] = {800};     /* values     */
    };
    struct Table33 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;          /* tbl_ele_sz */
        int tbl_type = 0;            /* tbl_type   */
/*
* Table 33
* This table contains the detector resolution as a fractional
* number for anode 13.
*/
        int tbl_var = 1;             /* tbl_var     */
        int tbl_expand = 0;          /* tbl_expand  */
        int crit_act_sz = 0;         /* crit_act_sz */
        int format [2] = {1, 1};    /* format     */
        int offset [2] = {0, 0};    /* offsets    */
        int scale [1] = {-4};        /* scale factor */
        int values [1] = {837};     /* values     */
    };
    struct Table34 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;          /* tbl_ele_sz */
        int tbl_type = 0;            /* tbl_type   */
/*
* Table 34
* This table contains the detector resolution as a fractional
* number for anode 14.
*/
        int tbl_var = 1;             /* tbl_var     */
        int tbl_expand = 0;          /* tbl_expand  */
        int crit_act_sz = 0;         /* crit_act_sz */
        int format [2] = {1, 1};    /* format     */
        int offset [2] = {0, 0};    /* offsets    */
        int scale [1] = {-4};        /* scale factor */
        int values [1] = {838};     /* values     */
    };
    struct Table35 {
        int tbl_sca_sz = 1;           /* tbl_sca_sz */
        int tbl_ele_sz = 1;          /* tbl_ele_sz */
        int tbl_type = 0;            /* tbl_type   */

```

```

/*
 * Table 35
 * This table contains the detector resolution as a fractional
 * number for anode 15.
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {1, 1}; /* format */
    int offset [2] = {0, 0}; /* offsets */
    int scale [1] = {-4}; /* scale factor */
    int values [1] = {812}; /* values */
};
struct Table36 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 36
 * This table contains the maximum monitor voltage [volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [2] = {-1, 1}; /* format */
    int offset [2] = {-1, 0}; /* offsets */
    int scale [1] = {0}; /* scale factor */
    int values [1] = {5}; /* values */
};
struct Table37 {
    int tbl_sca_sz = 0; /* tbl_sca_sz */
    int tbl_ele_sz = 5; /* tbl_ele_sz */
    int tbl_type = 1; /* tbl_type */
};
/*
 * Table 37
 * ASCII definitions of the status states
 */
    int tbl_var = 4; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_ele = 0; /* crit_act_ele */
    int format [3] = {-1, -1, 0}; /* format */
    int offset [3] = {-1, -1, 0}; /* offsets */
    string values [5] = { /* values */
        "Undefined", "Booting", "Safe", /* 000 - 002 */
        "Prom", "Normal" /* 003 - 004 */
    };
};
}

```

ELSSCIL VIDF File

```

vidf v3_ELSSCIL {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft   */
    string experiment = "ASPERA-3";    /* exp_desc    */
    string instrument = "ELS";         /* inst_desc   */
    string contact = "Dr. Rickard Lundin";
    string contact = "Swedish Institute of Space Physics";
    string contact = "Box 812 S-981 28.Kiruna";
    string contact = "Sweden";
    string contact = "rickard@irf.se";
/*
* This VIDF contains the science data from ELS. Data is stored
* for the low range of the ELS power supply as a function of the
* DAC representation. The DAC range is 12 bits representing a +5V
* control voltage. This control voltage sets the ELS deflection
* supply which has a range of 0V to 20.99V for the low range.
* Stored data value is the number of counts received in the
* accumulation interval of the ELS packets.
*
* The following is a list of tables which are in this vidf
* TABLE 0: Polynomial coefficients which places the data into
* math buffer.
* TABLE 1: Polynomial coefficients which converts sensor index
* into fraction of voltage range.
* TABLE 2: Maximum control input voltage [volts].
* TABLE 3: Maximum deflection low range voltage [volts].
* TABLE 4: Anode K-factors [eV/Volt].
* TABLE 5: Detector Efficiencies.
* TABLE 6: Efficiency Adjustment Factor For Each Anode [fractional
* number]
* TABLE 7: Geometric Factors [cm**2-sr]
* TABLE 8: MCP Transparency [fraction]
* TABLE 9: Grid Transparency [fraction]
* TABLE 10: Active Anode Area Ratio [fraction]
* TABLE 11: Amplitude Adjustment or Scaling Factor
* TABLE 12: Detector Resolution [fractional number]
* TABLE 13: Energy Conversion [erg/eV]
* TABLE 14: Conversion from Energy Intensity to Distribution Funct
* TABLE 15: Number of spectra summed together
* TABLE 16: Number of steps summed together
* TABLE 17: Polynomial coefficients which places the value on 1 into the
* math buffer.
* TABLE 18: Shift Calibration Set 1 to Upper bytes of 4-byte integer -
* High Range Sweep Summation (MSB)
* TABLE 19: Shift Calibration Set 2 to Lower bytes of 4-byte integer -
* High Range Sweep Summation (LSB)
* TABLE 20: Shift Calibration Set 3 to Upper bytes of 4-byte integer -
* Total Range Sweep Summation (MSB)
* TABLE 21: Shift Calibration Set 4 to Lower bytes of 4-byte integer -
* Total Range Sweep Summation (LSB)

```

- * TABLE 22: Shift Calibration Set 5 to Upper bytes of 4-byte integer -
- * Total Range Anode Summation (MSB)
- * TABLE 23: Shift Calibration Set 6 to Lower bytes of 4-byte integer -
- * Total Range Anode Summation (LSB)
- * TABLE 24: Convert Temperature Calibration (cal set 0) Value to Voltage
- * TABLE 25: Convert Temperature Calibration (cal set 0) Value to milliamp
- * TABLE 26: Convert Temperature Calibration (cal set 0) Value to degrees C
- * TABLE 27: Convert from eV to Joule [Joule/eV]
- * TABLE 28: Constant number 2 for conversion used in energy to velocity
- * TABLE 29: Contains the electron mass in [kg]
- * TABLE 30: Contains the constant number 1000 for converting from m to km
- * TABLE 31: Contains the constant number 1000 for converting m/s into km/s
- * TABLE 32: ASCII strings which define what state the bit values
- * represent for the status flags defined
- * TABLE 33: Table conversion factors to determine number of summations

* The following are units which can be derived from the tables.
 * The format is to give the tables applied followed by the
 * operations and unit definition

| * DATA TYPE | * TABLES | * OPERS | * UNIT |
|-------------|--------------|--------------------|------------------------|
| * Scan | 1 | 0 | Dimensionless fraction |
| * Scan | 1,2 | 0,3 | control voltage |
| * Scan | 1,3 | 0,3 | deflection voltage |
| * Scan | 1,3,4 | 0,3,3 | eV |
| * Scan | 1,3,4,27 | 0,3,3,3 | Joule |
| * Scan | 1,3,4,27,28, | 0,3,3,3,3, | m/s |
| * Scan | 29 | 64 | |
| * Scan | 1,3,4,27,28, | 0,3,3,3,3, | km/s |
| * Scan | 29,31 | 64,3 | |
| * Sweep | 0 | 0 | cnts/accum |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/accum (eff. cor) |
| * Sweep | 0,5,-1 | 0,4,2003 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/sec |
| * Sweep | 0,5,-1 | 150,4.2003 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/(cm**2-str-s) |
| * Sweep | 0,5,-1,7,11 | 150,4,2003,4,3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/(cm**2-str-s-eV) |
| * Sweep | 0,5,-1,7,1, | 150,4,2003,4,1000, | |
| * Sweep | 3,4,-1, | 1003,1003,2004, | |
| * Sweep | 12,11 | 4,3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | ergs/(cm**2-str-s-eV) |
| * Sweep | 0,5,-1,7, | 150,4,2003,4, | |
| * Sweep | 12,13 | 4,3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | sec**3/km**6 |
| * Sweep | 0,5,-1,7,1, | 150,4,2003,4,1000, | |
| * Sweep | 3,4,-1, | 1003,1003,2004, | |
| * Sweep | 12,-1,14, | 4,2004,3, | |
| * Sweep | 11 | 3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | sec**3/m**6 |
| * Sweep | 0,5,-1,7,1, | 150,4,2003,4,1000, | |
| * Sweep | 3,4,-1, | 1003,1003,2004, | |
| * Sweep | 12,-1,14, | 4,2004,3, | |
| * Sweep | 11,30,30,30, | 3,4,4,4, | |
| * Sweep | 30,30,30 | 4,4,4 | |

```

*   Sweep      15           0           Number Spectra Summed
*   Sweep      16           0           Number Energies Summed
*   Cal        18,19        0,1        High Range Sweep Sum
*   Cal        20,21        0,1        Total Range Sweep Sum
*   Cal        22,23        0,1        Total Anode Sum or Sum on MCP
*   Cal        24           0           Temperature in monitor voltage
*   Cal        25           0           Temp. in monitor milliamp
*   Cal        26           0           Temperature in degrees C
*   Mode       31           0           ASCII Mode Strings
*   Mode       32           0           # Spectra in Time Sum
*   Mode       32           0           # Energy Levels in Energy Sum
*
*   Data_len is set to the maximum, assuming that all sensors are returned.
*   (128 steps + 4 cal.) * 2 bytes * 16 sensors + 5 cal. * 2 bytes +
*   4 (nanosecond timing) + 20 bytes for the rest of the elements in the
*   data record.
*
*/
int s_year = 2003;           /* ds_year      */
int s_day = 1;              /* ds_day       */
int s_msec = 0;            /* ds_msec      */
int s_usec = 0;           /* ds_usec     */
int e_year = 2010;         /* de_year     */
int e_day = 1;             /* de_day      */
int e_msec = 0;           /* de_msec     */
int e_usec = 0;           /* de_usec     */
int smp_id = 1;            /* smp_id      */
int sen_mode = 2;          /* sen_mode    */
int n_qual = 5;            /* n_qual     */
int n_cal_sets = 9;        /* cal_sets   */
int n_tbls = 34;           /* num_tbls   */
int n_consts = 8;          /* num_consts  */
int n_status = 23;         /* status     */
int n_sensors = 16;        /* sen        */
int swp_len = 4096;        /* swp_len    */
int max_nss = 1;           /* max_nss    */
int data_len = 4258;       /* data_len   */
int fill_flag = 1;         /* fill_flg   */
int fill = 65535;          /* fill value  */
int da_method = 0;         /* da_method  */
int nano_defined = 1;      /* nsec timetag */
struct PitchAngle {
    int format = 1;
    string project = "MARS";
    string mission = "Mars_Express";
    string experiment = "MODELS";
    string instrument = "MAG";
    string vinstrument = "SAF_BMod";
    int bx = 0;
    int by = 1;
    int bz = 2;
    int num_tbls = 0;
};
struct Status0 {
    string name = "Software Version - Upper Byte"; /* name      */

```

```
        int state = 255;                                /* state    */
};
struct Status1 {
    string name = "Software Version - Lower Byte";    /* name      */
    int state = 255;                                  /* state    */
};
struct Status2 {
    string name = "Software Mode";                    /* name      */
    int state = 5;                                    /* state    */
};
struct Status3 {
    string name = "Time Summation";                   /* name      */
    int state = 17;                                   /* state    */
};
struct Status4 {
    string name = "Step Summation";                   /* name      */
    int state = 3;                                    /* state    */
};
struct Status5 {
    string name = "Log Compression";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status6 {
    string name = "Sector 0 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status7 {
    string name = "Sector 1 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status8 {
    string name = "Sector 2 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status9 {
    string name = "Sector 3 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status10 {
    string name = "Sector 4 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status11 {
    string name = "Sector 5 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status12 {
    string name = "Sector 6 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status13 {
    string name = "Sector 7 Enable";                  /* name      */
    int state = 2;                                    /* state    */
};
struct Status14 {
```

```

        string name = "Sector 8 Enable";           /* name      */
        int state = 2;                             /* state     */
};
struct Status15 {
    string name = "Sector 9 Enable";             /* name      */
    int state = 2;                               /* state     */
};
struct Status16 {
    string name = "Sector 10 Enable";           /* name      */
    int state = 2;                               /* state     */
};
struct Status17 {
    string name = "Sector 11 Enable";           /* name      */
    int state = 2;                               /* state     */
};
struct Status18 {
    string name = "Sector 12 Enable";           /* name      */
    int state = 2;                               /* state     */
};
struct Status19 {
    string name = "Sector 13 Enable";           /* name      */
    int state = 2;                               /* state     */
};
struct Status20 {
    string name = "Sector 14 Enable";           /* name      */
    int state = 2;                               /* state     */
};
struct Status21 {
    string name = "Sector 15 Enable";           /* name      */
    int state = 2;                               /* state     */
};
struct Status22 {
    string name = "Number Of Active Anodes";    /* name      */
    int state = 16;                             /* state     */
};
string qual_names = "Good Data";               /* name */
string qual_names = "Questionable Data";       /* name */
string qual_names = "Invalid Data";            /* name */
string qual_names = "Bad Data";                /* name */
string qual_names = "Unknown State";           /* name */
struct Sensor0 {
    string name = "ELS Anode 0";                /* name      */
    int d_type = 0;                             /* d_type    */
    int status = 1;                             /* status    */
    int tdw_len = 16;                           /* tdw_len   */
    int time_offset = 0;                        /* time_offset */
};
struct Sensor1 {
    string name = "ELS Anode 1";                /* name      */
    int d_type = 0;                             /* d_type    */
    int status = 1;                             /* status    */
    int tdw_len = 16;                           /* tdw_len   */
    int time_offset = 0;                        /* time_offset */
};
struct Sensor2 {

```

```

    string name = "ELS Anode 2";           /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor3 {
    string name = "ELS Anode 3";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor4 {
    string name = "ELS Anode 4";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor5 {
    string name = "ELS Anode 5";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor6 {
    string name = "ELS Anode 6";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor7 {
    string name = "ELS Anode 7";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor8 {
    string name = "ELS Anode 8";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor9 {
    string name = "ELS Anode 9";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                     /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
};

```

```

struct Sensor10 {
    string name = "ELS Anode 10";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                    /* time_offset   */
};
struct Sensor11 {
    string name = "ELS Anode 11";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                    /* time_offset   */
};
struct Sensor12 {
    string name = "ELS Anode 12";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                    /* time_offset   */
};
struct Sensor13 {
    string name = "ELS Anode 13";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                    /* time_offset   */
};
struct Sensor14 {
    string name = "ELS Anode 14";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                    /* time_offset   */
};
struct Sensor15 {
    string name = "ELS Anode 15";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                    /* time_offset   */
};
struct CalSet0 {
    string name = "ELS Temperature";        /* name          */
    int use = 0;                            /* use           */
    int word_len = 8;                        /* word length   */
    int target = 0;                          /* target        */
    int cal_scope = 1;                       /* scope         */
};
struct CalSet1 {
    string name = "ELS MCP Bias Reference";  /* name          */
    int use = 0;                            /* use           */
    int word_len = 8;                        /* word length   */
    int target = 0;                          /* target        */
    int cal_scope = 1;                       /* scope         */
};

```

```

};
struct CalSet2 {
    string name = "ELS MCP Bias Monitor";           /* name      */
    int use = 0;                                    /* use       */
    int word_len = 8;                               /* word length */
    int target = 0;                                 /* target    */
    int cal_scope = 1;                              /* scope     */
};
struct CalSet3 {
    string name = "Total Range Anode Summation (MSB)"; /* name      */
    int use = 0;                                    /* use       */
    int word_len = 16;                              /* word length */
    int target = 0;                                 /* target    */
    int cal_scope = 1;                              /* scope     */
};
struct CalSet4 {
    string name = "Total Range Anode Summation (LSB)"; /* name      */
    int use = 0;                                    /* use       */
    int word_len = 16;                              /* word length */
    int target = 0;                                 /* target    */
    int cal_scope = 1;                              /* scope     */
};
struct CalSet5 {
    string name = "Low Range Sweep Summation (MSB)"; /* name      */
    int use = 0;                                    /* use       */
    int word_len = 16;                              /* word length */
    int target = 0;                                 /* target    */
    int cal_scope = 0;                              /* scope     */
};
struct CalSet6 {
    string name = "Low Range Sweep Summation (LSB)"; /* name      */
    int use = 0;                                    /* use       */
    int word_len = 16;                              /* word length */
    int target = 0;                                 /* target    */
    int cal_scope = 0;                              /* scope     */
};
struct CalSet7 {
    string name = "Total Range Sweep Summation (MSB)"; /* name      */
    int use = 0;                                    /* use       */
    int word_len = 16;                              /* word length */
    int target = 0;                                 /* target    */
    int cal_scope = 0;                              /* scope     */
};
struct CalSet8 {
    string name = "Total Range Sweep Summation (LSB)"; /* name      */
    int use = 0;                                    /* use       */
    int word_len = 16;                              /* word length */
    int target = 0;                                 /* target    */
    int cal_scope = 0;                              /* scope     */
};
struct Table0 {
    int tbl_sca_sz = 2;                             /* tbl_sca_sz */
    int tbl_ele_sz = 2;                             /* tbl_ele_sz */
    int tbl_type = 0;                               /* tbl_type   */
};
/*

```

```

* Table 0
* This table contains the sets of polynomial coefficients which
* put the data value into the working math buffer.
*/
    int tbl_var = 0; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, 0}; /* scale factor */
    int values [2] = {0, 1}; /* values */
};
struct Table1 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 1
* This table contains the 1/index maximum value of the voltage DAC.
* Dividing the control index by this number gives the fractional
* amount of the voltage range used in deflecting the electron.
*/
    int tbl_var = 2; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, -12}; /* scale factor */
    int values [2] = {0, 244200244}; /* values */
};
struct Table2 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 2
* This table contains the maximum control input voltage [volt].
*/
    int tbl_var = 1; /* tbl_var */

```

```

int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    1, 1, 1, 1, 1, 1, /* 000 - 005 */
    1, 1, 1, 1, 1, 1, /* 006 - 011 */
    1, 1, 1, 1, /* 012 - 015 */
};
int offset [16] = { /* offsets */
    0, 0, 0, 0, 0, 0, /* 000 - 005 */
    0, 0, 0, 0, 0, 0, /* 006 - 011 */
    0, 0, 0, 0, /* 012 - 015 */
};
int scale [1] = {0}; /* scale factor */
int values [1] = {5}; /* values */
};
struct Table3 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 3
* This table contains the maximum low range deflection voltage
* [volt].
*/
int tbl_var = 1; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    1, 1, 1, 1, 1, 1, /* 000 - 005 */
    1, 1, 1, 1, 1, 1, /* 006 - 011 */
    1, 1, 1, 1, /* 012 - 015 */
};
int offset [16] = { /* offsets */
    0, 0, 0, 0, 0, 0, /* 000 - 005 */
    0, 0, 0, 0, 0, 0, /* 006 - 011 */
    0, 0, 0, 0, /* 012 - 015 */
};
int scale [1] = {-2}; /* scale factor */
int values [1] = {2099}; /* values */
};
struct Table4 {
    int tbl_sca_sz = 16; /* tbl_sca_sz */
    int tbl_ele_sz = 16; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 4
* This table contains the k-factors for each anode [eV/volt].
*/
int tbl_var = 1; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    1, 1, 1, 1, 1, 1, /* 000 - 005 */
    1, 1, 1, 1, 1, 1, /* 006 - 011 */
    1, 1, 1, 1, /* 012 - 015 */
};

```

```

};
int offset [16] = { /* offsets */
    0, 1, 2, 3, 4, 5, /* 000 - 005 */
    6, 7, 8, 9, 10, 11, /* 006 - 011 */
    12, 13, 14, 15 /* 012 - 015 */
};
int scale [16] = { /* scale factor*/
    -3, -3, -3, -3, -3, -3, /* 000 - 005 */
    -3, -3, -3, -3, -3, -3, /* 006 - 011 */
    -3, -3, -3, -3 /* 012 - 015 */
};
int values [16] = { /* values */
    7167, 7152, 7141, 7165, 7188, 7625, /* 000 - 005 */
    7262, 7266, 7275, 7254, 7262, 7255, /* 006 - 011 */
    7255, 7271, 7253, 7188 /* 012 - 015 */
};
};
struct Table5 {
    int tbl_sca_sz = 16; /* tbl_sca_sz */
    int tbl_ele_sz = 16; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
 * Table 5
 * This table contains the detector efficiencies for the detector
 */
int tbl_var = 1; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    1, 1, 1, 1, 1, 1, /* 000 - 005 */
    1, 1, 1, 1, 1, 1, /* 006 - 011 */
    1, 1, 1, 1 /* 012 - 015 */
};
int offset [16] = { /* offsets */
    0, 1, 2, 3, 4, 5, /* 000 - 005 */
    6, 7, 8, 9, 10, 11, /* 006 - 011 */
    12, 13, 14, 15 /* 012 - 015 */
};
int scale [16] = { /* scale factor*/
    -2, -2, -2, -2, -2, -2, /* 000 - 005 */
    -2, -2, -2, -2, -2, -2, /* 006 - 011 */
    -2, -2, -2, -2 /* 012 - 015 */
};
int values [16] = { /* values */
    95, 95, 95, 95, 95, 95, /* 000 - 005 */
    95, 95, 95, 95, 95, 95, /* 006 - 011 */
    95, 95, 95, 95 /* 012 - 015 */
};
};
struct Table6 {
    int tbl_sca_sz = 163; /* tbl_sca_sz */
    int tbl_ele_sz = 163; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
 * Table 6

```

* This table contains the efficiency adjustment factor for the detector.
 * This is a description of how the anode varies relative to the other
 * anodes and is described by voltage dependent functions.

*/

```

int tbl_var = 1; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    9, 11, 11, 9, 9, 9, /* 000 - 005 */
    11, 11, 11, 9, 9, 11, /* 006 - 011 */
    11, 10, 11, 11 /* 012 - 015 */
};
int offset [16] = { /* offsets */
    0, 9, 20, 31, 40, 49, /* 000 - 005 */
    58, 69, 80, 91, 100, 109, /* 006 - 011 */
    120, 131, 141, 152 /* 012 - 015 */
};
int scale [163] = { /* scale factor*/
    -6, -9, -11, -14, -17, -20, -24, -28, -32, /* 000 - 008 */
    -6, -9, -11, -13, -16, -19, -22, -26, -29, /* 009 - 017 */
    -33, -37, /* 018 - 019 */
    -6, -9, -11, -14, -17, -20, -23, -26, -30, /* 020 - 028 */
    -34, -38, /* 029 - 030 */
    -6, -9, -11, -14, -17, -20, -24, -28, -32, /* 031 - 039 */
    -6, -9, -11, -14, -17, -20, -24, -27, -32, /* 040 - 048 */
    -6, -9, -11, -14, -17, -20, -24, -27, -32, /* 049 - 057 */
    -7, -10, -12, -14, -17, -20, -23, -26, -30, /* 058 - 066 */
    -34, -38, /* 067 - 068 */
    -6, -9, -12, -15, -17, -20, -23, -26, -30, /* 069 - 077 */
    -34, -38, /* 078 - 079 */
    -6, -9, -12, -14, -17, -19, -23, -26, -30, /* 080 - 088 */
    -33, -38, /* 089 - 090 */
    -6, -9, -11, -14, -17, -20, -24, -28, -32, /* 091 - 099 */
    -6, -9, -11, -14, -17, -20, -24, -27, -32, /* 100 - 108 */
    -6, -10, -12, -14, -17, -20, -23, -26, -30, /* 109 - 117 */
    -34, -38, /* 118 - 119 */
    -6, -9, -12, -14, -17, -20, -23, -26, -30, /* 120 - 128 */
    -33, -38, /* 129 - 130 */
    -6, -8, -11, -13, -16, -19, -23, -26, -30, /* 131 - 139 */
    -34, /* 140 */
    -6, -9, -12, -14, -17, -20, -23, -26, -30, /* 141 - 149 */
    -33, -38, /* 150 - 151 */
    -6, -9, -11, -14, -16, -19, -23, -26, -29, /* 152 - 160 */
    -33, -37 /* 161 - 162 */
};
int values [163] = { /* values */
    2141859, -6024497, 1794353, -2796459, /* 000 - 003 */
    2543964, -1395010, 4532808, -8024142, /* 004 - 007 */
    5954283, /* 008 */
    1660858, -6522166, 3691054, -1073737, /* 009 - 012 */
    1839852, -1977613, 1369280, -6096699, /* 013 - 016 */
    1685387, -2630885, 1771363, /* 017 - 019 */
    2021935, -5955053, 1878866, -3736588, /* 020 - 023 */
    5172668, -5034701, 3363807, -1488898, /* 024 - 027 */
    4137518, -6504663, 4399947, /* 028 - 030 */

```

```

1659460, -4954925, 1526991, -2516840, /* 031 - 034 */
2433297, -1420192, 4915142, -9275283, /* 035 - 038 */
7345382, /* 039 */
1731412, -5380326, 1709947, -2884332, /* 040 - 043 */
2826155, -1658431, 5735419, -1076440, /* 044 - 047 */
8447731, /* 048 */
1811691, -5230411, 1584896, -2629686, /* 049 - 052 */
2573125, -1519547, 5306596, -1006748, /* 053 - 056 */
7984914, /* 057 */
9984187, 2018039, -3170439, 1589455, /* 058 - 061 */
-3484247, 4156025, -2951669, 1283619, /* 062 - 065 */
-3351854, 4821632, -2933104, /* 066 - 068 */
1593066, -2964337, 3217016, 9241350, /* 069 - 072 */
-3138800, 4137379, -3044785, 1346434, /* 073 - 076 */
-3552330, 5151235, -3156380, /* 077 - 079 */
2097414, -3125151, -4329302, 4317461, /* 080 - 083 */
-9923385, 1177737, -8280358, 3573841, /* 084 - 087 */
-9293869, 1335333, -8131189, /* 088 - 090 */
2062909, -4885969, 1427184, -2260352, /* 091 - 094 */
2111094, -1194770, 4024074, -7416131, /* 095 - 098 */
5754756, /* 099 */
2180664, -6296202, 1891070, -3064724, /* 100 - 103 */
2948811, -1724577, 5997964, -1138182, /* 104 - 107 */
9060635, /* 108 */
1603139, -8534362, -8667849, 4394307, /* 109 - 112 */
-8845054, 9830094, -6619258, 2766078, /* 113 - 116 */
-7009821, 9858768, -5896570, /* 117 - 119 */
2026128, -3624418, 1755093, 2385497, /* 120 - 123 */
-6688306, 8554336, -6280654, 2795381, /* 124 - 127 */
-7448619, 1092352, -6770721, /* 128 - 130 */
4264294, -2121222, 8360316, -1760020, /* 131 - 134 */
2193857, -1690439, 8124086, -2368735, /* 135 - 138 */
3831171, -2635711, /* 139 - 140 */
1871975, -2714091, -1640371, 3116431, /* 141 - 144 */
-7626180, 9286905, -6636238, 2903883, /* 145 - 148 */
-7657761, 1117466, -6928145, /* 149 - 151 */
1714980, -7089766, 3259217, -8410868, /* 152 - 155 */
1347052, -1391524, 9397261, -4114030, /* 156 - 159 */
1123482, -1737531, 1161350 /* 160 - 162 */
};
};
struct Table7 {
    int tbl_sca_sz = 16; /* tbl_sca_sz */
    int tbl_ele_sz = 16; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 7
 * This table contains the geometric factors for each anode
 * [cm**2-sr].
 */
int tbl_var = 1; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    1, 1, 1, 1, 1, 1, /* 000 - 005 */
};

```

```

        1, 1, 1, 1, 1, 1,          /* 006 - 011 */
        1, 1, 1, 1,          /* 012 - 015 */
    };
    int offset [16] = {          /* offsets    */
        0, 1, 2, 3, 4, 5,      /* 000 - 005 */
        6, 7, 8, 9, 10, 11,   /* 006 - 011 */
        12, 13, 14, 15       /* 012 - 015 */
    };
    int scale [16] = {          /* scale factor*/
        -6, -6, -6, -6, -6, -6, /* 000 - 005 */
        -6, -6, -6, -6, -6, -6, /* 006 - 011 */
        -6, -6, -6, -6       /* 012 - 015 */
    };
    int values [16] = {        /* values     */
        588, 588, 588, 588, 588, 588, /* 000 - 005 */
        588, 588, 588, 588, 588, 588, /* 006 - 011 */
        588, 588, 588, 588     /* 012 - 015 */
    };
};
struct Table8 {
    int tbl_sca_sz = 1;        /* tbl_sca_sz */
    int tbl_ele_sz = 1;        /* tbl_ele_sz */
    int tbl_type = 0;         /* tbl_type   */
/*
 * Table 8
 * This table contains the MCP transparency factor as quoted by the
 * manufacture
 */
    int tbl_var = 1;          /* tbl_var     */
    int tbl_expand = 0;      /* tbl_expand  */
    int crit_act_sz = 0;     /* crit_act_sz */
    int format [16] = {      /* format      */
        1, 1, 1, 1, 1, 1,   /* 000 - 005 */
        1, 1, 1, 1, 1, 1,   /* 006 - 011 */
        1, 1, 1, 1         /* 012 - 015 */
    };
    int offset [16] = {      /* offsets     */
        0, 0, 0, 0, 0, 0,   /* 000 - 005 */
        0, 0, 0, 0, 0, 0,   /* 006 - 011 */
        0, 0, 0, 0         /* 012 - 015 */
    };
    int scale [1] = {-2};    /* scale factor */
    int values [1] = { 58 }; /* values      */
};
struct Table9 {
    int tbl_sca_sz = 1;        /* tbl_sca_sz */
    int tbl_ele_sz = 1;        /* tbl_ele_sz */
    int tbl_type = 0;         /* tbl_type   */
/*
 * Table 9
 * This table contains the Grid transparency factor as determined by the
 * manufactures specifications (see SwRI document ES-SGT-15-03561).
 */
    int tbl_var = 1;          /* tbl_var     */
    int tbl_expand = 0;      /* tbl_expand  */

```

```

    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {-2}; /* scale factor */
    int values [1] = { 81 }; /* values */
};
struct Table10 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 10
* This table contains the fraction of the ratio between the manufactured
* anode versus the theoretical anode.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {-2}; /* scale factor */
    int values [1] = { 87 }; /* values */
};
struct Table11 {
    int tbl_sca_sz = 16; /* tbl_sca_sz */
    int tbl_ele_sz = 16; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 11
* This table contains the amplitude adjustment factors for each anode.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
};

```

```

int offset [16] = {                                /* offsets */
    0, 1, 2, 3, 4, 5,                               /* 000 - 005 */
    6, 7, 8, 9, 10, 11,                            /* 006 - 011 */
    12, 13, 14, 15                                 /* 012 - 015 */
};
int scale [16] = {                                  /* scale factor*/
    -6, -6, -6, -6, -6, -6,                         /* 000 - 005 */
    -6, -6, -6, -6, -6, -6,                         /* 006 - 011 */
    -6, -6, -6, -6                                     /* 012 - 015 */
};
int values [16] = {                                 /* values */
    2632867, 1000000, 635386, 712443, 810931,        /* 000 - 004 */
    896400, 1370552, 928571, 665921, 1000000,       /* 005 - 009 */
    807453, 1000000, 988789, 1461922, 928571,       /* 010 - 014 */
    2146711                                           /* 015 */
};
};
struct Table12 {
    int tbl_sca_sz = 16;                             /* tbl_sca_sz */
    int tbl_ele_sz = 16;                             /* tbl_ele_sz */
    int tbl_type = 0;                                /* tbl_type */
};
/*
 * Table 12
 * This table contains the detector resolution as a fractional
 * number.
 */
int tbl_var = 1;                                    /* tbl_var */
int tbl_expand = 0;                                 /* tbl_expand */
int crit_act_sz = 0;                                /* crit_act_sz */
int format [16] = {                                 /* format */
    1, 1, 1, 1, 1, 1,                               /* 000 - 005 */
    1, 1, 1, 1, 1, 1,                               /* 006 - 011 */
    1, 1, 1, 1                                       /* 012 - 015 */
};
int offset [16] = {                                 /* offsets */
    0, 1, 2, 3, 4, 5,                               /* 000 - 005 */
    6, 7, 8, 9, 10, 11,                            /* 006 - 011 */
    12, 13, 14, 15                                 /* 012 - 015 */
};
int scale [16] = {                                  /* scale factor*/
    -5, -5, -5, -5, -5, -5,                         /* 000 - 005 */
    -5, -5, -5, -5, -5, -5,                         /* 006 - 011 */
    -5, -5, -5, -5                                     /* 012 - 015 */
};
int values [16] = {                                 /* values */
    8653, 8394, 8331, 8579, 8124, 8480,             /* 000 - 005 */
    8194, 7890, 7812, 8094, 8095, 8346,            /* 006 - 011 */
    8297, 7353, 7396, 8843                          /* 012 - 015 */
};
};
struct Table13 {
    int tbl_sca_sz = 1;                             /* tbl_sca_sz */
    int tbl_ele_sz = 1;                             /* tbl_ele_sz */
    int tbl_type = 0;                                /* tbl_type */
};
/*

```

```

* Table 13
* This table contains the conversion factor from eV to erg [erg/eV]
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {-15}; /* scale factor */
    int values [1] = {1602}; /* values */
};
struct Table14 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 14
* This factor contains the mass dependency in computing
* distribution (needed since we make computation using the
* particle energy and not velocity) and also the necessary
* scaling to put units in s**3/km***6
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {-7}; /* scale factor */
    int values [1] = {1616895}; /* values */
};
struct Table15 {
    int tbl_sca_sz = 34; /* tbl_sca_sz */
    int tbl_ele_sz = 34; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 15
* This table is mode dependent on the time summation mode which determines
* the number of spectra being accumulated into the summation of counts.
*/

```

```

int tbl_var = 0; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 17; /* crit_act_sz */
struct CriticalAction { /* crit act def */
    int status [16] = { /* status used */
        3, 3, 3, 3, 3, 3, 3, 3, /* 0000 - 0007 */
        3, 3, 3, 3, 3, 3, 3, 3 /* 0008 - 0015 */
    };
    int offset [16] = { /* act offsets */
        0, 0, 0, 0, 0, 0, 0, 0, /* 0000 - 0007 */
        0, 0, 0, 0, 0, 0, 0, 0 /* 0008 - 0015 */
    };
    int table [17] = { /* action table */
        0, 2, 4, 6, 8, 10, 12, 14, /* 0000 - 0007 */
        16, 18, 20, 22, 24, 26, 28, 30, /* 0008 - 0015 */
        32 /* 0016 */
    };
};
int format [16] = { /* format */
    2, 2, 2, 2, 2, 2, /* 000 - 005 */
    2, 2, 2, 2, 2, 2, /* 006 - 011 */
    2, 2, 2, 2 /* 012 - 015 */
};
int offset [16] = { /* offsets */
    -1, -1, -1, -1, -1, -1, /* 000 - 005 */
    -1, -1, -1, -1, -1, -1, /* 006 - 011 */
    -1, -1, -1, -1 /* 012 - 015 */
};
int scale [34] = { /* scale factor*/
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 000 - 009 */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 010 - 019 */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 020 - 029 */
    0, 0, 0, 0 /* 030 - 033 */
};
int values [34] = { /* values */
    1, 0, 1, 0, 2, 0, 3, 0, 4, 0, /* 000 - 009 */
    5, 0, 6, 0, 7, 0, 8, 0, 9, 0, /* 010 - 019 */
    10, 0, 11, 0, 12, 0, 13, 0, 14, 0, /* 020 - 029 */
    15, 0, 16, 0 /* 030 - 033 */
};
};
struct Table16 {
    int tbl_sca_sz = 6; /* tbl_sca_sz */
    int tbl_ele_sz = 6; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 16
* This table is mode dependent on the step summation mode which determines
* the number of Energy steps being accumulated into the summation of counts.
*/
int tbl_var = 0; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 3; /* crit_act_sz */
struct CriticalAction { /* crit act def */
    int status [16] = { /* status used */

```

```

        4,  4,  4,  4,  4,  4,  4,  4,      /* 0000 - 0007 */
        4,  4,  4,  4,  4,  4,  4,  4      /* 0008 - 0015 */
    };
    int offset [16] = {                      /* act offsets */
        0,  0,  0,  0,  0,  0,  0,  0,      /* 0000 - 0007 */
        0,  0,  0,  0,  0,  0,  0,  0      /* 0008 - 0015 */
    };
    int table [3] = {0, 2, 4};              /* action table */
};
int format [16] = {                        /* format */
    2,  2,  2,  2,  2,  2,                /* 000 - 005 */
    2,  2,  2,  2,  2,  2,                /* 006 - 011 */
    2,  2,  2,  2                          /* 012 - 015 */
};
int offset [16] = {                       /* offsets */
    -1, -1, -1, -1, -1, -1,              /* 000 - 005 */
    -1, -1, -1, -1, -1, -1,              /* 006 - 011 */
    -1, -1, -1, -1                          /* 012 - 015 */
};
int scale [6] = {                         /* scale factor*/
    0,  0,  0,  0,  0,  0                /* 000 - 005 */
};
int values [6] = {                        /* values */
    1,  0,  2,  0,  4,  0                /* 000 - 005 */
};
};
struct Table17 {
    int tbl_sca_sz = 2;                    /* tbl_sca_sz */
    int tbl_ele_sz = 2;                    /* tbl_ele_sz */
    int tbl_type = 0;                       /* tbl_type */
};
/*
 * Table 17
 * This table contains the sets of polynomial coefficients which
 * place the value on 1 into the working data buffer.
 */
int tbl_var = 0;                          /* tbl_var */
int tbl_expand = 0;                        /* tbl_expand */
int crit_act_sz = 0;                       /* crit_act_sz */
int format [16] = {                       /* format */
    2,  2,  2,  2,  2,  2,                /* 000 - 005 */
    2,  2,  2,  2,  2,  2,                /* 006 - 011 */
    2,  2,  2,  2                          /* 012 - 015 */
};
int offset [16] = {                       /* offsets */
    0,  0,  0,  0,  0,  0,                /* 000 - 005 */
    0,  0,  0,  0,  0,  0,                /* 006 - 011 */
    0,  0,  0,  0                          /* 012 - 015 */
};
int scale [2] = {0, 0};                   /* scale factor */
int values [2] = {1, 0};                  /* values */
};
struct Table18 {
    int tbl_sca_sz = 2;                    /* tbl_sca_sz */
    int tbl_ele_sz = 2;                    /* tbl_ele_sz */
    int tbl_type = 0;                       /* tbl_type */
};

```

```

/*
* Table 18
* This table is a function of calibration set 5, Low Range Sweep
* Summation (MSB). This table is a polynomial to adjust the value to
* the upper bytes of a 4-byte integer.
*/
    int tbl_var = -6; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, 0}; /* scale factor */
    int values [2] = {0, 65536}; /* values */
};
struct Table19 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 19
* This table is a function of calibration set 6, Low Range Sweep
* Summation (LSB). This table is a polynomial to adjust the value to
* the lower bytes of a 4-byte integer.
*/
    int tbl_var = -7; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, 0}; /* scale factor */
    int values [2] = {0, 1}; /* values */
};
struct Table20 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 20
* This table is a function of calibration set 7, Total Range Sweep

```

```

* Summation (MSB). This table is a polynomial to adjust the value to
* the upper bytes of a 4-byte integer.
*/
    int tbl_var = -8;          /* tbl_var      */
    int tbl_expand = 0;       /* tbl_expand   */
    int crit_act_sz = 0;     /* crit_act_sz  */
    int format [16] = {      /* format       */
        2, 2, 2, 2, 2, 2,    /* 000 - 005   */
        2, 2, 2, 2, 2, 2,    /* 006 - 011   */
        2, 2, 2, 2          /* 012 - 015   */
    };
    int offset [16] = {      /* offsets      */
        0, 0, 0, 0, 0, 0,    /* 000 - 005   */
        0, 0, 0, 0, 0, 0,    /* 006 - 011   */
        0, 0, 0, 0          /* 012 - 015   */
    };
    int scale [2] = {0, 0};  /* scale factor */
    int values [2] = {0, 65536}; /* values      */
};
struct Table21 {
    int tbl_sca_sz = 2;      /* tbl_sca_sz   */
    int tbl_ele_sz = 2;     /* tbl_ele_sz   */
    int tbl_type = 0;       /* tbl_type     */
};
/*
* Table 21
* This table is a function of calibration set 8, Total Range Sweep
* Summation (LSB). This table is a polynomial to adjust the value to
* the lower bytes of a 4-byte integer.
*/
    int tbl_var = -9;          /* tbl_var      */
    int tbl_expand = 0;       /* tbl_expand   */
    int crit_act_sz = 0;     /* crit_act_sz  */
    int format [16] = {      /* format       */
        2, 2, 2, 2, 2, 2,    /* 000 - 005   */
        2, 2, 2, 2, 2, 2,    /* 006 - 011   */
        2, 2, 2, 2          /* 012 - 015   */
    };
    int offset [16] = {      /* offsets      */
        0, 0, 0, 0, 0, 0,    /* 000 - 005   */
        0, 0, 0, 0, 0, 0,    /* 006 - 011   */
        0, 0, 0, 0          /* 012 - 015   */
    };
    int scale [2] = {0, 0};  /* scale factor */
    int values [2] = {0, 1}; /* values      */
};
struct Table22 {
    int tbl_sca_sz = 2;      /* tbl_sca_sz   */
    int tbl_ele_sz = 2;     /* tbl_ele_sz   */
    int tbl_type = 0;       /* tbl_type     */
};
/*
* Table 22
* This table is a function of calibration set 3, Total Range Anode
* Summation (MSB). This table is a polynomial to adjust the value to
* the upper bytes of a 4-byte integer.
*/

```

```

    int tbl_var = -4;                /* tbl_var      */
    int tbl_expand = 0;              /* tbl_expand   */
    int crit_act_sz = 0;             /* crit_act_sz  */
    int format [16] = {              /* format       */
        2, 2, 2, 2, 2, 2,           /* 000 - 005   */
        2, 2, 2, 2, 2, 2,           /* 006 - 011   */
        2, 2, 2, 2,                /* 012 - 015   */
    };
    int offset [16] = {              /* offsets      */
        0, 0, 0, 0, 0, 0,           /* 000 - 005   */
        0, 0, 0, 0, 0, 0,           /* 006 - 011   */
        0, 0, 0, 0,                /* 012 - 015   */
    };
    int scale [2] = {0, 0};          /* scale factor */
    int values [2] = {0, 65536};     /* values       */
};
struct Table23 {
    int tbl_sca_sz = 2;              /* tbl_sca_sz   */
    int tbl_ele_sz = 2;              /* tbl_ele_sz   */
    int tbl_type = 0;                /* tbl_type     */
};
/*
 * Table 23
 * This table is a function of calibration set 4, Total Range Anode
 * Summation (LSB). This table is a polynomial to adjust the value to
 * the lower bytes of a 4-byte integer.
 */
    int tbl_var = -5;                /* tbl_var      */
    int tbl_expand = 0;              /* tbl_expand   */
    int crit_act_sz = 0;             /* crit_act_sz  */
    int format [16] = {              /* format       */
        2, 2, 2, 2, 2, 2,           /* 000 - 005   */
        2, 2, 2, 2, 2, 2,           /* 006 - 011   */
        2, 2, 2, 2,                /* 012 - 015   */
    };
    int offset [16] = {              /* offsets      */
        0, 0, 0, 0, 0, 0,           /* 000 - 005   */
        0, 0, 0, 0, 0, 0,           /* 006 - 011   */
        0, 0, 0, 0,                /* 012 - 015   */
    };
    int scale [2] = {0, 0};          /* scale factor */
    int values [2] = {0, 1};        /* values       */
};
struct Table24 {
    int tbl_sca_sz = 2;              /* tbl_sca_sz   */
    int tbl_ele_sz = 2;              /* tbl_ele_sz   */
    int tbl_type = 0;                /* tbl_type     */
};
/*
 * Table 24
 * This table contains the sets of polynomial coefficients which
 * converts calibration set 0 telemetry to voltage.
 */
    int tbl_var = -1;                /* tbl_var      */
    int tbl_expand = 0;              /* tbl_expand   */
    int crit_act_sz = 0;             /* crit_act_sz  */
    int format [16] = {              /* format       */

```

```

        2, 2, 2, 2, 2, 2,          /* 000 - 005 */
        2, 2, 2, 2, 2, 2,          /* 006 - 011 */
        2, 2, 2, 2,                /* 012 - 015 */
    };
    int offset [16] = {             /* offsets    */
        0, 0, 0, 0, 0, 0,          /* 000 - 005 */
        0, 0, 0, 0, 0, 0,          /* 006 - 011 */
        0, 0, 0, 0,                /* 012 - 015 */
    };
    int scale [2] = {0, -8};        /* scale factor */
    int values [2] = {0, 1960784}; /* values      */
};
struct Table25 {
    int tbl_sca_sz = 2;            /* tbl_sca_sz  */
    int tbl_ele_sz = 2;            /* tbl_ele_sz  */
    int tbl_type = 0;              /* tbl_type    */
};
/*
 * Table 25
 * This table contains the sets of polynomial coefficients which
 * converts calibration set 0 telemetry to microamp.
 */
    int tbl_var = -1;              /* tbl_var     */
    int tbl_expand = 0;            /* tbl_expand  */
    int crit_act_sz = 0;           /* crit_act_sz */
    int format [16] = {           /* format      */
        2, 2, 2, 2, 2, 2,          /* 000 - 005 */
        2, 2, 2, 2, 2, 2,          /* 006 - 011 */
        2, 2, 2, 2,                /* 012 - 015 */
    };
    int offset [16] = {           /* offsets    */
        0, 0, 0, 0, 0, 0,          /* 000 - 005 */
        0, 0, 0, 0, 0, 0,          /* 006 - 011 */
        0, 0, 0, 0,                /* 012 - 015 */
    };
    int scale [2] = {0, -6};        /* scale factor */
    int values [2] = {0, 1620483}; /* values      */
};
struct Table26 {
    int tbl_sca_sz = 2;            /* tbl_sca_sz  */
    int tbl_ele_sz = 2;            /* tbl_ele_sz  */
    int tbl_type = 0;              /* tbl_type    */
};
/*
 * Table 26
 * This table contains the sets of polynomial coefficients which
 * converts calibration set 0 telemetry to degrees C.
 */
    int tbl_var = -1;              /* tbl_var     */
    int tbl_expand = 0;            /* tbl_expand  */
    int crit_act_sz = 0;           /* crit_act_sz */
    int format [16] = {           /* format      */
        2, 2, 2, 2, 2, 2,          /* 000 - 005 */
        2, 2, 2, 2, 2, 2,          /* 006 - 011 */
        2, 2, 2, 2,                /* 012 - 015 */
    };
    int offset [16] = {           /* offsets    */

```

```

        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [2] = {-1, -6};
    int values [2] = {-2732, 1620483};
};
struct Table27 {
    int tbl_sca_sz = 1;
    int tbl_ele_sz = 1;
    int tbl_type = 0;
};
/*
* Table 27
* This table contains the conversion factor from eV to Joule [Joule/eV]
*/
    int tbl_var = 2;
    int tbl_expand = 0;
    int crit_act_sz = 0;
    int format [16] = {
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1
    };
    int offset [16] = {
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [1] = {-22};
    int values [1] = {1602};
};
struct Table28 {
    int tbl_sca_sz = 1;
    int tbl_ele_sz = 1;
    int tbl_type = 0;
};
/*
* Table 28
* This table contains the constant number 2 for conversion used in energy
* to velocity.
*/
    int tbl_var = 2;
    int tbl_expand = 0;
    int crit_act_sz = 0;
    int format [16] = {
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1
    };
    int offset [16] = {
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [1] = {0};
    int values [1] = {2};
};

```

```

};
struct Table29 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 29
* This table contains the electron mass in [kg].
*/
    int tbl_var = 2;             /* tbl_var */
    int tbl_expand = 0;          /* tbl_expand */
    int crit_act_sz = 0;         /* crit_act_ele */
    int format [16] = {         /* format */
        1, 1, 1, 1, 1, 1,       /* 000 - 005 */
        1, 1, 1, 1, 1, 1,       /* 006 - 011 */
        1, 1, 1, 1,             /* 012 - 015 */
    };
    int offset [16] = {         /* offsets */
        0, 0, 0, 0, 0, 0,       /* 000 - 005 */
        0, 0, 0, 0, 0, 0,       /* 006 - 011 */
        0, 0, 0, 0,             /* 012 - 015 */
    };
    int scale [1] = {-33};      /* scale factor */
    int values [1] = {911};     /* values */
};
struct Table30 {
    int tbl_sca_sz = 1;         /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 30
* This table contains the constant number 1000 for converting from m to km.
* This Table contains the number of m in a km [m/km]
*/
    int tbl_var = 1;             /* tbl_var */
    int tbl_expand = 0;          /* tbl_expand */
    int crit_act_sz = 0;         /* crit_act_ele */
    int format [16] = {         /* format */
        1, 1, 1, 1, 1, 1,       /* 000 - 005 */
        1, 1, 1, 1, 1, 1,       /* 006 - 011 */
        1, 1, 1, 1,             /* 012 - 015 */
    };
    int offset [16] = {         /* offsets */
        0, 0, 0, 0, 0, 0,       /* 000 - 005 */
        0, 0, 0, 0, 0, 0,       /* 006 - 011 */
        0, 0, 0, 0,             /* 012 - 015 */
    };
    int scale [1] = {3};        /* scale factor */
    int values [1] = {1};       /* values */
};
struct Table31 {
    int tbl_sca_sz = 1;         /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*

```

```

* Table 31
* This table contains the constant number 1000 for converting from m/s
* to km/s. This Table contains the number of m/s in a km/s [m/km]
*/
    int tbl_var = 2; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_ele */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {3}; /* scale factor */
    int values [1] = {1}; /* values */
};
struct Table32 {
    int tbl_sca_sz = 0; /* tbl_sca_sz */
    int tbl_ele_sz = 9; /* tbl_ele_sz */
    int tbl_type = 1; /* tbl_type */
}
/*
* Table 32
* This table is an ASCII look-up table which indicates what
* the settings of the bit status monitors represents.
*/
    int tbl_var = 4; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [23] = { /* format */
        -1, -1, 0, -1, -1, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0, 0, 0, /* 012 - 017 */
        0, 0, 0, 0, -1 /* 018 - 022 */
    };
    int offset [23] = { /* offsets */
        -1, -1, 4, -1, -1, 0, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2, 2, 2, /* 012 - 017 */
        2, 2, 2, 2, -1 /* 018 - 022 */
    };
    string values [9] = { /* values */
        "Off", "On", "Disabled", /* 000 - 002 */
        "Enabled", "Undefined", "Booting", /* 003 - 005 */
        "Safe", "Prom", "Normal" /* 006 - 008 */
    };
};
struct Table33 {
    int tbl_sca_sz = 20; /* tbl_sca_sz */
    int tbl_ele_sz = 20; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*

```

```

* Table 33
* This table is a look-up table which indicates the normalization factor
* for the science data. It applies to Time and Step summations.
*/
int tbl_var = 4; /* tbl_var */
int tbl_expand = 1; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [23] = { /* format */
    -1, -1, -1, 0, 0, -1, /* 000 - 005 */
    -1, -1, -1, -1, -1, -1, /* 006 - 011 */
    -1, -1, -1, -1, -1, -1, /* 012 - 017 */
    -1, -1, -1, -1, -1 /* 018 - 022 */
};
int offset [23] = { /* offsets */
    -1, -1, -1, 3, 0, -1, /* 000 - 005 */
    -1, -1, -1, -1, -1, -1, /* 006 - 011 */
    -1, -1, -1, -1, -1, -1, /* 012 - 017 */
    -1, -1, -1, -1, -1 /* 018 - 022 */
};
int scale [20] = { /* scale factor */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 000 - 009 */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0 /* 010 - 019 */
};
int values [20] = { /* values */
    1, 2, 4, 1, 1, 2, 3, 4, 5, 6, /* 000 - 009 */
    7, 8, 9, 10, 11, 12, 13, 14, 15, 16 /* 010 - 019 */
};
};
struct Constant0 {
    int id = 1;
/*
* Elevation angle as measured from the spacecraft +C axis
*/
int scale[16] = { -2, -2, -2, -2, -2, -2, -2, -2,
    -2, -2, -2, -2, -2, -2, -2, -2 };
int values[16] = { -16875, -14625, -12375, -10125,
    -7875, -5625, -3375, -1125,
    1125, 3375, 5625, 7875,
    10125, 12375, 14625, 16875 };
};
struct Constant1 {
    int id = 2;
/*
* Azimuthal angle offsets measured from the spacecraft +A axis
* in the direction of the spacecraft +B axis
*/
int scale[16] = { 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0 };
int values[16] = { 270, 270, 270, 270, 270, 270, 270, 270,
    90, 90, 90, 90, 90, 90, 90, 90 };
};
struct Constant2 {
    int id = 3;
/*
* Azimuthal field of view in the frame of the spacecraft (spacecraft

```

```

*      A-B plane)
*/
    int scale[16] = {  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
                      -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1 };
    int values[16] = {  50,  50,  50,  50,  50,  50,  50,  50,
                      50,  50,  50,  50,  50,  50,  50,  50 };
};
struct Constant3 {
    int id = 4;
/*
*      Initial Aperture Elevation Angle as measured from the spacecraft
*      +C axis
*/
    int scale[16] = {  -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2,
                      -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2 };
    int values[16] = { 15925, 13675, 11425,  9175,
                      6925,  4675,  2425,  175,
                      175,  2425,  4675,  6925,
                      9175, 11425, 13675, 15925 };
};
struct Constant4 {
    int id = 5;
/*
*      Final Aperture Elevation Angle as measured from the spacecraft
*      +C axis
*/
    int scale[16] = {  -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2,
                      -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2 };
    int values[16] = { 17825, 15575, 13325, 11075,
                      8825,  6575,  4325,  2075,
                      2075,  4325,  6575,  8825,
                      11075, 13325, 15575, 17825 };
};
struct Constant5 {
    int id = 6;
/*
*      Component of the aperture normal vector with respect to the +A axis
*      of the magnetometer.
*/
    int scale[16] = {  0,  0,  0,  0,  0,  0,  0,  0,
                      0,  0,  0,  0,  0,  0,  0,  0 };
    int values[16] = {  0,  0,  0,  0,
                      0,  0,  0,  0,
                      0,  0,  0,  0,
                      0,  0,  0,  0 };
};
struct Constant6 {
    int id = 7;
/*
*      Component of the aperture normal vector with respect to the +B axis
*      of the magnetometer.
*/
    int scale[16] = {  -6,  -6,  -6,  -6,  -6,  -6,  -6,  -6,
                      -6,  -6,  -6,  -6,  -6,  -6,  -6,  -6 };
    int values[16] = { -195090, -555570, -831470, -980785,

```

```
        -980785, -831470, -555570, -195090,  
        195090, 555570, 831470, 980785,  
        980785, 831470, 555570, 195090 };  
};  
struct Constant7 {  
    int id = 8;  
/*  
 *   Component of the aperture normal vector with respect to the +C axis  
 *   of the magnetometer.  
 */  
    int scale[16] = { -6, -6, -6, -6, -6, -6, -6, -6,  
                    -6, -6, -6, -6, -6, -6, -6, -6 };  
    int values[16] = { -980785, -831470, -555570, -195090,  
                    195090, 555570, 831470, 980785,  
                    980785, 831470, 555570, 195090,  
                    -195090, -555570, -831470, -980785 };  
};  
}
```

ELSSCIH VIDF File

```

vidf v3_ELSSCIH {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft    */
    string experiment = "ASPERA-3";     /* exp_desc     */
    string instrument = "ELS";          /* inst_desc    */
    string contact = "Dr. Rickard Lundin";
    string contact = "Swedish Institute of Space Physics";
    string contact = "Box 812 S-981 28.Kiruna";
    string contact = "Sweden";
    string contact = "rickard@irf.se";
/*
* This VIDF contains the science data from ELS. Data is stored
* for the high range of the ELS power supply as a function of the
* DAC representation. The DAC range is 12 bits representing a +5V
* control voltage. This control voltage sets the ELS deflection
* supply which has a range of 0V to 2800V for the high range.
* Stored data value is the number of counts received in the
* accumulation interval of the ELS packets.
*
* The following is a list of tables which are in this vidf
* TABLE 0: Polynomial coefficients which places the data into
* math buffer.
* TABLE 1: Polynomial coefficients which converts sensor index
* into fraction of voltage range.
* TABLE 2: Maximum control input voltage [volts].
* TABLE 3: Maximum deflection high range voltage [volts].
* TABLE 4: Anode K-factors [eV/Volt].
* TABLE 5: Detector Efficiencies.
* TABLE 6: Efficiency Adjustment Factor For Each Anode [fractional
* number]
* TABLE 7: Geometric Factors [cm**2-sr]
* TABLE 8: MCP Transparency [fraction]
* TABLE 9: Grid Transparency [fraction]
* TABLE 10: Active Anode Area Ratio [fraction]
* TABLE 11: Amplitude Adjustment or Scaling Factor
* TABLE 12: Detector Resolution [fractional number]
* TABLE 13: Energy Conversion [erg/eV]
* TABLE 14: Conversion from Energy Intensity to Distribution Funct
* TABLE 15: Number of spectra summed together
* TABLE 16: Number of steps summed together
* TABLE 17: Polynomial coefficients which places the value on 1 into the
* math buffer.
* TABLE 18: Shift Calibration Set 1 to Upper bytes of 4-byte integer -
* High Range Sweep Summation (MSB)
* TABLE 19: Shift Calibration Set 2 to Lower bytes of 4-byte integer -
* High Range Sweep Summation (LSB)
* TABLE 20: Shift Calibration Set 3 to Upper bytes of 4-byte integer -
* Total Range Sweep Summation (MSB)
* TABLE 21: Shift Calibration Set 4 to Lower bytes of 4-byte integer -
* Total Range Sweep Summation (LSB)

```

- * TABLE 22: Shift Calibration Set 5 to Upper bytes of 4-byte integer -
- * Total Range Anode Summation (MSB)
- * TABLE 23: Shift Calibration Set 6 to Lower bytes of 4-byte integer -
- * Total Range Anode Summation (LSB)
- * TABLE 24: Convert Temperature Calibration (cal set 0) Value to Voltage
- * TABLE 25: Convert Temperature Calibration (cal set 0) Value to milliamp
- * TABLE 26: Convert Temperature Calibration (cal set 0) Value to degrees C
- * TABLE 27: Convert from eV to Joule [Joule/eV]
- * TABLE 28: Constant number 2 for conversion used in energy to velocity
- * TABLE 29: Contains the electron mass in [kg]
- * TABLE 30: Contains the constant number 1000 for converting from m to km
- * TABLE 31: Contains the constant number 1000 for converting m/s into km/s
- * TABLE 32: ASCII strings which define what state the bit values
- * represent for the status flags defined
- * TABLE 33: Table conversion factors to determine number of summations

* The following are units which can be derived from the tables.
 * The format is to give the tables applied followed by the
 * operations and unit definition

| * DATA TYPE | * TABLES | * OPERS | * UNIT |
|-------------|--------------|--------------------|------------------------|
| * Scan | 1 | 0 | Dimensionless fraction |
| * Scan | 1,2 | 0,3 | control voltage |
| * Scan | 1,3 | 0,3 | deflection voltage |
| * Scan | 1,3,4 | 0,3,3 | eV |
| * Scan | 1,3,4,27 | 0,3,3,3 | Joule |
| * Scan | 1,3,4,27,28, | 0,3,3,3,3, | m/s |
| * Scan | 29 | 64 | |
| * Scan | 1,3,4,27,28, | 0,3,3,3,3, | km/s |
| * Scan | 29,31 | 64,3 | |
| * Sweep | 0 | 0 | cnts/accum |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/accum (eff. cor) |
| * Sweep | 0,5,-1 | 0,4,2003 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/sec |
| * Sweep | 0,5,-1 | 150,4.2003 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/(cm**2-str-s) |
| * Sweep | 0,5,-1,7,11 | 150,4,2003,4,3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | cnts/(cm**2-str-s-eV) |
| * Sweep | 0,5,-1,7,1, | 150,4,2003,4,1000, | |
| * Sweep | 3,4,-1, | 1003,1003,2004, | |
| * Sweep | 12,11 | 4,3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | ergs/(cm**2-str-s-eV) |
| * Sweep | 0,5,-1,7, | 150,4,2003,4, | |
| * Sweep | 12,13 | 4,3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | sec**3/km**6 |
| * Sweep | 0,5,-1,7,1, | 150,4,2003,4,1000, | |
| * Sweep | 3,4,-1, | 1003,1003,2004, | |
| * Sweep | 12,-1,14, | 4,2004,3, | |
| * Sweep | 11 | 3 | |
| * Sweep | 1,3,6, | 1000,1003,1000, | sec**3/m**6 |
| * Sweep | 0,5,-1,7,1, | 150,4,2003,4,1000, | |
| * Sweep | 3,4,-1, | 1003,1003,2004, | |
| * Sweep | 12,-1,14, | 4,2004,3, | |
| * Sweep | 11,30,30,30, | 3,4,4,4, | |
| * Sweep | 30,30,30 | 4,4,4 | |

```

*   Sweep      15           0           Number Spectra Summed
*   Sweep      16           0           Number Energies Summed
*   Cal        18,19        0,1        High Range Sweep Sum
*   Cal        20,21        0,1        Total Range Sweep Sum
*   Cal        22,23        0,1        Total Anode Sum or Sum on MCP
*   Cal        24           0           Temperature in monitor voltage
*   Cal        25           0           Temp. in monitor milliamp
*   Cal        26           0           Temperature in degrees C
*   Mode       31           0           ASCII Mode Strings
*   Mode       32           0           # Spectra in Time Sum
*   Mode       32           0           # Energy Levels in Energy Sum
*
*   Data_len is set to the maximum, assuming that all sensors are returned.
*   (128 steps + 4 cal.) * 2 bytes * 16 sensors + 5 cal. * 2 bytes +
*   4 (nanosecond timing) + 20 bytes for the rest of the elements in the
*   data record.
*
*/
int s_year = 2003;           /* ds_year      */
int s_day = 1;              /* ds_day       */
int s_msec = 0;            /* ds_msec      */
int s_usec = 0;           /* ds_usec      */
int e_year = 2010;        /* de_year      */
int e_day = 1;            /* de_day       */
int e_msec = 0;           /* de_msec      */
int e_usec = 0;           /* de_usec      */
int smp_id = 1;           /* smp_id       */
int sen_mode = 2;         /* sen_mode     */
int n_qual = 5;           /* n_qual       */
int n_cal_sets = 9;       /* cal_sets     */
int n_tbls = 34;          /* num_tbls     */
int n_consts = 8;         /* num_consts   */
int n_status = 23;        /* status       */
int n_sensors = 16;       /* sen          */
int swp_len = 4096;       /* swp_len      */
int max_nss = 1;         /* max_nss      */
int data_len = 4258;      /* data_len     */
int fill_flag = 1;        /* fill_flg     */
int fill = 65535;         /* fill value   */
int da_method = 0;        /* da_method    */
int nano_defined = 1;     /* nsec timetag */
struct PitchAngle {
    int format = 1;
    string project = "MARS";
    string mission = "Mars_Express";
    string experiment = "MODELS";
    string instrument = "MAG";
    string vinstrument = "SAF_BMod";
    int bx = 0;
    int by = 1;
    int bz = 2;
    int num_tbls = 0;
};
struct Status0 {
    string name = "Software Version - Upper Byte"; /* name      */

```

```
        int state = 255;                                /* state    */
};
struct Status1 {
    string name = "Software Version - Lower Byte";    /* name      */
    int state = 255;                                  /* state    */
};
struct Status2 {
    string name = "Software Mode";                    /* name      */
    int state = 5;                                    /* state    */
};
struct Status3 {
    string name = "Time Summation";                    /* name      */
    int state = 17;                                   /* state    */
};
struct Status4 {
    string name = "Step Summation";                    /* name      */
    int state = 3;                                    /* state    */
};
struct Status5 {
    string name = "Log Compression";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status6 {
    string name = "Sector 0 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status7 {
    string name = "Sector 1 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status8 {
    string name = "Sector 2 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status9 {
    string name = "Sector 3 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status10 {
    string name = "Sector 4 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status11 {
    string name = "Sector 5 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status12 {
    string name = "Sector 6 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status13 {
    string name = "Sector 7 Enable";                    /* name      */
    int state = 2;                                    /* state    */
};
struct Status14 {
```

```

        string name = "Sector 8 Enable";           /* name      */
        int state = 2;                             /* state     */
};
struct Status15 {
    string name = "Sector 9 Enable";             /* name      */
    int state = 2;                              /* state     */
};
struct Status16 {
    string name = "Sector 10 Enable";           /* name      */
    int state = 2;                             /* state     */
};
struct Status17 {
    string name = "Sector 11 Enable";           /* name      */
    int state = 2;                             /* state     */
};
struct Status18 {
    string name = "Sector 12 Enable";           /* name      */
    int state = 2;                             /* state     */
};
struct Status19 {
    string name = "Sector 13 Enable";           /* name      */
    int state = 2;                             /* state     */
};
struct Status20 {
    string name = "Sector 14 Enable";           /* name      */
    int state = 2;                             /* state     */
};
struct Status21 {
    string name = "Sector 15 Enable";           /* name      */
    int state = 2;                             /* state     */
};
struct Status22 {
    string name = "Number Of Active Anodes";    /* name      */
    int state = 16;                            /* state     */
};
string qual_names = "Good Data";              /* name */
string qual_names = "Questionable Data";      /* name */
string qual_names = "Invalid Data";           /* name */
string qual_names = "Bad Data";               /* name */
string qual_names = "Unknown State";         /* name */
struct Sensor0 {
    string name = "ELS Anode 0";               /* name      */
    int d_type = 0;                            /* d_type    */
    int status = 1;                            /* status    */
    int tdw_len = 16;                          /* tdw_len   */
    int time_offset = 0;                       /* time_offset */
};
struct Sensor1 {
    string name = "ELS Anode 1";               /* name      */
    int d_type = 0;                            /* d_type    */
    int status = 1;                            /* status    */
    int tdw_len = 16;                          /* tdw_len   */
    int time_offset = 0;                       /* time_offset */
};
struct Sensor2 {

```

```

    string name = "ELS Anode 2";           /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor3 {
    string name = "ELS Anode 3";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor4 {
    string name = "ELS Anode 4";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor5 {
    string name = "ELS Anode 5";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor6 {
    string name = "ELS Anode 6";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor7 {
    string name = "ELS Anode 7";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor8 {
    string name = "ELS Anode 8";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
struct Sensor9 {
    string name = "ELS Anode 9";         /* name          */
    int d_type = 0;                       /* d_type        */
    int status = 1;                       /* status        */
    int tdw_len = 16;                    /* tdw_len       */
    int time_offset = 0;                  /* time_offset   */
};
};

```

```

struct Sensor10 {
    string name = "ELS Anode 10";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                   /* time_offset   */
};
struct Sensor11 {
    string name = "ELS Anode 11";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                   /* time_offset   */
};
struct Sensor12 {
    string name = "ELS Anode 12";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                   /* time_offset   */
};
struct Sensor13 {
    string name = "ELS Anode 13";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                   /* time_offset   */
};
struct Sensor14 {
    string name = "ELS Anode 14";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                   /* time_offset   */
};
struct Sensor15 {
    string name = "ELS Anode 15";           /* name          */
    int d_type = 0;                         /* d_type        */
    int status = 1;                         /* status        */
    int tdw_len = 16;                       /* tdw_len       */
    int time_offset = 0;                   /* time_offset   */
};
struct CalSet0 {
    string name = "ELS Temperature";        /* name          */
    int use = 0;                            /* use           */
    int word_len = 8;                       /* word length   */
    int target = 0;                         /* target        */
    int cal_scope = 1;                      /* scope         */
};
struct CalSet1 {
    string name = "ELS MCP Bias Reference"; /* name          */
    int use = 0;                            /* use           */
    int word_len = 8;                       /* word length   */
    int target = 0;                         /* target        */
    int cal_scope = 1;                      /* scope         */
};

```

```

};
struct CalSet2 {
    string name = "ELS MCP Bias Monitor";           /* name      */
    int use = 0;                                   /* use       */
    int word_len = 8;                              /* word length */
    int target = 0;                                /* target    */
    int cal_scope = 1;                             /* scope     */
};
struct CalSet3 {
    string name = "Total Range Anode Summation (MSB)"; /* name      */
    int use = 0;                                   /* use       */
    int word_len = 16;                             /* word length */
    int target = 0;                                /* target    */
    int cal_scope = 1;                             /* scope     */
};
struct CalSet4 {
    string name = "Total Range Anode Summation (LSB)"; /* name      */
    int use = 0;                                   /* use       */
    int word_len = 16;                             /* word length */
    int target = 0;                                /* target    */
    int cal_scope = 1;                             /* scope     */
};
struct CalSet5 {
    string name = "High Range Sweep Summation (MSB)"; /* name      */
    int use = 0;                                   /* use       */
    int word_len = 16;                             /* word length */
    int target = 0;                                /* target    */
    int cal_scope = 0;                             /* scope     */
};
struct CalSet6 {
    string name = "High Range Sweep Summation (LSB)"; /* name      */
    int use = 0;                                   /* use       */
    int word_len = 16;                             /* word length */
    int target = 0;                                /* target    */
    int cal_scope = 0;                             /* scope     */
};
struct CalSet7 {
    string name = "Total Range Sweep Summation (MSB)"; /* name      */
    int use = 0;                                   /* use       */
    int word_len = 16;                             /* word length */
    int target = 0;                                /* target    */
    int cal_scope = 0;                             /* scope     */
};
struct CalSet8 {
    string name = "Total Range Sweep Summation (LSB)"; /* name      */
    int use = 0;                                   /* use       */
    int word_len = 16;                             /* word length */
    int target = 0;                                /* target    */
    int cal_scope = 0;                             /* scope     */
};
struct Table0 {
    int tbl_sca_sz = 2;                            /* tbl_sca_sz */
    int tbl_ele_sz = 2;                            /* tbl_ele_sz */
    int tbl_type = 0;                              /* tbl_type   */
};
/*

```

```

* Table 0
* This table contains the sets of polynomial coefficients which
* put the data value into the working math buffer.
*/
    int tbl_var = 0; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, 0}; /* scale factor */
    int values [2] = {0, 1}; /* values */
};
struct Table1 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 1
* This table contains the 1/index maximum value of the voltage DAC.
* Dividing the control index by this number gives the fractional
* amount of the voltage range used in deflecting the electron.
*/
    int tbl_var = 2; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, -12}; /* scale factor */
    int values [2] = {0, 244200244}; /* values */
};
struct Table2 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 2
* This table contains the maximum control input voltage [volt].
*/
    int tbl_var = 1; /* tbl_var */

```

```

    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1, /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0, /* 012 - 015 */
    };
    int scale [1] = {0}; /* scale factor */
    int values [1] = {5}; /* values */
};
struct Table3 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 3
 * This table contains the maximum high range deflection voltage
 * [volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1, /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0, /* 012 - 015 */
    };
    int scale [1] = {0}; /* scale factor */
    int values [1] = {2800}; /* values */
};
struct Table4 {
    int tbl_sca_sz = 16; /* tbl_sca_sz */
    int tbl_ele_sz = 16; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 4
 * This table contains the k-factors for each anode [eV/volt].
 */
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1, /* 012 - 015 */
    };

```

```

};
int offset [16] = {                                /* offsets      */
    0, 1, 2, 3, 4, 5,                             /* 000 - 005   */
    6, 7, 8, 9, 10, 11,                          /* 006 - 011   */
    12, 13, 14, 15                               /* 012 - 015   */
};
int scale [16] = {                                /* scale factor*/
    -3, -3, -3, -3, -3, -3,                       /* 000 - 005   */
    -3, -3, -3, -3, -3, -3,                       /* 006 - 011   */
    -3, -3, -3, -3                               /* 012 - 015   */
};
int values [16] = {                               /* values       */
    7167, 7152, 7141, 7165, 7188, 7625,           /* 000 - 005   */
    7262, 7266, 7275, 7254, 7262, 7255,         /* 006 - 011   */
    7255, 7271, 7253, 7188                       /* 012 - 015   */
};
};
struct Table5 {
    int tbl_sca_sz = 16;                          /* tbl_sca_sz   */
    int tbl_ele_sz = 16;                          /* tbl_ele_sz   */
    int tbl_type = 0;                             /* tbl_type     */
};
/*
 * Table 5
 * This table contains the detector efficiencies for the detector
 */
int tbl_var = 1;                                  /* tbl_var      */
int tbl_expand = 0;                              /* tbl_expand   */
int crit_act_sz = 0;                             /* crit_act_sz  */
int format [16] = {                              /* format       */
    1, 1, 1, 1, 1, 1,                             /* 000 - 005   */
    1, 1, 1, 1, 1, 1,                             /* 006 - 011   */
    1, 1, 1, 1                                   /* 012 - 015   */
};
int offset [16] = {                              /* offsets      */
    0, 1, 2, 3, 4, 5,                             /* 000 - 005   */
    6, 7, 8, 9, 10, 11,                          /* 006 - 011   */
    12, 13, 14, 15                               /* 012 - 015   */
};
int scale [16] = {                              /* scale factor*/
    -2, -2, -2, -2, -2, -2,                       /* 000 - 005   */
    -2, -2, -2, -2, -2, -2,                       /* 006 - 011   */
    -2, -2, -2, -2                               /* 012 - 015   */
};
int values [16] = {                              /* values       */
    95, 95, 95, 95, 95, 95,                       /* 000 - 005   */
    95, 95, 95, 95, 95, 95,                       /* 006 - 011   */
    95, 95, 95, 95                               /* 012 - 015   */
};
};
struct Table6 {
    int tbl_sca_sz = 163;                          /* tbl_sca_sz   */
    int tbl_ele_sz = 163;                          /* tbl_ele_sz   */
    int tbl_type = 0;                             /* tbl_type     */
};
/*
 * Table 6

```

* This table contains the efficiency adjustment factor for the detector.
 * This is a description of how the anode varies relative to the other
 * anodes and is described by voltage dependent functions.

*/

```

int tbl_var = 1; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    9, 11, 11, 9, 9, 9, /* 000 - 005 */
    11, 11, 11, 9, 9, 11, /* 006 - 011 */
    11, 10, 11, 11 /* 012 - 015 */
};
int offset [16] = { /* offsets */
    0, 9, 20, 31, 40, 49, /* 000 - 005 */
    58, 69, 80, 91, 100, 109, /* 006 - 011 */
    120, 131, 141, 152 /* 012 - 015 */
};
int scale [163] = { /* scale factor*/
    -6, -9, -11, -14, -17, -20, -24, -28, -32, /* 000 - 008 */
    -6, -9, -11, -13, -16, -19, -22, -26, -29, /* 009 - 017 */
    -33, -37, /* 018 - 019 */
    -6, -9, -11, -14, -17, -20, -23, -26, -30, /* 020 - 028 */
    -34, -38, /* 029 - 030 */
    -6, -9, -11, -14, -17, -20, -24, -28, -32, /* 031 - 039 */
    -6, -9, -11, -14, -17, -20, -24, -27, -32, /* 040 - 048 */
    -6, -9, -11, -14, -17, -20, -24, -27, -32, /* 049 - 057 */
    -7, -10, -12, -14, -17, -20, -23, -26, -30, /* 058 - 066 */
    -34, -38, /* 067 - 068 */
    -6, -9, -12, -15, -17, -20, -23, -26, -30, /* 069 - 077 */
    -34, -38, /* 078 - 079 */
    -6, -9, -12, -14, -17, -19, -23, -26, -30, /* 080 - 088 */
    -33, -38, /* 089 - 090 */
    -6, -9, -11, -14, -17, -20, -24, -28, -32, /* 091 - 099 */
    -6, -9, -11, -14, -17, -20, -24, -27, -32, /* 100 - 108 */
    -6, -10, -12, -14, -17, -20, -23, -26, -30, /* 109 - 117 */
    -34, -38, /* 118 - 119 */
    -6, -9, -12, -14, -17, -20, -23, -26, -30, /* 120 - 128 */
    -33, -38, /* 129 - 130 */
    -6, -8, -11, -13, -16, -19, -23, -26, -30, /* 131 - 139 */
    -34, /* 140 */
    -6, -9, -12, -14, -17, -20, -23, -26, -30, /* 141 - 149 */
    -33, -38, /* 150 - 151 */
    -6, -9, -11, -14, -16, -19, -23, -26, -29, /* 152 - 160 */
    -33, -37 /* 161 - 162 */
};
int values [163] = { /* values */
    2141859, -6024497, 1794353, -2796459, /* 000 - 003 */
    2543964, -1395010, 4532808, -8024142, /* 004 - 007 */
    5954283, /* 008 */
    1660858, -6522166, 3691054, -1073737, /* 009 - 012 */
    1839852, -1977613, 1369280, -6096699, /* 013 - 016 */
    1685387, -2630885, 1771363, /* 017 - 019 */
    2021935, -5955053, 1878866, -3736588, /* 020 - 023 */
    5172668, -5034701, 3363807, -1488898, /* 024 - 027 */
    4137518, -6504663, 4399947, /* 028 - 030 */
};

```

```

1659460, -4954925, 1526991, -2516840, /* 031 - 034 */
2433297, -1420192, 4915142, -9275283, /* 035 - 038 */
7345382, /* 039 */
1731412, -5380326, 1709947, -2884332, /* 040 - 043 */
2826155, -1658431, 5735419, -1076440, /* 044 - 047 */
8447731, /* 048 */
1811691, -5230411, 1584896, -2629686, /* 049 - 052 */
2573125, -1519547, 5306596, -1006748, /* 053 - 056 */
7984914, /* 057 */
9984187, 2018039, -3170439, 1589455, /* 058 - 061 */
-3484247, 4156025, -2951669, 1283619, /* 062 - 065 */
-3351854, 4821632, -2933104, /* 066 - 068 */
1593066, -2964337, 3217016, 9241350, /* 069 - 072 */
-3138800, 4137379, -3044785, 1346434, /* 073 - 076 */
-3552330, 5151235, -3156380, /* 077 - 079 */
2097414, -3125151, -4329302, 4317461, /* 080 - 083 */
-9923385, 1177737, -8280358, 3573841, /* 084 - 087 */
-9293869, 1335333, -8131189, /* 088 - 090 */
2062909, -4885969, 1427184, -2260352, /* 091 - 094 */
2111094, -1194770, 4024074, -7416131, /* 095 - 098 */
5754756, /* 099 */
2180664, -6296202, 1891070, -3064724, /* 100 - 103 */
2948811, -1724577, 5997964, -1138182, /* 104 - 107 */
9060635, /* 108 */
1603139, -8534362, -8667849, 4394307, /* 109 - 112 */
-8845054, 9830094, -6619258, 2766078, /* 113 - 116 */
-7009821, 9858768, -5896570, /* 117 - 119 */
2026128, -3624418, 1755093, 2385497, /* 120 - 123 */
-6688306, 8554336, -6280654, 2795381, /* 124 - 127 */
-7448619, 1092352, -6770721, /* 128 - 130 */
4264294, -2121222, 8360316, -1760020, /* 131 - 134 */
2193857, -1690439, 8124086, -2368735, /* 135 - 138 */
3831171, -2635711, /* 139 - 140 */
1871975, -2714091, -1640371, 3116431, /* 141 - 144 */
-7626180, 9286905, -6636238, 2903883, /* 145 - 148 */
-7657761, 1117466, -6928145, /* 149 - 151 */
1714980, -7089766, 3259217, -8410868, /* 152 - 155 */
1347052, -1391524, 9397261, -4114030, /* 156 - 159 */
1123482, -1737531, 1161350 /* 160 - 162 */
};
};
struct Table7 {
    int tbl_sca_sz = 16; /* tbl_sca_sz */
    int tbl_ele_sz = 16; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
 * Table 7
 * This table contains the geometric factors for each anode
 * [cm**2-sr].
 */
int tbl_var = 1; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    1, 1, 1, 1, 1, 1, /* 000 - 005 */
};

```

```

        1, 1, 1, 1, 1, 1,          /* 006 - 011 */
        1, 1, 1, 1,          /* 012 - 015 */
    };
    int offset [16] = {          /* offsets    */
        0, 1, 2, 3, 4, 5,      /* 000 - 005 */
        6, 7, 8, 9, 10, 11,   /* 006 - 011 */
        12, 13, 14, 15       /* 012 - 015 */
    };
    int scale [16] = {          /* scale factor*/
        -6, -6, -6, -6, -6, -6, /* 000 - 005 */
        -6, -6, -6, -6, -6, -6, /* 006 - 011 */
        -6, -6, -6, -6       /* 012 - 015 */
    };
    int values [16] = {        /* values     */
        588, 588, 588, 588, 588, 588, /* 000 - 005 */
        588, 588, 588, 588, 588, 588, /* 006 - 011 */
        588, 588, 588, 588     /* 012 - 015 */
    };
};
struct Table8 {
    int tbl_sca_sz = 1;        /* tbl_sca_sz */
    int tbl_ele_sz = 1;        /* tbl_ele_sz */
    int tbl_type = 0;         /* tbl_type   */
};
/*
 * Table 8
 * This table contains the MCP transparency factor as quoted by the
 * manufacture
 */
    int tbl_var = 1;          /* tbl_var     */
    int tbl_expand = 0;       /* tbl_expand  */
    int crit_act_sz = 0;     /* crit_act_sz */
    int format [16] = {      /* format      */
        1, 1, 1, 1, 1, 1,   /* 000 - 005 */
        1, 1, 1, 1, 1, 1,   /* 006 - 011 */
        1, 1, 1, 1         /* 012 - 015 */
    };
    int offset [16] = {      /* offsets     */
        0, 0, 0, 0, 0, 0,   /* 000 - 005 */
        0, 0, 0, 0, 0, 0,   /* 006 - 011 */
        0, 0, 0, 0         /* 012 - 015 */
    };
    int scale [1] = {-2};    /* scale factor */
    int values [1] = { 58 }; /* values      */
};
struct Table9 {
    int tbl_sca_sz = 1;        /* tbl_sca_sz */
    int tbl_ele_sz = 1;        /* tbl_ele_sz */
    int tbl_type = 0;         /* tbl_type   */
};
/*
 * Table 9
 * This table contains the Grid transparency factor as determined by the
 * manufactures specifications (see SwRI document ES-SGT-15-03561).
 */
    int tbl_var = 1;          /* tbl_var     */
    int tbl_expand = 0;       /* tbl_expand  */

```

```

    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1, /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0, /* 012 - 015 */
    };
    int scale [1] = {-2}; /* scale factor */
    int values [1] = { 81 }; /* values */
};
struct Table10 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 10
* This table contains the fraction of the ratio between the manufactured
* anode versus the theoretical anode.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1, /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0, /* 012 - 015 */
    };
    int scale [1] = {-2}; /* scale factor */
    int values [1] = { 87 }; /* values */
};
struct Table11 {
    int tbl_sca_sz = 16; /* tbl_sca_sz */
    int tbl_ele_sz = 16; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 11
* This table contains the amplitude adjustment factors for each anode.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1, /* 012 - 015 */
    };
};

```

```

int offset [16] = {                                /* offsets */
    0, 1, 2, 3, 4, 5,                               /* 000 - 005 */
    6, 7, 8, 9, 10, 11,                           /* 006 - 011 */
    12, 13, 14, 15                                /* 012 - 015 */
};
int scale [16] = {                                  /* scale factor*/
    -6, -6, -6, -6, -6, -6,                         /* 000 - 005 */
    -6, -6, -6, -6, -6, -6,                         /* 006 - 011 */
    -6, -6, -6, -6                                  /* 012 - 015 */
};
int values [16] = {                                 /* values */
    2632867, 1000000, 635386, 712443, 810931,        /* 000 - 004 */
    896400, 1370552, 928571, 665921, 1000000,      /* 005 - 009 */
    807453, 1000000, 988789, 1461922, 928571,      /* 010 - 014 */
    2146711                                         /* 015 */
};
};
struct Table12 {
    int tbl_sca_sz = 16;                            /* tbl_sca_sz */
    int tbl_ele_sz = 16;                            /* tbl_ele_sz */
    int tbl_type = 0;                               /* tbl_type */
};
/*
 * Table 12
 * This table contains the detector resolution as a fractional
 * number.
 */
int tbl_var = 1;                                    /* tbl_var */
int tbl_expand = 0;                                 /* tbl_expand */
int crit_act_sz = 0;                                /* crit_act_sz */
int format [16] = {                                 /* format */
    1, 1, 1, 1, 1, 1,                               /* 000 - 005 */
    1, 1, 1, 1, 1, 1,                               /* 006 - 011 */
    1, 1, 1, 1                                      /* 012 - 015 */
};
int offset [16] = {                                 /* offsets */
    0, 1, 2, 3, 4, 5,                               /* 000 - 005 */
    6, 7, 8, 9, 10, 11,                           /* 006 - 011 */
    12, 13, 14, 15                                /* 012 - 015 */
};
int scale [16] = {                                  /* scale factor*/
    -5, -5, -5, -5, -5, -5,                         /* 000 - 005 */
    -5, -5, -5, -5, -5, -5,                         /* 006 - 011 */
    -5, -5, -5, -5                                  /* 012 - 015 */
};
int values [16] = {                                 /* values */
    8653, 8394, 8331, 8579, 8124, 8480,             /* 000 - 005 */
    8194, 7890, 7812, 8094, 8095, 8346,           /* 006 - 011 */
    8297, 7353, 7396, 8843                         /* 012 - 015 */
};
};
struct Table13 {
    int tbl_sca_sz = 1;                            /* tbl_sca_sz */
    int tbl_ele_sz = 1;                            /* tbl_ele_sz */
    int tbl_type = 0;                               /* tbl_type */
};
/*

```

```

* Table 13
* This table contains the conversion factor from eV to erg [erg/eV]
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {-15}; /* scale factor */
    int values [1] = {1602}; /* values */
};
struct Table14 {
    int tbl_sca_sz = 1; /* tbl_sca_sz */
    int tbl_ele_sz = 1; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 14
* This factor contains the mass dependency in computing
* distribution (needed since we make computation using the
* particle energy and not velocity) and also the necessary
* scaling to put units in s**3/km***6
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {-7}; /* scale factor */
    int values [1] = {1616895}; /* values */
};
struct Table15 {
    int tbl_sca_sz = 34; /* tbl_sca_sz */
    int tbl_ele_sz = 34; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 15
* This table is mode dependent on the time summation mode which determines
* the number of spectra being accumulated into the summation of counts.
*/

```

```

int tbl_var = 0; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 17; /* crit_act_sz */
struct CriticalAction { /* crit act def */
    int status [16] = { /* status used */
        3, 3, 3, 3, 3, 3, 3, 3, /* 0000 - 0007 */
        3, 3, 3, 3, 3, 3, 3, 3 /* 0008 - 0015 */
    };
    int offset [16] = { /* act offsets */
        0, 0, 0, 0, 0, 0, 0, 0, /* 0000 - 0007 */
        0, 0, 0, 0, 0, 0, 0, 0 /* 0008 - 0015 */
    };
    int table [17] = { /* action table */
        0, 2, 4, 6, 8, 10, 12, 14, /* 0000 - 0007 */
        16, 18, 20, 22, 24, 26, 28, 30, /* 0008 - 0015 */
        32 /* 0016 */
    };
};
int format [16] = { /* format */
    2, 2, 2, 2, 2, 2, /* 000 - 005 */
    2, 2, 2, 2, 2, 2, /* 006 - 011 */
    2, 2, 2, 2 /* 012 - 015 */
};
int offset [16] = { /* offsets */
    -1, -1, -1, -1, -1, -1, /* 000 - 005 */
    -1, -1, -1, -1, -1, -1, /* 006 - 011 */
    -1, -1, -1, -1 /* 012 - 015 */
};
int scale [34] = { /* scale factor*/
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 000 - 009 */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 010 - 019 */
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 020 - 029 */
    0, 0, 0, 0 /* 030 - 033 */
};
int values [34] = { /* values */
    1, 0, 1, 0, 2, 0, 3, 0, 4, 0, /* 000 - 009 */
    5, 0, 6, 0, 7, 0, 8, 0, 9, 0, /* 010 - 019 */
    10, 0, 11, 0, 12, 0, 13, 0, 14, 0, /* 020 - 029 */
    15, 0, 16, 0 /* 030 - 033 */
};
};
struct Table16 {
    int tbl_sca_sz = 6; /* tbl_sca_sz */
    int tbl_ele_sz = 6; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 16
* This table is mode dependent on the step summation mode which determines
* the number of Energy steps being accumulated into the summation of counts.
*/
int tbl_var = 0; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 3; /* crit_act_sz */
struct CriticalAction { /* crit act def */
    int status [16] = { /* status used */

```

```

        4, 4, 4, 4, 4, 4, 4, 4, /* 0000 - 0007 */
        4, 4, 4, 4, 4, 4, 4, 4 /* 0008 - 0015 */
};
int offset [16] = { /* act offsets */
    0, 0, 0, 0, 0, 0, 0, 0, /* 0000 - 0007 */
    0, 0, 0, 0, 0, 0, 0, 0 /* 0008 - 0015 */
};
int table [3] = {0, 2, 4}; /* action table */
};
int format [16] = { /* format */
    2, 2, 2, 2, 2, 2, /* 000 - 005 */
    2, 2, 2, 2, 2, 2, /* 006 - 011 */
    2, 2, 2, 2 /* 012 - 015 */
};
int offset [16] = { /* offsets */
    -1, -1, -1, -1, -1, -1, /* 000 - 005 */
    -1, -1, -1, -1, -1, -1, /* 006 - 011 */
    -1, -1, -1, -1 /* 012 - 015 */
};
int scale [6] = { /* scale factor*/
    0, 0, 0, 0, 0, 0 /* 000 - 005 */
};
int values [6] = { /* values */
    1, 0, 2, 0, 4, 0 /* 000 - 005 */
};
};
struct Table17 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 17
* This table contains the sets of polynomial coefficients which
* place the value on 1 into the working data buffer.
*/
int tbl_var = 0; /* tbl_var */
int tbl_expand = 0; /* tbl_expand */
int crit_act_sz = 0; /* crit_act_sz */
int format [16] = { /* format */
    2, 2, 2, 2, 2, 2, /* 000 - 005 */
    2, 2, 2, 2, 2, 2, /* 006 - 011 */
    2, 2, 2, 2 /* 012 - 015 */
};
int offset [16] = { /* offsets */
    0, 0, 0, 0, 0, 0, /* 000 - 005 */
    0, 0, 0, 0, 0, 0, /* 006 - 011 */
    0, 0, 0, 0 /* 012 - 015 */
};
int scale [2] = {0, 0}; /* scale factor */
int values [2] = {1, 0}; /* values */
};
struct Table18 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};

```

```

/*
* Table 18
* This table is a function of calibration set 5, High Range Sweep
* Summation (MSB). This table is a polynomial to adjust the value to
* the upper bytes of a 4-byte integer.
*/
    int tbl_var = -6; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, 0}; /* scale factor */
    int values [2] = {0, 65536}; /* values */
};
struct Table19 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 19
* This table is a function of calibration set 6, High Range Sweep
* Summation (LSB). This table is a polynomial to adjust the value to
* the lower bytes of a 4-byte integer.
*/
    int tbl_var = -7; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [16] = { /* format */
        2, 2, 2, 2, 2, 2, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [2] = {0, 0}; /* scale factor */
    int values [2] = {0, 1}; /* values */
};
struct Table20 {
    int tbl_sca_sz = 2; /* tbl_sca_sz */
    int tbl_ele_sz = 2; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
};
/*
* Table 20
* This table is a function of calibration set 7, Total Range Sweep

```

```

* Summation (MSB). This table is a polynomial to adjust the value to
* the upper bytes of a 4-byte integer.
*/
    int tbl_var = -8;          /* tbl_var      */
    int tbl_expand = 0;       /* tbl_expand   */
    int crit_act_sz = 0;     /* crit_act_sz  */
    int format [16] = {      /* format       */
        2, 2, 2, 2, 2, 2,   /* 000 - 005   */
        2, 2, 2, 2, 2, 2,   /* 006 - 011   */
        2, 2, 2, 2         /* 012 - 015   */
    };
    int offset [16] = {      /* offsets      */
        0, 0, 0, 0, 0, 0,   /* 000 - 005   */
        0, 0, 0, 0, 0, 0,   /* 006 - 011   */
        0, 0, 0, 0         /* 012 - 015   */
    };
    int scale [2] = {0, 0};  /* scale factor */
    int values [2] = {0, 65536}; /* values      */
};
struct Table21 {
    int tbl_sca_sz = 2;     /* tbl_sca_sz   */
    int tbl_ele_sz = 2;     /* tbl_ele_sz   */
    int tbl_type = 0;      /* tbl_type     */
};
/*
* Table 21
* This table is a function of calibration set 8, Total Range Sweep
* Summation (LSB). This table is a polynomial to adjust the value to
* the lower bytes of a 4-byte integer.
*/
    int tbl_var = -9;          /* tbl_var      */
    int tbl_expand = 0;       /* tbl_expand   */
    int crit_act_sz = 0;     /* crit_act_sz  */
    int format [16] = {      /* format       */
        2, 2, 2, 2, 2, 2,   /* 000 - 005   */
        2, 2, 2, 2, 2, 2,   /* 006 - 011   */
        2, 2, 2, 2         /* 012 - 015   */
    };
    int offset [16] = {      /* offsets      */
        0, 0, 0, 0, 0, 0,   /* 000 - 005   */
        0, 0, 0, 0, 0, 0,   /* 006 - 011   */
        0, 0, 0, 0         /* 012 - 015   */
    };
    int scale [2] = {0, 0};  /* scale factor */
    int values [2] = {0, 1}; /* values      */
};
struct Table22 {
    int tbl_sca_sz = 2;     /* tbl_sca_sz   */
    int tbl_ele_sz = 2;     /* tbl_ele_sz   */
    int tbl_type = 0;      /* tbl_type     */
};
/*
* Table 22
* This table is a function of calibration set 3, Total Range Anode
* Summation (MSB). This table is a polynomial to adjust the value to
* the upper bytes of a 4-byte integer.
*/

```

```

    int tbl_var = -4;                /* tbl_var      */
    int tbl_expand = 0;             /* tbl_expand   */
    int crit_act_sz = 0;           /* crit_act_sz  */
    int format [16] = {            /* format       */
        2, 2, 2, 2, 2, 2,         /* 000 - 005   */
        2, 2, 2, 2, 2, 2,         /* 006 - 011   */
        2, 2, 2, 2                /* 012 - 015   */
    };
    int offset [16] = {            /* offsets      */
        0, 0, 0, 0, 0, 0,         /* 000 - 005   */
        0, 0, 0, 0, 0, 0,         /* 006 - 011   */
        0, 0, 0, 0                /* 012 - 015   */
    };
    int scale [2] = {0, 0};        /* scale factor */
    int values [2] = {0, 65536};   /* values       */
};
struct Table23 {
    int tbl_sca_sz = 2;           /* tbl_sca_sz   */
    int tbl_ele_sz = 2;           /* tbl_ele_sz   */
    int tbl_type = 0;            /* tbl_type     */
};
/*
 * Table 23
 * This table is a function of calibration set 4, Total Range Anode
 * Summation (LSB). This table is a polynomial to adjust the value to
 * the lower bytes of a 4-byte integer.
 */
    int tbl_var = -5;                /* tbl_var      */
    int tbl_expand = 0;             /* tbl_expand   */
    int crit_act_sz = 0;           /* crit_act_sz  */
    int format [16] = {            /* format       */
        2, 2, 2, 2, 2, 2,         /* 000 - 005   */
        2, 2, 2, 2, 2, 2,         /* 006 - 011   */
        2, 2, 2, 2                /* 012 - 015   */
    };
    int offset [16] = {            /* offsets      */
        0, 0, 0, 0, 0, 0,         /* 000 - 005   */
        0, 0, 0, 0, 0, 0,         /* 006 - 011   */
        0, 0, 0, 0                /* 012 - 015   */
    };
    int scale [2] = {0, 0};        /* scale factor */
    int values [2] = {0, 1};      /* values       */
};
struct Table24 {
    int tbl_sca_sz = 2;           /* tbl_sca_sz   */
    int tbl_ele_sz = 2;           /* tbl_ele_sz   */
    int tbl_type = 0;            /* tbl_type     */
};
/*
 * Table 24
 * This table contains the sets of polynomial coefficients which
 * converts calibration set 0 telemetry to voltage.
 */
    int tbl_var = -1;                /* tbl_var      */
    int tbl_expand = 0;             /* tbl_expand   */
    int crit_act_sz = 0;           /* crit_act_sz  */
    int format [16] = {            /* format       */

```

```

        2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2,
        2, 2, 2, 2
    };
    int offset [16] = {
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [2] = {0, -8};
    int values [2] = {0, 1960784};
};
struct Table25 {
    int tbl_sca_sz = 2;
    int tbl_ele_sz = 2;
    int tbl_type = 0;
};
/*
 * Table 25
 * This table contains the sets of polynomial coefficients which
 * converts calibration set 0 telemetry to microamp.
 */
    int tbl_var = -1;
    int tbl_expand = 0;
    int crit_act_sz = 0;
    int format [16] = {
        2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2,
        2, 2, 2, 2
    };
    int offset [16] = {
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [2] = {0, -6};
    int values [2] = {0, 1620483};
};
struct Table26 {
    int tbl_sca_sz = 2;
    int tbl_ele_sz = 2;
    int tbl_type = 0;
};
/*
 * Table 26
 * This table contains the sets of polynomial coefficients which
 * converts calibration set 0 telemetry to degrees C.
 */
    int tbl_var = -1;
    int tbl_expand = 0;
    int crit_act_sz = 0;
    int format [16] = {
        2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2,
        2, 2, 2, 2
    };
    int offset [16] = {

```

```

        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [2] = {-1, -6};
    int values [2] = {-2732, 1620483};
};
struct Table27 {
    int tbl_sca_sz = 1;
    int tbl_ele_sz = 1;
    int tbl_type = 0;
};
/*
* Table 27
* This table contains the conversion factor from eV to Joule [Joule/eV]
*/
    int tbl_var = 2;
    int tbl_expand = 0;
    int crit_act_sz = 0;
    int format [16] = {
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1
    };
    int offset [16] = {
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [1] = {-22};
    int values [1] = {1602};
};
struct Table28 {
    int tbl_sca_sz = 1;
    int tbl_ele_sz = 1;
    int tbl_type = 0;
};
/*
* Table 28
* This table contains the constant number 2 for conversion used in energy
* to velocity.
*/
    int tbl_var = 2;
    int tbl_expand = 0;
    int crit_act_sz = 0;
    int format [16] = {
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1
    };
    int offset [16] = {
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0
    };
    int scale [1] = {0};
    int values [1] = {2};
};

```

```

};
struct Table29 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 29
* This table contains the electron mass in [kg].
*/
    int tbl_var = 2;           /* tbl_var */
    int tbl_expand = 0;        /* tbl_expand */
    int crit_act_sz = 0;       /* crit_act_ele */
    int format [16] = {        /* format */
        1, 1, 1, 1, 1, 1,     /* 000 - 005 */
        1, 1, 1, 1, 1, 1,     /* 006 - 011 */
        1, 1, 1, 1           /* 012 - 015 */
    };
    int offset [16] = {        /* offsets */
        0, 0, 0, 0, 0, 0,     /* 000 - 005 */
        0, 0, 0, 0, 0, 0,     /* 006 - 011 */
        0, 0, 0, 0           /* 012 - 015 */
    };
    int scale [1] = {-33};     /* scale factor */
    int values [1] = {911};    /* values */
};
struct Table30 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*
* Table 30
* This table contains the constant number 1000 for converting from m to km.
* This Table contains the number of m in a km [m/km]
*/
    int tbl_var = 1;           /* tbl_var */
    int tbl_expand = 0;        /* tbl_expand */
    int crit_act_sz = 0;       /* crit_act_ele */
    int format [16] = {        /* format */
        1, 1, 1, 1, 1, 1,     /* 000 - 005 */
        1, 1, 1, 1, 1, 1,     /* 006 - 011 */
        1, 1, 1, 1           /* 012 - 015 */
    };
    int offset [16] = {        /* offsets */
        0, 0, 0, 0, 0, 0,     /* 000 - 005 */
        0, 0, 0, 0, 0, 0,     /* 006 - 011 */
        0, 0, 0, 0           /* 012 - 015 */
    };
    int scale [1] = {3};       /* scale factor */
    int values [1] = {1};     /* values */
};
struct Table31 {
    int tbl_sca_sz = 1;           /* tbl_sca_sz */
    int tbl_ele_sz = 1;         /* tbl_ele_sz */
    int tbl_type = 0;           /* tbl_type */
/*

```

```

* Table 31
* This table contains the constant number 1000 for converting from m/s
* to km/s. This Table contains the number of m/s in a km/s [m/km]
*/
    int tbl_var = 2; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_ele */
    int format [16] = { /* format */
        1, 1, 1, 1, 1, 1, /* 000 - 005 */
        1, 1, 1, 1, 1, 1, /* 006 - 011 */
        1, 1, 1, 1 /* 012 - 015 */
    };
    int offset [16] = { /* offsets */
        0, 0, 0, 0, 0, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0 /* 012 - 015 */
    };
    int scale [1] = {3}; /* scale factor */
    int values [1] = {1}; /* values */
};
struct Table32 {
    int tbl_sca_sz = 0; /* tbl_sca_sz */
    int tbl_ele_sz = 9; /* tbl_ele_sz */
    int tbl_type = 1; /* tbl_type */
}
/*
* Table 32
* This table is an ASCII look-up table which indicates what
* the settings of the bit status monitors represents.
*/
    int tbl_var = 4; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [23] = { /* format */
        -1, -1, 0, -1, -1, 0, /* 000 - 005 */
        0, 0, 0, 0, 0, 0, /* 006 - 011 */
        0, 0, 0, 0, 0, 0, /* 012 - 017 */
        0, 0, 0, 0, -1 /* 018 - 022 */
    };
    int offset [23] = { /* offsets */
        -1, -1, 4, -1, -1, 0, /* 000 - 005 */
        2, 2, 2, 2, 2, 2, /* 006 - 011 */
        2, 2, 2, 2, 2, 2, /* 012 - 017 */
        2, 2, 2, 2, -1 /* 018 - 022 */
    };
    string values [9] = { /* values */
        "Off", "On", "Disabled", /* 000 - 002 */
        "Enabled", "Undefined", "Booting", /* 003 - 005 */
        "Safe", "Prom", "Normal" /* 006 - 008 */
    };
};
struct Table33 {
    int tbl_sca_sz = 20; /* tbl_sca_sz */
    int tbl_ele_sz = 20; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*

```

```

* Table 33
* This table is a look-up table which indicates the normalization factor
* for the science data. It applies to Time and Step summations.
*/
    int tbl_var = 4; /* tbl_var */
    int tbl_expand = 1; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [23] = { /* format */
        -1, -1, -1, 0, 0, -1, /* 000 - 005 */
        -1, -1, -1, -1, -1, -1, /* 006 - 011 */
        -1, -1, -1, -1, -1, -1, /* 012 - 017 */
        -1, -1, -1, -1, -1 /* 018 - 022 */
    };
    int offset [23] = { /* offsets */
        -1, -1, -1, 3, 0, -1, /* 000 - 005 */
        -1, -1, -1, -1, -1, -1, /* 006 - 011 */
        -1, -1, -1, -1, -1, -1, /* 012 - 017 */
        -1, -1, -1, -1, -1 /* 018 - 022 */
    };
    int scale [20] = { /* scale factor */
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, /* 000 - 009 */
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0 /* 010 - 019 */
    };
    int values [20] = { /* values */
        1, 2, 4, 1, 1, 2, 3, 4, 5, 6, /* 000 - 009 */
        7, 8, 9, 10, 11, 12, 13, 14, 15, 16 /* 010 - 019 */
    };
};
struct Constant0 {
    int id = 1;
/*
* Elevation angle as measured from the spacecraft +C axis
*/
    int scale[16] = { -2, -2, -2, -2, -2, -2, -2, -2,
                    -2, -2, -2, -2, -2, -2, -2, -2 };
    int values[16] = { -16875, -14625, -12375, -10125,
                      -7875, -5625, -3375, -1125,
                      1125, 3375, 5625, 7875,
                      10125, 12375, 14625, 16875 };
};
struct Constant1 {
    int id = 2;
/*
* Azimuthal angle offsets measured from the spacecraft +A axis
* in the direction of the spacecraft +B axis
*/
    int scale[16] = { 0, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0 };
    int values[16] = { 270, 270, 270, 270, 270, 270, 270, 270,
                      90, 90, 90, 90, 90, 90, 90, 90 };
};
struct Constant2 {
    int id = 3;
/*
* Azimuthal field of view in the frame of the spacecraft (spacecraft

```

```

*      A-B plane)
*/
    int scale[16] = {  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,
                      -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1 };
    int values[16] = {  50,  50,  50,  50,  50,  50,  50,  50,
                      50,  50,  50,  50,  50,  50,  50,  50 };
};
struct Constant3 {
    int id = 4;
/*
*      Initial Aperture Elevation Angle as measured from the spacecraft
*      +C axis
*/
    int scale[16] = {  -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2,
                      -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2 };
    int values[16] = { 15925, 13675, 11425,  9175,
                      6925,  4675,  2425,  175,
                      175,  2425,  4675,  6925,
                      9175, 11425, 13675, 15925 };
};
struct Constant4 {
    int id = 5;
/*
*      Final Aperture Elevation Angle as measured from the spacecraft
*      +C axis
*/
    int scale[16] = {  -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2,
                      -2,  -2,  -2,  -2,  -2,  -2,  -2,  -2 };
    int values[16] = { 17825, 15575, 13325, 11075,
                      8825,  6575,  4325,  2075,
                      2075,  4325,  6575,  8825,
                      11075, 13325, 15575, 17825 };
};
struct Constant5 {
    int id = 6;
/*
*      Component of the aperture normal vector with respect to the +A axis
*      of the magnetometer.
*/
    int scale[16] = {  0,  0,  0,  0,  0,  0,  0,  0,
                      0,  0,  0,  0,  0,  0,  0,  0 };
    int values[16] = {  0,  0,  0,  0,
                      0,  0,  0,  0,
                      0,  0,  0,  0,
                      0,  0,  0,  0 };
};
struct Constant6 {
    int id = 7;
/*
*      Component of the aperture normal vector with respect to the +B axis
*      of the magnetometer.
*/
    int scale[16] = {  -6,  -6,  -6,  -6,  -6,  -6,  -6,  -6,
                      -6,  -6,  -6,  -6,  -6,  -6,  -6,  -6 };
    int values[16] = { -195090, -555570, -831470, -980785,

```

```
        -980785, -831470, -555570, -195090,  
        195090, 555570, 831470, 980785,  
        980785, 831470, 555570, 195090 };  
};  
struct Constant7 {  
    int id = 8;  
/*  
 *   Component of the aperture normal vector with respect to the +C axis  
 *   of the magnetometer.  
 */  
    int scale[16] = { -6, -6, -6, -6, -6, -6, -6, -6,  
                     -6, -6, -6, -6, -6, -6, -6, -6 };  
    int values[16] = { -980785, -831470, -555570, -195090,  
                      195090, 555570, 831470, 980785,  
                      980785, 831470, 555570, 195090,  
                      -195090, -555570, -831470, -980785 };  
};  
}
```

Appendix B - ELS Instrument Operation Scenarios

Science array is created every 4 sec

Word length is 2 bytes long-----

There are 16 sectors-----

DPU defines 128 energy steps-----

So every 4 sec, you generate a matrix of 128 x 16 x 2 = 4096 bytes

The science array is compressed into a science matrix.

This science matrix can be further compressed by:

- 1) a sector mask to eliminate sector data from being transmitted.
 There are 16 bits and 16 sectors, one bit for each sector.
 Bits are set to 0 to block sector data, 1 to transmit sector data;
 thus, value can be 0-65535 decimal
- 2) bit compression to compress the 2 byte word lengths (16 bits each word) to 1 byte word lengths (8 bits) before transmission.
 Values are 0 decimal for full resolution, 16 bit words
 1 decimal for compressed words of 8 bits
- 3) time sum where you add together science matrices taken at different times before transmission.
 Values are 0 decimal meaning 1 sweep or 4 sec
 1 decimal meaning 2 sweeps or 8 sec
 2 decimal meaning 4 sweeps or 16 sec
 3 decimal meaning 8 sweeps or 32 sec
 4 decimal meaning 16 sweeps or 64 sec
- 4) energy sum where you add together energy steps before transmission.
 Values are 0 decimal meaning 1 step sum - 128 step sweep
 1 decimal meaning 2 step sum (0+1, 2+3, 4+5, 6+7, 8+9, etc.) - 64 step sweep
 2 decimal meaning 4 step sum (0+1+2+3, 4+5+6+7, 8+9+10+11, etc.) - 32 step sweep

The following is a table of possible science Matrices:

| Sector Mask | Bit Comp. | Time Sum | Energy Sum | Science Matrix Size Bytes | Accum | Effective Rate |
|-------------|-----------|----------|------------|---------------------------|--------|----------------|
| 16 bits | 0 | 0 | 0 | 128 x 16 x 2 = 4096 | 4 sec | 8192 bps |
| 16 bits | 0 | 0 | 1 | 64 X 16 X 2 = 2048 | 4 sec | 4096 bps |
| 16 bits | 0 | 0 | 2 | 32 x 16 x 2 = 1024 | 4 sec | 2048 bps |
| 16 bits | 0 | 1 | 0 | 128 x 16 x 2 = 4096 | 8 sec | 4096 bps |
| 16 bits | 0 | 1 | 1 | 64 x 16 x 2 = 2048 | 8 sec | 2048 bps |
| 16 bits | 0 | 1 | 2 | 32 x 16 x 2 = 1024 | 8 sec | 1024 bps |
| 16 bits | 0 | 2 | 0 | 128 x 16 x 2 = 4096 | 16 sec | 2048 bps |
| 16 bits | 0 | 2 | 1 | 64 x 16 x 2 = 2048 | 16 sec | 1024 bps |
| 16 bits | 0 | 2 | 2 | 32 x 16 x 2 = 1024 | 16 sec | 512 bps |
| 16 bits | 0 | 3 | 0 | 128 x 16 x 2 = 4096 | 32 sec | 1024 bps |
| 16 bits | 0 | 3 | 1 | 64 x 16 x 2 = 2048 | 32 sec | 512 bps |
| 16 bits | 0 | 3 | 2 | 32 x 16 x 2 = 1024 | 32 sec | 256 bps |
| 16 bits | 0 | 4 | 0 | 128 x 16 x 2 = 4096 | 64 sec | 512 bps |
| 16 bits | 0 | 4 | 1 | 64 x 16 x 2 = 2048 | 64 sec | 256 bps |
| 16 bits | 0 | 4 | 2 | 32 x 16 x 2 = 1024 | 64 sec | 128 bps |

| | | | | | | |
|---------|---|---|---|---------------------|--------|----------|
| 16 bits | 1 | 0 | 0 | 128 x 16 x 1 = 2048 | 4 sec | 4096 bps |
| 16 bits | 1 | 0 | 1 | 64 x 16 x 1 = 1024 | 4 sec | 2048 bps |
| 16 bits | 1 | 0 | 2 | 32 x 16 x 1 = 512 | 4 sec | 1024 bps |
| 16 bits | 1 | 1 | 0 | 128 x 16 x 1 = 2048 | 8 sec | 2048 bps |
| 16 bits | 1 | 1 | 1 | 64 x 16 x 1 = 1024 | 8 sec | 1024 bps |
| 16 bits | 1 | 1 | 2 | 32 x 16 x 1 = 512 | 8 sec | 512 bps |
| 16 bits | 1 | 2 | 0 | 128 x 16 x 1 = 2048 | 16 sec | 1024 bps |
| 16 bits | 1 | 2 | 1 | 64 x 16 x 1 = 1024 | 16 sec | 512 bps |
| 16 bits | 1 | 2 | 2 | 32 x 16 x 1 = 512 | 16 sec | 256 bps |
| 16 bits | 1 | 3 | 0 | 128 x 16 x 1 = 2048 | 32 sec | 512 bps |
| 16 bits | 1 | 3 | 1 | 64 x 16 x 1 = 1024 | 32 sec | 256 bps |
| 16 bits | 1 | 3 | 2 | 32 x 16 x 1 = 512 | 32 sec | 128 bps |
| 16 bits | 1 | 4 | 0 | 128 x 16 x 1 = 2048 | 64 sec | 256 bps |
| 16 bits | 1 | 4 | 1 | 64 x 16 x 1 = 1024 | 64 sec | 128 bps |
| 16 bits | 1 | 4 | 2 | 32 x 16 x 1 = 512 | 64 sec | 64 bps |
| 15 bits | 0 | 0 | 0 | 128 x 15 x 2 = 3840 | 4 sec | 7680 bps |
| 15 bits | 0 | 0 | 1 | 64 x 15 x 2 = 1920 | 4 sec | 3840 bps |
| 15 bits | 0 | 0 | 2 | 32 x 15 x 2 = 960 | 4 sec | 1920 bps |
| 15 bits | 0 | 1 | 0 | 128 x 15 x 2 = 3840 | 8 sec | 3840 bps |
| 15 bits | 0 | 1 | 1 | 64 x 15 x 2 = 1920 | 8 sec | 1920 bps |
| 15 bits | 0 | 1 | 2 | 32 x 15 x 2 = 960 | 8 sec | 960 bps |
| 15 bits | 0 | 2 | 0 | 128 x 15 x 2 = 3840 | 16 sec | 1920 bps |
| 15 bits | 0 | 2 | 1 | 64 x 15 x 2 = 1920 | 16 sec | 960 bps |
| 15 bits | 0 | 2 | 2 | 32 x 15 x 2 = 960 | 16 sec | 480 bps |
| 15 bits | 0 | 3 | 0 | 128 x 15 x 2 = 3840 | 32 sec | 960 bps |
| 15 bits | 0 | 3 | 1 | 64 x 15 x 2 = 1920 | 32 sec | 480 bps |
| 15 bits | 0 | 3 | 2 | 32 x 15 x 2 = 960 | 32 sec | 240 bps |
| 15 bits | 0 | 4 | 0 | 128 x 15 x 2 = 3840 | 64 sec | 480 bps |
| 15 bits | 0 | 4 | 1 | 64 x 15 x 2 = 1920 | 64 sec | 240 bps |
| 15 bits | 0 | 4 | 2 | 32 x 15 x 2 = 960 | 64 sec | 120 bps |
| 15 bits | 1 | 0 | 0 | 128 x 15 x 1 = 1920 | 4 sec | 3840 bps |
| 15 bits | 1 | 0 | 1 | 64 x 15 x 1 = 960 | 4 sec | 1920 bps |
| 15 bits | 1 | 0 | 2 | 32 x 15 x 1 = 480 | 4 sec | 960 bps |
| 15 bits | 1 | 1 | 0 | 128 x 15 x 1 = 1920 | 8 sec | 1920 bps |
| 15 bits | 1 | 1 | 1 | 64 x 15 x 1 = 960 | 8 sec | 960 bps |
| 15 bits | 1 | 1 | 2 | 32 x 15 x 1 = 480 | 8 sec | 480 bps |
| 15 bits | 1 | 2 | 0 | 128 x 15 x 1 = 1920 | 16 sec | 960 bps |
| 15 bits | 1 | 2 | 1 | 64 x 15 x 1 = 960 | 16 sec | 480 bps |
| 15 bits | 1 | 2 | 2 | 32 x 15 x 1 = 480 | 16 sec | 240 bps |
| 15 bits | 1 | 3 | 0 | 128 x 15 x 1 = 1920 | 32 sec | 480 bps |
| 15 bits | 1 | 3 | 1 | 64 x 15 x 1 = 960 | 32 sec | 240 bps |
| 15 bits | 1 | 3 | 2 | 32 x 15 x 1 = 480 | 32 sec | 120 bps |
| 15 bits | 1 | 4 | 0 | 128 x 15 x 1 = 1920 | 64 sec | 240 bps |
| 15 bits | 1 | 4 | 1 | 64 x 15 x 1 = 960 | 64 sec | 120 bps |
| 15 bits | 1 | 4 | 2 | 32 x 15 x 1 = 480 | 64 sec | 60 bps |
| 14 bits | 0 | 0 | 0 | 128 x 14 x 2 = 3584 | 4 sec | 7168 bps |
| 14 bits | 0 | 0 | 1 | 64 x 14 x 2 = 1792 | 4 sec | 3584 bps |
| 14 bits | 0 | 0 | 2 | 32 x 14 x 2 = 896 | 4 sec | 1792 bps |
| 14 bits | 0 | 1 | 0 | 128 x 14 x 2 = 3584 | 8 sec | 3584 bps |
| 14 bits | 0 | 1 | 1 | 64 x 14 x 2 = 1792 | 8 sec | 1792 bps |
| 14 bits | 0 | 1 | 2 | 32 x 14 x 2 = 896 | 8 sec | 896 bps |
| 14 bits | 0 | 2 | 0 | 128 x 14 x 2 = 3584 | 16 sec | 1792 bps |
| 14 bits | 0 | 2 | 1 | 64 x 14 x 2 = 1792 | 16 sec | 896 bps |
| 14 bits | 0 | 2 | 2 | 32 x 14 x 2 = 896 | 16 sec | 448 bps |
| 14 bits | 0 | 3 | 0 | 128 x 14 x 2 = 3584 | 32 sec | 896 bps |

| | | | | | | | |
|---------|---|---|---|----------------|------|--------|----------|
| 14 bits | 0 | 3 | 1 | 64 x 14 x 2 = | 1792 | 32 sec | 448 bps |
| 14 bits | 0 | 3 | 2 | 32 x 14 x 2 = | 896 | 32 sec | 224 bps |
| 14 bits | 0 | 4 | 0 | 128 x 14 x 2 = | 3584 | 64 sec | 448 bps |
| 14 bits | 0 | 4 | 1 | 64 x 14 x 2 = | 1792 | 64 sec | 224 bps |
| 14 bits | 0 | 4 | 2 | 32 x 14 x 2 = | 896 | 64 sec | 112 bps |
| 14 bits | 1 | 0 | 0 | 128 x 14 x 1 = | 1792 | 4 sec | 3584 bps |
| 14 bits | 1 | 0 | 1 | 64 x 14 x 1 = | 896 | 4 sec | 1792 bps |
| 14 bits | 1 | 0 | 2 | 32 x 14 x 1 = | 448 | 4 sec | 896 bps |
| 14 bits | 1 | 1 | 0 | 128 x 14 x 1 = | 1792 | 8 sec | 1792 bps |
| 14 bits | 1 | 1 | 1 | 64 x 14 x 1 = | 896 | 8 sec | 896 bps |
| 14 bits | 1 | 1 | 2 | 32 x 14 x 1 = | 448 | 8 sec | 448 bps |
| 14 bits | 1 | 2 | 0 | 128 x 14 x 1 = | 1792 | 16 sec | 896 bps |
| 14 bits | 1 | 2 | 1 | 64 x 14 x 1 = | 896 | 16 sec | 448 bps |
| 14 bits | 1 | 2 | 2 | 32 x 14 x 1 = | 448 | 16 sec | 224 bps |
| 14 bits | 1 | 3 | 0 | 128 x 14 x 1 = | 1792 | 32 sec | 448 bps |
| 14 bits | 1 | 3 | 1 | 64 x 14 x 1 = | 896 | 32 sec | 224 bps |
| 14 bits | 1 | 3 | 2 | 32 x 14 x 1 = | 448 | 32 sec | 112 bps |
| 14 bits | 1 | 4 | 0 | 128 x 14 x 1 = | 1792 | 64 sec | 224 bps |
| 14 bits | 1 | 4 | 1 | 64 x 14 x 1 = | 896 | 64 sec | 112 bps |
| 14 bits | 1 | 4 | 2 | 32 x 14 x 1 = | 448 | 64 sec | 56 bps |
| 13 bits | 0 | 0 | 0 | 128 x 13 x 2 = | 3328 | 4 sec | 6656 bps |
| 13 bits | 0 | 0 | 1 | 64 x 13 x 2 = | 1664 | 4 sec | 3328 bps |
| 13 bits | 0 | 0 | 2 | 32 x 13 x 2 = | 832 | 4 sec | 1664 bps |
| 13 bits | 0 | 1 | 0 | 128 x 13 x 2 = | 3328 | 8 sec | 3328 bps |
| 13 bits | 0 | 1 | 1 | 64 x 13 x 2 = | 1664 | 8 sec | 1664 bps |
| 13 bits | 0 | 1 | 2 | 32 x 13 x 2 = | 832 | 8 sec | 832 bps |
| 13 bits | 0 | 2 | 0 | 128 x 13 x 2 = | 3328 | 16 sec | 1664 bps |
| 13 bits | 0 | 2 | 1 | 64 x 13 x 2 = | 1664 | 16 sec | 832 bps |
| 13 bits | 0 | 2 | 2 | 32 x 13 x 2 = | 832 | 16 sec | 416 bps |
| 13 bits | 0 | 3 | 0 | 128 x 13 x 2 = | 3328 | 32 sec | 832 bps |
| 13 bits | 0 | 3 | 1 | 64 x 13 x 2 = | 1664 | 32 sec | 416 bps |
| 13 bits | 0 | 3 | 2 | 32 x 13 x 2 = | 832 | 32 sec | 208 bps |
| 13 bits | 0 | 4 | 0 | 128 x 13 x 2 = | 3328 | 64 sec | 416 bps |
| 13 bits | 0 | 4 | 1 | 64 x 13 x 2 = | 1664 | 64 sec | 208 bps |
| 13 bits | 0 | 4 | 2 | 32 x 13 x 2 = | 832 | 64 sec | 104 bps |
| 13 bits | 1 | 0 | 0 | 128 x 13 x 1 = | 1664 | 4 sec | 3328 bps |
| 13 bits | 1 | 0 | 1 | 64 x 13 x 1 = | 832 | 4 sec | 1664 bps |
| 13 bits | 1 | 0 | 2 | 32 x 13 x 1 = | 416 | 4 sec | 832 bps |
| 13 bits | 1 | 1 | 0 | 128 x 13 x 1 = | 1664 | 8 sec | 1664 bps |
| 13 bits | 1 | 1 | 1 | 64 x 13 x 1 = | 832 | 8 sec | 832 bps |
| 13 bits | 1 | 1 | 2 | 32 x 13 x 1 = | 416 | 8 sec | 416 bps |
| 13 bits | 1 | 2 | 0 | 128 x 13 x 1 = | 1664 | 16 sec | 832 bps |
| 13 bits | 1 | 2 | 1 | 64 x 13 x 1 = | 832 | 16 sec | 416 bps |
| 13 bits | 1 | 2 | 2 | 32 x 13 x 1 = | 416 | 16 sec | 208 bps |
| 13 bits | 1 | 3 | 0 | 128 x 13 x 1 = | 1664 | 32 sec | 416 bps |
| 13 bits | 1 | 3 | 1 | 64 x 13 x 1 = | 832 | 32 sec | 208 bps |
| 13 bits | 1 | 3 | 2 | 32 x 13 x 1 = | 416 | 32 sec | 104 bps |
| 13 bits | 1 | 4 | 0 | 128 x 13 x 1 = | 1664 | 64 sec | 208 bps |
| 13 bits | 1 | 4 | 1 | 64 x 13 x 1 = | 832 | 64 sec | 104 bps |
| 13 bits | 1 | 4 | 2 | 32 x 13 x 1 = | 416 | 64 sec | 52 bps |
| 12 bits | 0 | 0 | 0 | 128 x 12 x 2 = | 3072 | 4 sec | 6144 bps |
| 12 bits | 0 | 0 | 1 | 64 x 12 x 2 = | 1536 | 4 sec | 3072 bps |
| 12 bits | 0 | 0 | 2 | 32 x 12 x 2 = | 768 | 4 sec | 1536 bps |
| 12 bits | 0 | 1 | 0 | 128 x 12 x 2 = | 3072 | 8 sec | 3072 bps |
| 12 bits | 0 | 1 | 1 | 64 x 12 x 2 = | 1536 | 8 sec | 1536 bps |

| | | | | | | | |
|---------|---|---|---|----------------|------|--------|----------|
| 12 bits | 0 | 1 | 2 | 32 x 12 x 2 = | 768 | 8 sec | 768 bps |
| 12 bits | 0 | 2 | 0 | 128 x 12 x 2 = | 3072 | 16 sec | 1536 bps |
| 12 bits | 0 | 2 | 1 | 64 x 12 x 2 = | 1536 | 16 sec | 768 bps |
| 12 bits | 0 | 2 | 2 | 32 x 12 x 2 = | 768 | 16 sec | 384 bps |
| 12 bits | 0 | 3 | 0 | 128 x 12 x 2 = | 3072 | 32 sec | 768 bps |
| 12 bits | 0 | 3 | 1 | 64 x 12 x 2 = | 1536 | 32 sec | 384 bps |
| 12 bits | 0 | 3 | 2 | 32 x 12 x 2 = | 768 | 32 sec | 192 bps |
| 12 bits | 0 | 4 | 0 | 128 x 12 x 2 = | 3072 | 64 sec | 384 bps |
| 12 bits | 0 | 4 | 1 | 64 x 12 x 2 = | 1536 | 64 sec | 192 bps |
| 12 bits | 0 | 4 | 2 | 32 x 12 x 2 = | 768 | 64 sec | 96 bps |
| 12 bits | 1 | 0 | 0 | 128 x 12 x 1 = | 1536 | 4 sec | 3072 bps |
| 12 bits | 1 | 0 | 1 | 64 x 12 x 1 = | 768 | 4 sec | 1536 bps |
| 12 bits | 1 | 0 | 2 | 32 x 12 x 1 = | 384 | 4 sec | 768 bps |
| 12 bits | 1 | 1 | 0 | 128 x 12 x 1 = | 1536 | 8 sec | 1536 bps |
| 12 bits | 1 | 1 | 1 | 64 x 12 x 1 = | 768 | 8 sec | 768 bps |
| 12 bits | 1 | 1 | 2 | 32 x 12 x 1 = | 384 | 8 sec | 384 bps |
| 12 bits | 1 | 2 | 0 | 128 x 12 x 1 = | 1536 | 16 sec | 768 bps |
| 12 bits | 1 | 2 | 1 | 64 x 12 x 1 = | 768 | 16 sec | 384 bps |
| 12 bits | 1 | 2 | 2 | 32 x 12 x 1 = | 384 | 16 sec | 192 bps |
| 12 bits | 1 | 3 | 0 | 128 x 12 x 1 = | 1536 | 32 sec | 384 bps |
| 12 bits | 1 | 3 | 1 | 64 x 12 x 1 = | 768 | 32 sec | 192 bps |
| 12 bits | 1 | 3 | 2 | 32 x 12 x 1 = | 384 | 32 sec | 96 bps |
| 12 bits | 1 | 4 | 0 | 128 x 12 x 1 = | 1536 | 64 sec | 192 bps |
| 12 bits | 1 | 4 | 1 | 64 x 12 x 1 = | 768 | 64 sec | 96 bps |
| 12 bits | 1 | 4 | 2 | 32 x 12 x 1 = | 385 | 64 sec | 48 bps |
| 11 bits | 0 | 0 | 0 | 128 x 11 x 2 = | 2816 | 4 sec | 5632 bps |
| 11 bits | 0 | 0 | 1 | 64 x 11 x 2 = | 1408 | 4 sec | 2816 bps |
| 11 bits | 0 | 0 | 2 | 32 x 11 x 2 = | 704 | 4 sec | 1408 bps |
| 11 bits | 0 | 1 | 0 | 128 x 11 x 2 = | 2816 | 8 sec | 2816 bps |
| 11 bits | 0 | 1 | 1 | 64 x 11 x 2 = | 1408 | 8 sec | 1408 bps |
| 11 bits | 0 | 1 | 2 | 32 x 11 x 2 = | 704 | 8 sec | 704 bps |
| 11 bits | 0 | 2 | 0 | 128 x 11 x 2 = | 2816 | 16 sec | 1408 bps |
| 11 bits | 0 | 2 | 1 | 64 x 11 x 2 = | 1408 | 16 sec | 704 bps |
| 11 bits | 0 | 2 | 2 | 32 x 11 x 2 = | 704 | 16 sec | 352 bps |
| 11 bits | 0 | 3 | 0 | 128 x 11 x 2 = | 2816 | 32 sec | 704 bps |
| 11 bits | 0 | 3 | 1 | 64 x 11 x 2 = | 1408 | 32 sec | 352 bps |
| 11 bits | 0 | 3 | 2 | 32 x 11 x 2 = | 704 | 32 sec | 176 bps |
| 11 bits | 0 | 4 | 0 | 128 x 11 x 2 = | 2816 | 64 sec | 352 bps |
| 11 bits | 0 | 4 | 1 | 64 x 11 x 2 = | 1408 | 64 sec | 176 bps |
| 11 bits | 0 | 4 | 2 | 32 x 11 x 2 = | 704 | 64 sec | 88 bps |
| 11 bits | 1 | 0 | 0 | 128 x 11 x 1 = | 1408 | 4 sec | 2816 bps |
| 11 bits | 1 | 0 | 1 | 64 x 11 x 1 = | 704 | 4 sec | 1408 bps |
| 11 bits | 1 | 0 | 2 | 32 x 11 x 1 = | 352 | 4 sec | 704 bps |
| 11 bits | 1 | 1 | 0 | 128 x 11 x 1 = | 1408 | 8 sec | 1408 bps |
| 11 bits | 1 | 1 | 1 | 64 x 11 x 1 = | 704 | 8 sec | 704 bps |
| 11 bits | 1 | 1 | 2 | 32 x 11 x 1 = | 352 | 8 sec | 352 bps |
| 11 bits | 1 | 2 | 0 | 128 x 11 x 1 = | 1408 | 16 sec | 704 bps |
| 11 bits | 1 | 2 | 1 | 64 x 11 x 1 = | 704 | 16 sec | 352 bps |
| 11 bits | 1 | 2 | 2 | 32 x 11 x 1 = | 352 | 16 sec | 176 bps |
| 11 bits | 1 | 3 | 0 | 128 x 11 x 1 = | 1408 | 32 sec | 352 bps |
| 11 bits | 1 | 3 | 1 | 64 x 11 x 1 = | 704 | 32 sec | 176 bps |
| 11 bits | 1 | 3 | 2 | 32 x 11 x 1 = | 352 | 32 sec | 88 bps |
| 11 bits | 1 | 4 | 0 | 128 x 11 x 1 = | 1408 | 64 sec | 176 bps |
| 11 bits | 1 | 4 | 1 | 64 x 11 x 1 = | 704 | 64 sec | 88 bps |
| 11 bits | 1 | 4 | 2 | 32 x 11 x 1 = | 352 | 64 sec | 44 bps |

| | | | | | | |
|---------|---|---|---|---------------------|--------|----------|
| 10 bits | 0 | 0 | 0 | 128 x 10 x 2 = 2560 | 4 sec | 5120 bps |
| 10 bits | 0 | 0 | 1 | 64 x 10 x 2 = 1280 | 4 sec | 2560 bps |
| 10 bits | 0 | 0 | 2 | 32 x 10 x 2 = 640 | 4 sec | 1280 bps |
| 10 bits | 0 | 1 | 0 | 128 x 10 x 2 = 2560 | 8 sec | 2560 bps |
| 10 bits | 0 | 1 | 1 | 64 x 10 x 2 = 1280 | 8 sec | 1280 bps |
| 10 bits | 0 | 1 | 2 | 32 x 10 x 2 = 640 | 8 sec | 640 bps |
| 10 bits | 0 | 2 | 0 | 128 x 10 x 2 = 2560 | 16 sec | 2560 bps |
| 10 bits | 0 | 2 | 1 | 64 x 10 x 2 = 1280 | 16 sec | 1280 bps |
| 10 bits | 0 | 2 | 2 | 32 x 10 x 2 = 640 | 16 sec | 640 bps |
| 10 bits | 0 | 3 | 0 | 128 x 10 x 2 = 2560 | 32 sec | 1280 bps |
| 10 bits | 0 | 3 | 1 | 64 x 10 x 2 = 1280 | 32 sec | 640 bps |
| 10 bits | 0 | 3 | 2 | 32 x 10 x 2 = 640 | 32 sec | 320 bps |
| 10 bits | 0 | 4 | 0 | 128 x 10 x 2 = 2560 | 64 sec | 640 bps |
| 10 bits | 0 | 4 | 1 | 64 x 10 x 2 = 1280 | 64 sec | 320 bps |
| 10 bits | 0 | 4 | 2 | 32 x 10 x 2 = 640 | 64 sec | 160 bps |
| 10 bits | 1 | 0 | 0 | 128 x 10 x 1 = 1280 | 4 sec | 2560 bps |
| 10 bits | 1 | 0 | 1 | 64 x 10 x 1 = 640 | 4 sec | 1280 bps |
| 10 bits | 1 | 0 | 2 | 32 x 10 x 1 = 320 | 4 sec | 640 bps |
| 10 bits | 1 | 1 | 0 | 128 x 10 x 1 = 1280 | 8 sec | 1280 bps |
| 10 bits | 1 | 1 | 1 | 64 x 10 x 1 = 640 | 8 sec | 640 bps |
| 10 bits | 1 | 1 | 2 | 32 x 10 x 1 = 320 | 8 sec | 320 bps |
| 10 bits | 1 | 2 | 0 | 128 x 10 x 1 = 1280 | 16 sec | 640 bps |
| 10 bits | 1 | 2 | 1 | 64 x 10 x 1 = 640 | 16 sec | 320 bps |
| 10 bits | 1 | 2 | 2 | 32 x 10 x 1 = 320 | 16 sec | 160 bps |
| 10 bits | 1 | 3 | 0 | 128 x 10 x 1 = 1280 | 32 sec | 320 bps |
| 10 bits | 1 | 3 | 1 | 64 x 10 x 1 = 640 | 32 sec | 160 bps |
| 10 bits | 1 | 3 | 2 | 32 x 10 x 1 = 320 | 32 sec | 80 bps |
| 10 bits | 1 | 4 | 0 | 128 x 10 x 1 = 1280 | 64 sec | 160 bps |
| 10 bits | 1 | 4 | 1 | 64 x 10 x 1 = 640 | 64 sec | 80 bps |
| 10 bits | 1 | 4 | 2 | 32 x 10 x 1 = 320 | 64 sec | 40 bps |
| 9 bits | 0 | 0 | 0 | 128 x 9 x 2 = 2304 | 4 sec | 4608 bps |
| 9 bits | 0 | 0 | 1 | 64 x 9 x 2 = 1152 | 4 sec | 2304 bps |
| 9 bits | 0 | 0 | 2 | 32 x 9 x 2 = 576 | 4 sec | 1152 bps |
| 9 bits | 0 | 1 | 0 | 128 x 9 x 2 = 2304 | 8 sec | 2304 bps |
| 9 bits | 0 | 1 | 1 | 64 x 9 x 2 = 1152 | 8 sec | 1152 bps |
| 9 bits | 0 | 1 | 2 | 32 x 9 x 2 = 576 | 8 sec | 576 bps |
| 9 bits | 0 | 2 | 0 | 128 x 9 x 2 = 2304 | 16 sec | 1152 bps |
| 9 bits | 0 | 2 | 1 | 64 x 9 x 2 = 1152 | 16 sec | 576 bps |
| 9 bits | 0 | 2 | 2 | 32 x 9 x 2 = 576 | 16 sec | 288 bps |
| 9 bits | 0 | 3 | 0 | 128 x 9 x 2 = 2304 | 32 sec | 576 bps |
| 9 bits | 0 | 3 | 1 | 64 x 9 x 2 = 1152 | 32 sec | 288 bps |
| 9 bits | 0 | 3 | 2 | 32 x 9 x 2 = 576 | 32 sec | 144 bps |
| 9 bits | 0 | 4 | 0 | 128 x 9 x 2 = 2304 | 64 sec | 288 bps |
| 9 bits | 0 | 4 | 1 | 64 x 9 x 2 = 1152 | 64 sec | 144 bps |
| 9 bits | 0 | 4 | 2 | 32 x 9 x 2 = 576 | 64 sec | 72 bps |
| 9 bits | 1 | 0 | 0 | 128 x 9 x 1 = 1152 | 4 sec | 2304 bps |
| 9 bits | 1 | 0 | 1 | 64 x 9 x 1 = 576 | 4 sec | 1152 bps |
| 9 bits | 1 | 0 | 2 | 32 x 9 x 1 = 288 | 4 sec | 576 bps |
| 9 bits | 1 | 1 | 0 | 129 x 9 x 1 = 1152 | 8 sec | 1152 bps |
| 9 bits | 1 | 1 | 1 | 64 x 9 x 1 = 576 | 8 sec | 576 bps |
| 9 bits | 1 | 1 | 2 | 32 x 9 x 1 = 288 | 8 sec | 288 bps |
| 9 bits | 1 | 2 | 0 | 128 x 9 x 1 = 1152 | 16 sec | 576 bps |
| 9 bits | 1 | 2 | 1 | 64 x 9 x 1 = 576 | 16 sec | 288 bps |
| 9 bits | 1 | 2 | 2 | 32 x 9 x 1 = 288 | 16 sec | 144 bps |
| 9 bits | 1 | 3 | 0 | 128 x 9 x 1 = 1152 | 32 sec | 288 bps |

| | | | | | | | | |
|--------|---|---|---|-------|---------|------|--------|----------|
| 9 bits | 1 | 3 | 1 | 64 x | 9 x 1 = | 576 | 32 sec | 144 bps |
| 9 bits | 1 | 3 | 2 | 32 x | 9 x 1 = | 288 | 32 sec | 72 bps |
| 9 bits | 1 | 4 | 0 | 128 x | 9 x 1 = | 1152 | 64 sec | 144 bps |
| 9 bits | 1 | 4 | 1 | 64 x | 9 x 1 = | 576 | 64 sec | 72 bps |
| 9 bits | 1 | 4 | 2 | 32 x | 9 x 1 = | 288 | 64 sec | 36 bps |
| 8 bits | 0 | 0 | 0 | 128 x | 8 x 2 = | 2048 | 4 sec | 4096 bps |
| 8 bits | 0 | 0 | 1 | 64 x | 8 x 2 = | 1024 | 4 sec | 2048 bps |
| 8 bits | 0 | 0 | 2 | 32 x | 8 x 2 = | 512 | 4 sec | 1024 bps |
| 8 bits | 0 | 1 | 0 | 128 x | 8 x 2 = | 2048 | 8 sec | 2048 bps |
| 8 bits | 0 | 1 | 1 | 64 x | 8 x 2 = | 1024 | 8 sec | 1024 bps |
| 8 bits | 0 | 1 | 2 | 32 x | 8 x 2 = | 512 | 8 sec | 512 bps |
| 8 bits | 0 | 2 | 0 | 128 x | 8 x 2 = | 2048 | 16 sec | 1024 bps |
| 8 bits | 0 | 2 | 1 | 64 x | 8 x 2 = | 1024 | 16 sec | 512 bps |
| 8 bits | 0 | 2 | 2 | 32 x | 8 x 2 = | 512 | 16 sec | 256 bps |
| 8 bits | 0 | 3 | 0 | 128 x | 8 x 2 = | 2048 | 32 sec | 512 bps |
| 8 bits | 0 | 3 | 1 | 64 x | 8 x 2 = | 1024 | 32 sec | 256 bps |
| 8 bits | 0 | 3 | 2 | 32 x | 8 x 2 = | 512 | 32 sec | 128 bps |
| 8 bits | 0 | 4 | 0 | 128 x | 8 x 2 = | 2048 | 64 sec | 256 bps |
| 8 bits | 0 | 4 | 1 | 64 x | 8 x 2 = | 1024 | 64 sec | 128 bps |
| 8 bits | 0 | 4 | 2 | 32 x | 8 x 2 = | 512 | 64 sec | 64 bps |
| 8 bits | 1 | 0 | 0 | 128 x | 8 x 1 = | 1024 | 4 sec | 2048 bps |
| 8 bits | 1 | 0 | 1 | 64 x | 8 x 1 = | 512 | 4 sec | 1024 bps |
| 8 bits | 1 | 0 | 2 | 32 x | 8 x 1 = | 256 | 4 sec | 512 bps |
| 8 bits | 1 | 1 | 0 | 128 x | 8 x 1 = | 1024 | 8 sec | 1024 bps |
| 8 bits | 1 | 1 | 1 | 64 x | 8 x 1 = | 512 | 8 sec | 512 bps |
| 8 bits | 1 | 1 | 2 | 32 x | 8 x 1 = | 256 | 8 sec | 256 bps |
| 8 bits | 1 | 2 | 0 | 128 x | 8 x 1 = | 1024 | 16 sec | 512 bps |
| 8 bits | 1 | 2 | 1 | 64 x | 8 x 1 = | 512 | 16 sec | 256 bps |
| 8 bits | 1 | 2 | 2 | 32 x | 8 x 1 = | 256 | 16 sec | 128 bps |
| 8 bits | 1 | 3 | 0 | 128 x | 8 x 1 = | 1024 | 32 sec | 256 bps |
| 8 bits | 1 | 3 | 1 | 64 x | 8 x 1 = | 512 | 32 sec | 128 bps |
| 8 bits | 1 | 3 | 2 | 32 x | 8 x 1 = | 256 | 32 sec | 64 bps |
| 8 bits | 1 | 4 | 0 | 128 x | 8 x 1 = | 1024 | 64 sec | 128 bps |
| 8 bits | 1 | 4 | 1 | 64 x | 8 x 1 = | 512 | 64 sec | 64 bps |
| 8 bits | 1 | 4 | 2 | 32 x | 8 x 1 = | 256 | 64 sec | 32 bps |
| 7 bits | 0 | 0 | 0 | 128 x | 7 x 2 = | 1792 | 4 sec | 3584 bps |
| 7 bits | 0 | 0 | 1 | 64 x | 7 x 2 = | 896 | 4 sec | 1792 bps |
| 7 bits | 0 | 0 | 2 | 32 x | 7 x 2 = | 448 | 4 sec | 896 bps |
| 7 bits | 0 | 1 | 0 | 128 x | 7 x 2 = | 1792 | 8 sec | 1792 bps |
| 7 bits | 0 | 1 | 1 | 64 x | 7 x 2 = | 896 | 8 sec | 896 bps |
| 7 bits | 0 | 1 | 2 | 32 x | 7 x 2 = | 448 | 8 sec | 448 bps |
| 7 bits | 0 | 2 | 0 | 128 x | 7 x 2 = | 1792 | 16 sec | 896 bps |
| 7 bits | 0 | 2 | 1 | 64 x | 7 x 2 = | 896 | 16 sec | 448 bps |
| 7 bits | 0 | 2 | 2 | 32 x | 7 x 2 = | 448 | 16 sec | 224 bps |
| 7 bits | 0 | 3 | 0 | 128 x | 7 x 2 = | 1792 | 32 sec | 448 bps |
| 7 bits | 0 | 3 | 1 | 64 x | 7 x 2 = | 896 | 32 sec | 224 bps |
| 7 bits | 0 | 3 | 2 | 32 x | 7 x 2 = | 448 | 32 sec | 112 bps |
| 7 bits | 0 | 4 | 0 | 128 x | 7 x 2 = | 1792 | 64 sec | 224 bps |
| 7 bits | 0 | 4 | 1 | 64 x | 7 x 2 = | 896 | 64 sec | 112 bps |
| 7 bits | 0 | 4 | 2 | 32 x | 7 x 2 = | 448 | 64 sec | 56 bps |
| 7 bits | 1 | 0 | 0 | 128 x | 7 x 1 = | 896 | 4 sec | 1792 bps |
| 7 bits | 1 | 0 | 1 | 64 x | 7 x 1 = | 448 | 4 sec | 896 bps |
| 7 bits | 1 | 0 | 2 | 32 x | 7 x 1 = | 224 | 4 sec | 448 bps |
| 7 bits | 1 | 1 | 0 | 128 x | 7 x 1 = | 896 | 8 sec | 896 bps |
| 7 bits | 1 | 1 | 1 | 64 x | 7 x 1 = | 448 | 8 sec | 448 bps |

| | | | | | | | | |
|--------|---|---|---|-------|---------|------|--------|----------|
| 7 bits | 1 | 1 | 2 | 32 x | 7 x 1 = | 224 | 8 sec | 224 bps |
| 7 bits | 1 | 2 | 0 | 128 x | 7 x 1 = | 896 | 16 sec | 448 bps |
| 7 bits | 1 | 2 | 1 | 64 x | 7 x 1 = | 448 | 16 sec | 224 bps |
| 7 bits | 1 | 2 | 2 | 32 x | 7 x 1 = | 224 | 16 sec | 112 bps |
| 7 bits | 1 | 3 | 0 | 128 x | 7 x 1 = | 896 | 32 sec | 224 bps |
| 7 bits | 1 | 3 | 1 | 64 x | 7 x 1 = | 448 | 32 sec | 112 bps |
| 7 bits | 1 | 3 | 2 | 32 x | 7 x 1 = | 224 | 32 sec | 56 bps |
| 7 bits | 1 | 4 | 0 | 128 x | 7 x 1 = | 896 | 64 sec | 112 bps |
| 7 bits | 1 | 4 | 1 | 64 x | 7 x 1 = | 448 | 64 sec | 56 bps |
| 7 bits | 1 | 4 | 2 | 32 x | 7 x 1 = | 224 | 64 sec | 28 bps |
| 6 bits | 0 | 0 | 0 | 128 x | 6 x 2 = | 1536 | 4 sec | 3072 bps |
| 6 bits | 0 | 0 | 1 | 64 x | 6 x 2 = | 768 | 4 sec | 1536 bps |
| 6 bits | 0 | 0 | 2 | 32 x | 6 x 2 = | 384 | 4 sec | 768 bps |
| 6 bits | 0 | 1 | 0 | 128 x | 6 x 2 = | 1536 | 8 sec | 1536 bps |
| 6 bits | 0 | 1 | 1 | 64 x | 6 x 2 = | 768 | 8 sec | 768 bps |
| 6 bits | 0 | 1 | 2 | 32 x | 6 x 2 = | 384 | 8 sec | 384 bps |
| 6 bits | 0 | 2 | 0 | 128 x | 6 x 2 = | 1536 | 16 sec | 768 bps |
| 6 bits | 0 | 2 | 1 | 64 x | 6 x 2 = | 768 | 16 sec | 384 bps |
| 6 bits | 0 | 2 | 2 | 32 x | 6 x 2 = | 384 | 16 sec | 192 bps |
| 6 bits | 0 | 3 | 0 | 128 x | 6 x 2 = | 1536 | 32 sec | 384 bps |
| 6 bits | 0 | 3 | 1 | 64 x | 6 x 2 = | 768 | 32 sec | 192 bps |
| 6 bits | 0 | 3 | 2 | 32 x | 6 x 2 = | 384 | 32 sec | 96 bps |
| 6 bits | 0 | 4 | 0 | 128 x | 6 x 2 = | 1536 | 64 sec | 192 bps |
| 6 bits | 0 | 4 | 1 | 64 x | 6 x 2 = | 768 | 64 sec | 96 bps |
| 6 bits | 0 | 4 | 2 | 32 x | 6 x 2 = | 384 | 64 sec | 48 bps |
| 6 bits | 1 | 0 | 0 | 128 x | 6 x 1 = | 768 | 4 sec | 1536 bps |
| 6 bits | 1 | 0 | 1 | 64 x | 6 x 1 = | 384 | 4 sec | 768 bps |
| 6 bits | 1 | 0 | 2 | 32 x | 6 x 1 = | 192 | 4 sec | 384 bps |
| 6 bits | 1 | 1 | 0 | 128 x | 6 x 1 = | 768 | 8 sec | 768 bps |
| 6 bits | 1 | 1 | 1 | 64 x | 6 x 1 = | 384 | 8 sec | 384 bps |
| 6 bits | 1 | 1 | 2 | 32 x | 6 x 1 = | 192 | 8 sec | 192 bps |
| 6 bits | 1 | 2 | 0 | 128 x | 6 x 1 = | 768 | 16 sec | 384 bps |
| 6 bits | 1 | 2 | 1 | 64 x | 6 x 1 = | 384 | 16 sec | 192 bps |
| 6 bits | 1 | 2 | 2 | 32 x | 6 x 1 = | 192 | 16 sec | 96 bps |
| 6 bits | 1 | 3 | 0 | 128 x | 6 x 1 = | 768 | 32 sec | 192 bps |
| 6 bits | 1 | 3 | 1 | 64 x | 6 x 1 = | 384 | 32 sec | 96 bps |
| 6 bits | 1 | 3 | 2 | 32 x | 6 x 1 = | 192 | 32 sec | 48 bps |
| 6 bits | 1 | 4 | 0 | 128 x | 6 x 1 = | 768 | 64 sec | 96 bps |
| 6 bits | 1 | 4 | 1 | 64 x | 6 x 1 = | 384 | 64 sec | 48 bps |
| 6 bits | 1 | 4 | 2 | 32 x | 6 x 1 = | 192 | 64 sec | 24 bps |
| 5 bits | 0 | 0 | 0 | 128 x | 5 x 2 = | 1280 | 4 sec | 2560 bps |
| 5 bits | 0 | 0 | 1 | 64 x | 5 x 2 = | 640 | 4 sec | 1280 bps |
| 5 bits | 0 | 0 | 2 | 32 x | 5 x 2 = | 320 | 4 sec | 640 bps |
| 5 bits | 0 | 1 | 0 | 128 x | 5 x 2 = | 1280 | 8 sec | 1280 bps |
| 5 bits | 0 | 1 | 1 | 64 x | 5 x 2 = | 640 | 8 sec | 640 bps |
| 5 bits | 0 | 1 | 2 | 32 x | 5 x 2 = | 320 | 8 sec | 320 bps |
| 5 bits | 0 | 2 | 0 | 128 x | 5 x 2 = | 1280 | 16 sec | 640 bps |
| 5 bits | 0 | 2 | 1 | 64 x | 5 x 2 = | 640 | 16 sec | 320 bps |
| 5 bits | 0 | 2 | 2 | 32 x | 5 x 2 = | 320 | 16 sec | 160 bps |
| 5 bits | 0 | 3 | 0 | 128 x | 5 x 2 = | 1280 | 32 sec | 320 bps |
| 5 bits | 0 | 3 | 1 | 64 x | 5 x 2 = | 640 | 32 sec | 160 bps |
| 5 bits | 0 | 3 | 2 | 32 x | 5 x 2 = | 320 | 32 sec | 80 bps |
| 5 bits | 0 | 4 | 0 | 128 x | 5 x 2 = | 1280 | 64 sec | 160 bps |
| 5 bits | 0 | 4 | 1 | 64 x | 5 x 2 = | 640 | 64 sec | 80 bps |
| 5 bits | 0 | 4 | 2 | 32 x | 5 x 2 = | 320 | 64 sec | 40 bps |

| | | | | | | | | |
|--------|---|---|---|-------|---------|------|--------|----------|
| 5 bits | 1 | 0 | 0 | 128 x | 5 x 1 = | 640 | 4 sec | 1280 bps |
| 5 bits | 1 | 0 | 1 | 64 x | 5 x 1 = | 320 | 4 sec | 640 bps |
| 5 bits | 1 | 0 | 2 | 32 x | 5 x 1 = | 160 | 4 sec | 320 bps |
| 5 bits | 1 | 1 | 0 | 128 x | 5 x 1 = | 640 | 8 sec | 640 bps |
| 5 bits | 1 | 1 | 1 | 64 x | 5 x 1 = | 320 | 8 sec | 320 bps |
| 5 bits | 1 | 1 | 2 | 32 x | 5 x 1 = | 160 | 8 sec | 160 bps |
| 5 bits | 1 | 2 | 0 | 128 x | 5 x 1 = | 640 | 16 sec | 320 bps |
| 5 bits | 1 | 2 | 1 | 64 x | 5 x 1 = | 320 | 16 sec | 160 bps |
| 5 bits | 1 | 2 | 2 | 32 x | 5 x 1 = | 160 | 16 sec | 80 bps |
| 5 bits | 1 | 3 | 0 | 128 x | 5 x 1 = | 640 | 32 sec | 160 bps |
| 5 bits | 1 | 3 | 1 | 64 x | 5 x 1 = | 320 | 32 sec | 80 bps |
| 5 bits | 1 | 3 | 2 | 32 x | 5 x 1 = | 160 | 32 sec | 40 bps |
| 5 bits | 1 | 4 | 0 | 128 x | 5 x 1 = | 640 | 64 sec | 80 bps |
| 5 bits | 1 | 4 | 1 | 64 x | 5 x 1 = | 320 | 64 sec | 40 bps |
| 5 bits | 1 | 4 | 2 | 32 x | 5 x 1 = | 160 | 64 sec | 20 bps |
| 4 bits | 0 | 0 | 0 | 128 x | 4 x 2 = | 1024 | 4 sec | 2048 bps |
| 4 bits | 0 | 0 | 1 | 64 x | 4 x 2 = | 512 | 4 sec | 1024 bps |
| 4 bits | 0 | 0 | 2 | 32 x | 4 x 2 = | 256 | 4 sec | 512 bps |
| 4 bits | 0 | 1 | 0 | 128 x | 4 x 2 = | 1024 | 8 sec | 1024 bps |
| 4 bits | 0 | 1 | 1 | 64 x | 4 x 2 = | 512 | 8 sec | 512 bps |
| 4 bits | 0 | 1 | 2 | 32 x | 4 x 2 = | 256 | 8 sec | 256 bps |
| 4 bits | 0 | 2 | 0 | 128 x | 4 x 2 = | 1024 | 16 sec | 512 bps |
| 4 bits | 0 | 2 | 1 | 64 x | 4 x 2 = | 512 | 16 sec | 256 bps |
| 4 bits | 0 | 2 | 2 | 32 x | 4 x 2 = | 256 | 16 sec | 128 bps |
| 4 bits | 0 | 3 | 0 | 128 x | 4 x 2 = | 1024 | 32 sec | 256 bps |
| 4 bits | 0 | 3 | 1 | 64 x | 4 x 2 = | 512 | 32 sec | 128 bps |
| 4 bits | 0 | 3 | 2 | 32 x | 4 x 2 = | 256 | 32 sec | 64 bps |
| 4 bits | 0 | 4 | 0 | 128 x | 4 x 2 = | 1024 | 64 sec | 128 bps |
| 4 bits | 0 | 4 | 1 | 64 x | 4 x 2 = | 512 | 64 sec | 64 bps |
| 4 bits | 0 | 4 | 2 | 32 x | 4 x 2 = | 256 | 64 sec | 32 bps |
| 4 bits | 1 | 0 | 0 | 128 x | 4 x 1 = | 512 | 4 sec | 1024 bps |
| 4 bits | 1 | 0 | 1 | 64 x | 4 x 1 = | 256 | 4 sec | 512 bps |
| 4 bits | 1 | 0 | 2 | 32 x | 4 x 1 = | 128 | 4 sec | 256 bps |
| 4 bits | 1 | 1 | 0 | 128 x | 4 x 1 = | 512 | 8 sec | 512 bps |
| 4 bits | 1 | 1 | 1 | 64 x | 4 x 1 = | 256 | 8 sec | 256 bps |
| 4 bits | 1 | 1 | 2 | 32 x | 4 x 1 = | 128 | 8 sec | 128 bps |
| 4 bits | 1 | 2 | 0 | 128 x | 4 x 1 = | 512 | 16 sec | 256 bps |
| 4 bits | 1 | 2 | 1 | 64 x | 4 x 1 = | 256 | 16 sec | 128 bps |
| 4 bits | 1 | 2 | 2 | 32 x | 4 x 1 = | 128 | 16 sec | 64 bps |
| 4 bits | 1 | 3 | 0 | 128 x | 4 x 1 = | 512 | 32 sec | 128 bps |
| 4 bits | 1 | 3 | 1 | 64 x | 4 x 1 = | 256 | 32 sec | 64 bps |
| 4 bits | 1 | 3 | 2 | 32 x | 4 x 1 = | 128 | 32 sec | 32 bps |
| 4 bits | 1 | 4 | 0 | 128 x | 4 x 1 = | 512 | 64 sec | 64 bps |
| 4 bits | 1 | 4 | 1 | 64 x | 4 x 1 = | 256 | 64 sec | 32 bps |
| 4 bits | 1 | 4 | 2 | 32 x | 4 x 1 = | 128 | 64 sec | 16 bps |
| 3 bits | 0 | 0 | 0 | 128 x | 3 x 2 = | 768 | 4 sec | 1536 bps |
| 3 bits | 0 | 0 | 1 | 64 x | 3 x 2 = | 384 | 4 sec | 768 bps |
| 3 bits | 0 | 0 | 2 | 32 x | 3 x 2 = | 192 | 4 sec | 384 bps |
| 3 bits | 0 | 1 | 0 | 128 x | 3 x 2 = | 768 | 8 sec | 768 bps |
| 3 bits | 0 | 1 | 1 | 64 x | 3 x 2 = | 384 | 8 sec | 384 bps |
| 3 bits | 0 | 1 | 2 | 32 x | 3 x 2 = | 192 | 8 sec | 192 bps |
| 3 bits | 0 | 2 | 0 | 128 x | 3 x 2 = | 768 | 16 sec | 384 bps |
| 3 bits | 0 | 2 | 1 | 64 x | 3 x 2 = | 384 | 16 sec | 192 bps |
| 3 bits | 0 | 2 | 2 | 32 x | 3 x 2 = | 192 | 16 sec | 96 bps |
| 3 bits | 0 | 3 | 0 | 128 x | 3 x 2 = | 768 | 32 sec | 192 bps |

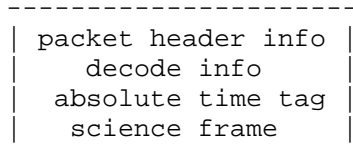
| | | | | | | | | |
|--------|---|---|---|-------|---------|-----|--------|----------|
| 3 bits | 0 | 3 | 1 | 64 x | 3 x 2 = | 384 | 32 sec | 96 bps |
| 3 bits | 0 | 3 | 2 | 32 x | 3 x 2 = | 192 | 32 sec | 48 bps |
| 3 bits | 0 | 4 | 0 | 128 x | 3 x 2 = | 768 | 64 sec | 96 bps |
| 3 bits | 0 | 4 | 1 | 64 x | 3 x 2 = | 384 | 64 sec | 48 bps |
| 3 bits | 0 | 4 | 2 | 32 x | 3 x 2 = | 192 | 64 sec | 24 bps |
| 3 bits | 1 | 0 | 0 | 128 x | 3 x 1 = | 384 | 4 sec | 768 bps |
| 3 bits | 1 | 0 | 1 | 64 x | 3 x 1 = | 192 | 4 sec | 384 bps |
| 3 bits | 1 | 0 | 2 | 32 x | 3 x 1 = | 96 | 4 sec | 192 bps |
| 3 bits | 1 | 1 | 0 | 128 x | 3 x 1 = | 384 | 8 sec | 384 bps |
| 3 bits | 1 | 1 | 1 | 64 x | 3 x 1 = | 192 | 8 sec | 192 bps |
| 3 bits | 1 | 1 | 2 | 32 x | 3 x 1 = | 96 | 8 sec | 96 bps |
| 3 bits | 1 | 2 | 0 | 128 x | 3 x 1 = | 384 | 16 sec | 192 bps |
| 3 bits | 1 | 2 | 1 | 64 x | 3 x 1 = | 192 | 16 sec | 96 bps |
| 3 bits | 1 | 2 | 2 | 32 x | 3 x 1 = | 96 | 16 sec | 48 bps |
| 3 bits | 1 | 3 | 0 | 128 x | 3 x 1 = | 384 | 32 sec | 96 bps |
| 3 bits | 1 | 3 | 1 | 64 x | 3 x 1 = | 192 | 32 sec | 48 bps |
| 3 bits | 1 | 3 | 2 | 32 x | 3 x 1 = | 96 | 32 sec | 24 bps |
| 3 bits | 1 | 4 | 0 | 128 x | 3 x 1 = | 384 | 64 sec | 48 bps |
| 3 bits | 1 | 4 | 1 | 64 x | 3 x 1 = | 192 | 64 sec | 24 bps |
| 3 bits | 1 | 4 | 2 | 32 x | 3 x 1 = | 96 | 64 sec | 12 bps |
| 2 bits | 0 | 0 | 0 | 128 x | 2 x 2 = | 512 | 4 sec | 1024 bps |
| 2 bits | 0 | 0 | 1 | 64 x | 2 x 2 = | 256 | 4 sec | 512 bps |
| 2 bits | 0 | 0 | 2 | 32 x | 2 x 2 = | 128 | 4 sec | 256 bps |
| 2 bits | 0 | 1 | 0 | 128 x | 2 x 2 = | 512 | 8 sec | 512 bps |
| 2 bits | 0 | 1 | 1 | 64 x | 2 x 2 = | 256 | 8 sec | 256 bps |
| 2 bits | 0 | 1 | 2 | 32 x | 2 x 2 = | 128 | 8 sec | 128 bps |
| 2 bits | 0 | 2 | 0 | 128 x | 2 x 2 = | 512 | 16 sec | 256 bps |
| 2 bits | 0 | 2 | 1 | 64 x | 2 x 2 = | 256 | 16 sec | 128 bps |
| 2 bits | 0 | 2 | 2 | 32 x | 2 x 2 = | 128 | 16 sec | 64 bps |
| 2 bits | 0 | 3 | 0 | 128 x | 2 x 2 = | 512 | 32 sec | 128 bps |
| 2 bits | 0 | 3 | 1 | 64 x | 2 x 2 = | 256 | 32 sec | 64 bps |
| 2 bits | 0 | 3 | 2 | 32 x | 2 x 2 = | 128 | 32 sec | 32 bps |
| 2 bits | 0 | 4 | 0 | 128 x | 2 x 2 = | 512 | 64 sec | 64 bps |
| 2 bits | 0 | 4 | 1 | 64 x | 2 x 2 = | 256 | 64 sec | 32 bps |
| 2 bits | 0 | 4 | 2 | 32 x | 2 x 2 = | 128 | 64 sec | 16 bps |
| 2 bits | 1 | 0 | 0 | 128 x | 2 x 1 = | 256 | 4 sec | 512 bps |
| 2 bits | 1 | 0 | 1 | 64 x | 2 x 1 = | 128 | 4 sec | 256 bps |
| 2 bits | 1 | 0 | 2 | 32 x | 2 x 1 = | 64 | 4 sec | 128 bps |
| 2 bits | 1 | 1 | 0 | 128 x | 2 x 1 = | 256 | 8 sec | 256 bps |
| 2 bits | 1 | 1 | 1 | 64 x | 2 x 1 = | 128 | 8 sec | 128 bps |
| 2 bits | 1 | 1 | 2 | 32 x | 2 x 1 = | 64 | 8 sec | 64 bps |
| 2 bits | 1 | 2 | 0 | 129 x | 2 x 1 = | 256 | 16 sec | 128 bps |
| 2 bits | 1 | 2 | 1 | 64 x | 2 x 1 = | 128 | 16 sec | 64 bps |
| 2 bits | 1 | 2 | 2 | 32 x | 2 x 1 = | 64 | 16 sec | 32 bps |
| 2 bits | 1 | 3 | 0 | 128 x | 2 x 1 = | 256 | 32 sec | 64 bps |
| 2 bits | 1 | 3 | 1 | 64 x | 2 x 1 = | 128 | 32 sec | 32 bps |
| 2 bits | 1 | 3 | 2 | 32 x | 2 x 1 = | 64 | 32 sec | 16 bps |
| 2 bits | 1 | 4 | 0 | 128 x | 2 x 1 = | 256 | 64 sec | 32 bps |
| 2 bits | 1 | 4 | 1 | 64 x | 2 x 1 = | 128 | 64 sec | 16 bps |
| 2 bits | 1 | 4 | 2 | 32 x | 2 x 1 = | 64 | 64 sec | 8 bps |
| 1 bit | 0 | 0 | 0 | 128 x | 1 x 2 = | 256 | 4 sec | 512 bps |
| 1 bit | 0 | 0 | 1 | 64 x | 1 x 2 = | 128 | 4 sec | 256 bps |
| 1 bit | 0 | 0 | 2 | 32 x | 1 x 2 = | 64 | 4 sec | 128 bps |
| 1 bit | 0 | 1 | 0 | 128 x | 1 x 2 = | 256 | 8 sec | 256 bps |
| 1 bit | 0 | 1 | 1 | 64 x | 1 x 2 = | 128 | 8 sec | 128 bps |

| | | | | | | | | |
|-------|---|---|---|-------|---------|-----|--------|---------|
| 1 bit | 0 | 1 | 2 | 32 x | 1 x 2 = | 64 | 8 sec | 64 bps |
| 1 bit | 0 | 2 | 0 | 128 x | 1 x 2 = | 256 | 16 sec | 128 bps |
| 1 bit | 0 | 2 | 1 | 64 x | 1 x 2 = | 128 | 16 sec | 64 bps |
| 1 bit | 0 | 2 | 2 | 32 x | 1 x 2 = | 64 | 16 sec | 32 bps |
| 1 bit | 0 | 3 | 0 | 128 x | 1 x 2 = | 256 | 32 sec | 64 bps |
| 1 bit | 0 | 3 | 1 | 64 x | 1 x 2 = | 128 | 32 sec | 32 bps |
| 1 bit | 0 | 3 | 2 | 32 x | 1 x 2 = | 64 | 32 sec | 16 bps |
| 1 bit | 0 | 4 | 0 | 128 x | 1 x 2 = | 256 | 64 sec | 32 bps |
| 1 bit | 0 | 4 | 1 | 64 x | 1 x 2 = | 128 | 64 sec | 16 bps |
| 1 bit | 0 | 4 | 2 | 32 x | 1 x 2 = | 64 | 64 sec | 8 bps |
| 1 bit | 1 | 0 | 0 | 128 x | 1 x 1 = | 128 | 4 sec | 256 bps |
| 1 bit | 1 | 0 | 1 | 64 x | 1 x 1 = | 64 | 4 sec | 128 bps |
| 1 bit | 1 | 0 | 2 | 32 x | 1 x 1 = | 32 | 4 sec | 64 bps |
| 1 bit | 1 | 1 | 0 | 128 x | 1 x 1 = | 128 | 8 sec | 128 bps |
| 1 bit | 1 | 1 | 1 | 64 x | 1 x 1 = | 64 | 8 sec | 64 bps |
| 1 bit | 1 | 1 | 2 | 32 x | 1 x 1 = | 32 | 8 sec | 32 bps |
| 1 bit | 1 | 2 | 0 | 128 x | 1 x 1 = | 128 | 16 sec | 64 bps |
| 1 bit | 1 | 2 | 1 | 64 x | 1 x 1 = | 64 | 16 sec | 32 bps |
| 1 bit | 1 | 2 | 2 | 32 x | 1 x 1 = | 32 | 16 sec | 16 bps |
| 1 bit | 1 | 3 | 0 | 128 x | 1 x 1 = | 128 | 32 sec | 32 bps |
| 1 bit | 1 | 3 | 1 | 64 x | 1 x 1 = | 64 | 32 sec | 16 bps |
| 1 bit | 1 | 3 | 2 | 32 x | 1 x 1 = | 32 | 32 sec | 8 bps |
| 1 bit | 1 | 4 | 0 | 128 x | 1 x 1 = | 128 | 64 sec | 16 bps |
| 1 bit | 1 | 4 | 1 | 64 x | 1 x 1 = | 64 | 64 sec | 8 bps |
| 1 bit | 1 | 4 | 2 | 32 x | 1 x 1 = | 32 | 64 sec | 4 bps |

The above compressions are termed lossy compression because some of the data resolution is lost when forming the final product. There is a lossless data compression technique which will be used by all ASPERA-3 science data called Rice Compression. Rice Compression may be deactivated. When using the ASPERA-3 GSE, they will perform tests where the Rice Compression is disabled. Currently, it is envisioned that data from the spacecraft will always be Rice Compressed. Exactly when Rice Compression is invoked should be defined by FMI.

One or more science matrices will be accumulated for a science frame. The size will be defined by FMI. In a science data packet, there should be some decode information (with respect to compression schemes utilized), the absolute time tag information, and then the science frame. At least one science packet should be produced in a scanner cycle. Science data should never be accumulated across a scanner boundary. Science packets can be generated at a faster rate than the scanner cycle, but never at a slower rate than the scanner cycle.

So, the science packet should look like this:



Here is some helpful info on the different compression techniques. If they must be invoked, the top priority should be sector mask and then bit compression and then time summation and lastly energy summation. The reason

for having so many compression techniques is due to telemetry restrictions. Ideally, ELS would transmit all of its data all of the time; however since the Mars Express spacecraft has imposed telemetry limitations, ELS must compress its data. The goal is to get down as much "good quality" data as possible. Here, the "good quality" is defined by the science. For various studies, time/angle resolution will be emphasized over energy resolution. For others, Energy resolution will be emphasized. Everything using a lossy compression technique represents a tradeoff and that tradeoff will be defined by the science, which will be variable.

Sector Mask

The sector mask is a 16 bit field, one bit representing each sector of ELS; the sector mask is a 1-for-1 bit-to-sector correspondence. The sector mask is meant to block unwanted instrument data from entering the telemetry stream. The blocked data is from that sector which has data determined to be obstructed or not scientifically useful. The aim here is to not use TM space by not bringing down data which is of questionable use or not as important; TM may be freed for other data or higher resolution data (fewer compression techniques applied). So one of the ways to get down more of the useful data from sectors is to block data from those ELS sectors which do not yield the effect that the scientists wish to study.

At the beginning of the mission, all ELS sector data will be sent down. At the beginning of the mission, the scientists are not going to know exactly what sector will be producing "bad data" for their studies. The most raised question is the sectors that look into the spacecraft. This is judging from line-of-sight. However, electrons follow magnetic field and not the line-of-sight. The path of the electrons may skirt around the spacecraft and show no influence, show spacecraft shadowing in another sector, show an energy dependent influence, or show a sector change of the influence depending on position. At any rate, Mars will be sufficiently complicated so that a predetermination will not be able to be determined.

When a bit within the sector mask is turned on, then data from that sector is allowed to pass to the telemetry stream and accumulates in the science matrix. At the start of the mission, all sectors should pass their data to be accumulated within the science data matrix.

Bit Compression

Normally, ELS words are 16 bits long. This format gives the best resolution to the data. Bit compression may be used to save space. Bit compression is a lossy compression of compressing the normal 16-bit word into 8-bits. When bit compression is on, there are additional errors that arise in the science analysis due to the uncertainty in the determination of the telemetry which is caused by representing a 16-bit number by only 8-bits.

Bit compression techniques are designed to represent as much of the data value at low counts as possible and it applies more bit compression to the larger data values. In the analysis of the data, this adds in an additional error term and causes a limit to the error in the larger data values. This limits the accuracy of any science analysis performed with this data.

At the initial stage, bit compression will most likely have to be turned on. As data are excluded by the sector mask, it may be possible to return the full 16-bit data words from ELS.

Time Summation

Time summation adds together the values of science matrices before sending the telemetry (science frame). The time summation causes increase in counting values at the expense of time resolution. Since the scanner runs at 128 sec, 64 sec, or 32 sec, setting the time summation could also be thought of as coarsening the angular resolution at the expense of larger data values. For example, if the scanner is running at the 128 sec rate, and the time compression is set to accumulate 4 sweeps, then the angular resolution which is achieved will be the same as if the scanner were running at the 32 sec rate. However, the accumulated data will be 4 times greater for the time summed case over the data taken at the faster rate scanner cycle.

Time summations are additions of complete energy arrays. The steep accumulation period should not be adjusted to account for a time summation. What should happen is that the DPU adds two arrays together. Time summation should occur after the sector mask is applied, but before bit compression. The order of application should be as follows:

Energy Summation -> Time Summation -> Sector Mask -> Bit Compression -> Rice Compression

Time summation should never represent more time than a scanner cycle takes. For example, a time summation representing a 64 sec sum should not be allowed when the scanner cycle is in the 32 sec scan mode. Below is a table showing the allow states of time summation versus scanner cycle:

| Time Summation | Scanner Cycle | | |
|----------------|---------------|---------|---------|
| | 32 sec | 64 sec | 128 sec |
| 0 - 4 sec | Allowed | Allowed | Allowed |
| 1 - 8 sec | Allowed | Allowed | Allowed |
| 2 - 16 sec | Allowed | Allowed | Allowed |
| 3 - 32 sec | Allowed | Allowed | Allowed |
| 4 - 64 sec | Not Allowed | Allowed | Allowed |

Energy Summation

In most cases of Mars Express, we will only use energy summing as a last resort to get down data. Normally the ELS energy step is 128 values. Scientifically, we over sample the region of the energy spectrum where photoelectron peaks occur and under sample other regions. ELS was designed to try to reproduce the significant portions of the energy spectrum with enough resolution to obtain an accurate resemblance of the energy spectrum. The energy spectrum should show peaks corresponding to various gasses that are present in the Martian atmosphere. ELS was designed to detect the electrons produced when Martian gasses in its atmosphere ionize. Summing energy levels defeats this purpose. One will not be able to reconstruct an accurate energy spectrum.

However, there are cases where energy spectrum summation could be used to an advantage. Since the ELS energy sweep is programmable, one could program

adjacent energy levels to the same value, and then sum energy steps. This would create a smaller energy spectrum with better counting statistics (the values returned in TM will be from larger numbers).

Summation occurs for adjacent energy levels. From the original 128 step energy sweep, energy summation adds together two adjacent sweeps creating a 64 point energy spectrum. Here steps 0+1, 2+3, 4+5, 6+7, etc. are output. The energy summation can also be set to sum together 4 energy step values. Here steps 0+1+2+3, 4+5+6+7, 8+9+10+11, 12+13+14+15, etc. are output. When analyzing this data, one must be aware of what energy steps are summed together in order to represent the energy correctly.

Order of Application within the DPU

The DPU collects a stream of data from ELS. The all sector data for an energy step is generated in parallel. The DPU controls the energy step by cycling through the deflection voltages. Thus, from the DPU's perspective, it is easiest to do the Energy Summation first. Here the 16 samples can be accumulated by the correct amount before being released further into the DPU. In essence, the registers are accumulated for 1, 2, or 4 samples before strobing out to higher functions. The appropriate energy steps can then be accumulated over the sweep so that the resulting sweep is 128, 64, or 32 steps long.

The second application should then be the time summation. Here, the stream (after energy summation) is accumulated in an array in DPU memory. When summing time steps, you add together samples from only the same energy step.

The third application should be the sector mask. The sector mask could be applied to the data as it is transferring to the next stage, which is the bit compression. The bit compression should be done after all summations. To save compute time, bit compression should only occur on data actually being telemetered.

After bit compression, Rice Compression is applied. At this point, the files are ready for the telemetry stream. Thus the order should be:

```
Data From      Energy      Time      Sector      Bit
ELS           -> Summation -> Summation -> Mask   -> compression ->

              Rice
              compression -> TM
```

Under no circumstances should summations occur on compressed data. Summations should occur only on the full 16 bit number registers of the data.

Meaning of Each Compression to Processing

To unfold the data during processing, adjustments must be made in the way in which data is interpreted. Data from the Mars Express spacecraft for ELS is likely to be compressed in some manner. The first issue to deal with is Rice Decompression. If the science data is Rice Compressed, it must be Rice Decompressed in order to interpret the data. There should be coordination between FMI and SwRI so that the proper compression/decompression algorithms are used at both ends. We want the ELS data to be reproduced. ELS data

before Rice Compression in the DPU should be the same as after the ELS data is Rice Decompressed on the ground.

Bit compression means that there is a look-up table which must be added to the science VIDFs which translate the 8-bit words into 16-bit words. When Bit Compression is off, there is no table applied. The Bit Compression flag translates into an instrument mode in the Processing software.

The sector mask means that there is data missing from the science frame. The sector mask should be interpreted on the ground and used to expand the ELS telemetry. The sector mask is an indication of the ELS sectors available in the telemetry.

Time summation is again a mode. Processing chooses an appropriate accumulation time and places this in the header. There will also be a need to shift the time normalization, so the time summation flag will need to be used as a mode, specifying which set of timing to use.

Lastly, energy summation has two effects on the processing. First it tells how many energy steps there are in the sweep and second, it is used as a mode to control the energy look-up table values.

Appendix C – Decompression of ELS Packets

There are two types of ELS packets. The first type is an ELS engineering packet. The ELS engineering packet comes once per scanner cycle (32, 64, or 128 s) and is not compressed. It contains the reference values and monitor values for all 128 ELS sweep steps besides other engineering parameters. The ELS sweep data occur at byte 38 and after. There are four types of important quantities that you will pull from byte 38 and beyond. The first type is the position within the sweep for a particular energy step. The values should range from 0 to 127 and this is created by your own internal counter as you read the values in the packet. The energy step should be discerned from the order in which the data is positioned within the ELS engineering packet. This internal counter forms the sweep index written in the header record of **ELSSWPH / ELSSWPL**.

The second type of important information is the power supply range bit. The power supply range bit determines whether the energy step is in low range or high range. The power supply values are dual valued, so you need to look at the range bit to tell if the power supply and data will be in low range or high range. This will determine in which virtual data is written.

The third type is reference data. This indicates the value to which the power supply commanded when acquiring data. The value of the reference is written as the index of the step in the header record of **ELSSCIH / ELSSCIL** and is written as data at the sweep sequence in **ELSSWPH / ELSSWPL**. You should use the range bit to determine whether the data is placed in a low or high range IDFS. ELS is programmable so that the value represented by step 6 (for example) may not be the same from scanner cycle to scanner cycle.

The fourth type is the monitor value for that step. The monitor value indicates what was actually achieved. The monitor value is the data value which is written in **ELSSWPH / ELSSWPL** at the index of the energy step along with its corresponding reference data value.

The second type of ELS packet is the ELS science packet. When viewing the ELS science packet, you have to look at the ELS sector mask, Rice Compression bit, Log Compression bit, Energy Compression bits, and Time Compression bits. At this point, you are going to fill in a matrix which is 16 anodes by 128 energy steps long. The first active anode is written in the first location of the array. The second active anode is written in the second anode position of the matrix, etc. Here, the first active anode does not have to be the Anode 0 and the second active anode does not have to be adjacent to the first active anode; however, the anode represented must be increasing as more anode data is detected. The mask byte will tell you which anodes are active.

The first thing to do is to examine the Rice Compression bit. The Rice Compression bit will tell you if bytes 32 to the end of the packet are Rice Compressed. The data must be Rice Decompressed if the Rice Compression bit is set.

The second thing to do is determine the word size by looking at the Log Compression bit. If that is not set, then the words are 16 bit long. If that bit is set, then the data are 8 bits long and must be decompressed into the 16 bit word. Use the f8 decompression routine to expand the compressed 8 bit word into the 16 bit word.

The third thing to do is to look at the energy summation. If this is 0, then there should be 128 energy steps per sensor, if this is 1, then there should be 64 steps per sensor, and if this is 2, then there should be 32 steps per sensor. A 0 will mean that there is a data value corresponding to each energy step. A value of 1 means that two data values have been summed together, 0 & 1, 2 & 3, 4 & 5, 6 & 7, 8 & 9, 10 & 11, etc. A value of 2 means that four data values have been summed together, 0 & 1 & 2 & 3, 4 & 5 & 6 & 7, 8 & 9 & 10 & 11, etc. When placing values into your matrix, you will repeat the values in all locations represented by the summation. For example, if the energy compression is a 1, the first data value is placed in both step 0 and step 1, and the second data value is placed in step 2 and step 3, etc.

The fourth thing to do is to look at the sector mask. The sector mask corresponds to the ELS anodes and tells which are returning data. If a sector is masked, there will be no data from that anode in the data field. Values are written into the packet as anode varying, first. So, you will want to start reading the data value and placing them into the matrix in sequential order of the number of active anode. The sensor indices will be the locations of the active anodes in the sector mask. Look at the sector mask to determine the sensor and count the number of times you read all active anodes from the sector mask. The number of cycles of the sector mask will tell you which energy step and the Energy Compression value tells you how many steps should have the same value. When you complete reading all of the data, you should have the matrix filled with the data included in the packet and the pad in the higher order sensor locations.

The last step is to look at the Time Compression. Possible values of the Time Compression are 0, 1, 2, 3, and 4. This corresponds to the number of sweeps that were measured in the sample and indicates the number of records that should be the same in the data file. A 0 means that there was 1 sweep taken. The **dr_time** value in the data field should be about 4 sec apart for each sweep. If Time Compression is 1, then you should duplicate the data record twice, but change **dr_time** in the second record to be a sweep after the original **dr_time**. Time Compression value of 2 indicates 4 sweeps were summed together, so duplicate values 4 times and increment **dr_time** for each record. Time Compression value of 3 indicates 8 sweeps were summed together, so duplicate values 8 times and increment **dr_time** for each record. Similarly, Time Compression value of 4 indicates 16 sweeps were summed together, so duplicate values 16 times and increment **dr_time** for each record.

For example, if the sector mask is [0,0,0,1,1,0,0,0,1,0,0,0,0,1,0,0], the Bit Compression is 0, the Energy Compression is 2, and the Time Compression is 1 at an SECT TIME of 2004 124 00:23:57.238, the matrix would be for 4 active anodes where the sector numbers are 2, 7, 11, and 12. The matrix would look like this:

info you got from | --> 16 locations for the possible 16 different sensors

engineering pkt | data from science pkt

| step | index | range | sensor 2 | sensor 7 | sensor 11 | sensor 12 | |
|------|-------|-------|----------|----------|-----------|-----------|--------------|
| 000 | 1 | fff | Byte 32 | Byte 34 | Byte 36 | Byte 38 | all are fill |
| 001 | 1 | eba | Byte 32 | Byte 34 | Byte 36 | Byte 38 | all are fill |
| 002 | 1 | d90 | Byte 32 | Byte 34 | Byte 36 | Byte 38 | all are fill |
| 003 | 1 | c7d | Byte 32 | Byte 34 | Byte 36 | Byte 38 | all are fill |
| 004 | 1 | b80 | Byte 40 | Byte 42 | Byte 44 | Byte 46 | all are fill |
| 005 | 1 | a97 | Byte 40 | Byte 42 | Byte 44 | Byte 46 | all are fill |
| 006 | 1 | 9c0 | Byte 40 | Byte 42 | Byte 44 | Byte 46 | all are fill |
| 007 | 1 | 8fb | Byte 40 | Byte 42 | Byte 44 | Byte 46 | all are fill |
| 008 | 1 | 845 | Byte 48 | Byte 50 | Byte 52 | Byte 54 | all are fill |
| 009 | 1 | 79d | Byte 48 | Byte 50 | Byte 52 | Byte 54 | all are fill |
| 010 | 1 | 703 | Byte 48 | Byte 50 | Byte 52 | Byte 54 | all are fill |
| 011 | 1 | 675 | Byte 48 | Byte 50 | Byte 52 | Byte 54 | all are fill |
| 012 | 1 | 5f2 | Byte 56 | Byte 58 | Byte 60 | Byte 62 | all are fill |
| 013 | 1 | 57a | Byte 56 | Byte 58 | Byte 60 | Byte 62 | all are fill |
| 014 | 1 | 50b | Byte 56 | Byte 58 | Byte 60 | Byte 62 | all are fill |
| 015 | 1 | 4a4 | Byte 56 | Byte 58 | Byte 60 | Byte 62 | all are fill |
| 016 | 1 | 446 | Byte 64 | Byte 66 | Byte 68 | Byte 70 | all are fill |
| 017 | 1 | 3f0 | Byte 64 | Byte 66 | Byte 68 | Byte 70 | all are fill |
| 018 | 1 | 3a0 | Byte 64 | Byte 66 | Byte 68 | Byte 70 | all are fill |
| 019 | 1 | 356 | Byte 64 | Byte 66 | Byte 68 | Byte 70 | all are fill |
| 020 | 1 | 313 | Byte 72 | Byte 74 | Byte 76 | Byte 78 | all are fill |
| 021 | 1 | 2d4 | Byte 72 | Byte 74 | Byte 76 | Byte 78 | all are fill |
| 022 | 1 | 29b | Byte 72 | Byte 74 | Byte 76 | Byte 78 | all are fill |
| 023 | 1 | 266 | Byte 72 | Byte 74 | Byte 76 | Byte 78 | all are fill |
| 024 | 1 | 236 | Byte 80 | Byte 82 | Byte 84 | Byte 86 | all are fill |
| 025 | 1 | 209 | Byte 80 | Byte 82 | Byte 84 | Byte 86 | all are fill |
| 026 | 1 | 1e0 | Byte 80 | Byte 82 | Byte 84 | Byte 86 | all are fill |
| 027 | 1 | 1ba | Byte 80 | Byte 82 | Byte 84 | Byte 86 | all are fill |
| 028 | 1 | 197 | Byte 88 | Byte 90 | Byte 92 | Byte 94 | all are fill |
| 029 | 1 | 176 | Byte 88 | Byte 90 | Byte 92 | Byte 94 | all are fill |
| 030 | 1 | 159 | Byte 88 | Byte 90 | Byte 92 | Byte 94 | all are fill |
| 031 | 1 | 13d | Byte 88 | Byte 90 | Byte 92 | Byte 94 | all are fill |
| 032 | 1 | 124 | Byte 96 | Byte 98 | Byte 100 | Byte 102 | all are fill |
| 033 | 1 | 10d | Byte 96 | Byte 98 | Byte 100 | Byte 102 | all are fill |
| 034 | 1 | f8 | Byte 96 | Byte 98 | Byte 100 | Byte 102 | all are fill |
| 035 | 1 | e4 | Byte 96 | Byte 98 | Byte 100 | Byte 102 | all are fill |
| 036 | 1 | d2 | Byte 104 | Byte 106 | Byte 108 | Byte 110 | all are fill |
| 037 | 1 | c1 | Byte 104 | Byte 106 | Byte 108 | Byte 110 | all are fill |
| 038 | 1 | b2 | Byte 104 | Byte 106 | Byte 108 | Byte 110 | all are fill |
| 039 | 1 | a4 | Byte 104 | Byte 106 | Byte 108 | Byte 110 | all are fill |
| 040 | 1 | 97 | Byte 112 | Byte 114 | Byte 116 | Byte 118 | all are fill |
| 041 | 1 | 8b | Byte 112 | Byte 114 | Byte 116 | Byte 118 | all are fill |
| 042 | 1 | 80 | Byte 112 | Byte 114 | Byte 116 | Byte 118 | all are fill |
| 043 | 1 | 76 | Byte 112 | Byte 114 | Byte 116 | Byte 118 | all are fill |
| 044 | 1 | 6c | Byte 120 | Byte 122 | Byte 124 | Byte 126 | all are fill |
| 045 | 1 | 64 | Byte 120 | Byte 122 | Byte 124 | Byte 126 | all are fill |
| 046 | 1 | 5c | Byte 120 | Byte 122 | Byte 124 | Byte 126 | all are fill |
| 047 | 1 | 54 | Byte 120 | Byte 122 | Byte 124 | Byte 126 | all are fill |
| 048 | 1 | 4e | Byte 128 | Byte 130 | Byte 132 | Byte 134 | all are fill |
| 049 | 1 | 48 | Byte 128 | Byte 130 | Byte 132 | Byte 134 | all are fill |

| | | | | | | | |
|-----|---|-----|----------|----------|----------|----------|--------------|
| 050 | 1 | 42 | Byte 128 | Byte 130 | Byte 132 | Byte 134 | all are fill |
| 051 | 1 | 3d | Byte 128 | Byte 130 | Byte 132 | Byte 134 | all are fill |
| 052 | 1 | 38 | Byte 136 | Byte 138 | Byte 140 | Byte 142 | all are fill |
| 053 | 1 | 33 | Byte 136 | Byte 138 | Byte 140 | Byte 142 | all are fill |
| 054 | 1 | 2f | Byte 136 | Byte 138 | Byte 140 | Byte 142 | all are fill |
| 055 | 1 | 2b | Byte 136 | Byte 138 | Byte 140 | Byte 142 | all are fill |
| 056 | 1 | 28 | Byte 144 | Byte 146 | Byte 148 | Byte 150 | all are fill |
| 057 | 1 | 25 | Byte 144 | Byte 146 | Byte 148 | Byte 150 | all are fill |
| 058 | 1 | 22 | Byte 144 | Byte 146 | Byte 148 | Byte 150 | all are fill |
| 059 | 1 | 1f | Byte 144 | Byte 146 | Byte 148 | Byte 150 | all are fill |
| 060 | 0 | f29 | Byte 152 | Byte 154 | Byte 156 | Byte 158 | all are fill |
| 061 | 0 | df6 | Byte 152 | Byte 154 | Byte 156 | Byte 158 | all are fill |
| 062 | 0 | cdb | Byte 152 | Byte 154 | Byte 156 | Byte 158 | all are fill |
| 063 | 0 | bd7 | Byte 152 | Byte 154 | Byte 156 | Byte 158 | all are fill |
| 064 | 0 | ae7 | Byte 160 | Byte 162 | Byte 164 | Byte 166 | all are fill |
| 065 | 0 | a0a | Byte 160 | Byte 162 | Byte 164 | Byte 166 | all are fill |
| 066 | 0 | 93e | Byte 160 | Byte 162 | Byte 164 | Byte 166 | all are fill |
| 067 | 0 | 883 | Byte 160 | Byte 162 | Byte 164 | Byte 166 | all are fill |
| 068 | 0 | 7d7 | Byte 168 | Byte 170 | Byte 172 | Byte 174 | all are fill |
| 069 | 0 | 738 | Byte 168 | Byte 170 | Byte 172 | Byte 174 | all are fill |
| 070 | 0 | 6a6 | Byte 168 | Byte 170 | Byte 172 | Byte 174 | all are fill |
| 071 | 0 | 61f | Byte 168 | Byte 170 | Byte 172 | Byte 174 | all are fill |
| 072 | 0 | 5a3 | Byte 176 | Byte 178 | Byte 180 | Byte 182 | all are fill |
| 073 | 0 | 531 | Byte 176 | Byte 178 | Byte 180 | Byte 182 | all are fill |
| 074 | 0 | 4c7 | Byte 176 | Byte 178 | Byte 180 | Byte 182 | all are fill |
| 075 | 0 | 467 | Byte 176 | Byte 178 | Byte 180 | Byte 182 | all are fill |
| 076 | 0 | 40d | Byte 184 | Byte 186 | Byte 188 | Byte 190 | all are fill |
| 077 | 0 | 3bb | Byte 184 | Byte 186 | Byte 188 | Byte 190 | all are fill |
| 078 | 0 | 370 | Byte 184 | Byte 186 | Byte 188 | Byte 190 | all are fill |
| 079 | 0 | 32a | Byte 184 | Byte 186 | Byte 188 | Byte 190 | all are fill |
| 080 | 0 | 2ea | Byte 192 | Byte 194 | Byte 196 | Byte 198 | all are fill |
| 081 | 0 | 2af | Byte 192 | Byte 194 | Byte 196 | Byte 198 | all are fill |
| 082 | 0 | 278 | Byte 192 | Byte 194 | Byte 196 | Byte 198 | all are fill |
| 083 | 0 | 246 | Byte 192 | Byte 194 | Byte 196 | Byte 198 | all are fill |
| 084 | 0 | 218 | Byte 200 | Byte 202 | Byte 204 | Byte 206 | all are fill |
| 085 | 0 | 1ee | Byte 200 | Byte 202 | Byte 204 | Byte 206 | all are fill |
| 086 | 0 | 1c7 | Byte 200 | Byte 202 | Byte 204 | Byte 206 | all are fill |
| 087 | 0 | 1a3 | Byte 200 | Byte 202 | Byte 204 | Byte 206 | all are fill |
| 088 | 0 | 181 | Byte 208 | Byte 210 | Byte 212 | Byte 214 | all are fill |
| 089 | 0 | 163 | Byte 208 | Byte 210 | Byte 212 | Byte 214 | all are fill |
| 090 | 0 | 147 | Byte 208 | Byte 210 | Byte 212 | Byte 214 | all are fill |
| 091 | 0 | 12d | Byte 208 | Byte 210 | Byte 212 | Byte 214 | all are fill |
| 092 | 0 | 115 | Byte 216 | Byte 218 | Byte 220 | Byte 222 | all are fill |
| 093 | 0 | ff | Byte 216 | Byte 218 | Byte 220 | Byte 222 | all are fill |
| 094 | 0 | eb | Byte 216 | Byte 218 | Byte 220 | Byte 222 | all are fill |
| 095 | 0 | d8 | Byte 216 | Byte 218 | Byte 220 | Byte 222 | all are fill |
| 096 | 0 | c7 | Byte 224 | Byte 226 | Byte 228 | Byte 230 | all are fill |
| 097 | 0 | b7 | Byte 224 | Byte 226 | Byte 228 | Byte 230 | all are fill |
| 098 | 0 | a9 | Byte 224 | Byte 226 | Byte 228 | Byte 230 | all are fill |
| 099 | 0 | 9b | Byte 224 | Byte 226 | Byte 228 | Byte 230 | all are fill |
| 100 | 0 | 8f | Byte 232 | Byte 234 | Byte 236 | Byte 238 | all are fill |
| 101 | 0 | 84 | Byte 232 | Byte 234 | Byte 236 | Byte 238 | all are fill |
| 102 | 0 | 79 | Byte 232 | Byte 234 | Byte 236 | Byte 238 | all are fill |
| 103 | 0 | 70 | Byte 232 | Byte 234 | Byte 236 | Byte 238 | all are fill |
| 104 | 0 | 67 | Byte 240 | Byte 242 | Byte 244 | Byte 246 | all are fill |

| | | | | | | | |
|-----|---|-----|----------|----------|----------|----------|--------------|
| 105 | 0 | 5f | Byte 240 | Byte 242 | Byte 244 | Byte 246 | all are fill |
| 106 | 0 | 57 | Byte 240 | Byte 242 | Byte 244 | Byte 246 | all are fill |
| 107 | 0 | 50 | Byte 240 | Byte 242 | Byte 244 | Byte 246 | all are fill |
| 108 | 0 | 4a | Byte 248 | Byte 250 | Byte 252 | Byte 254 | all are fill |
| 109 | 0 | 44 | Byte 248 | Byte 250 | Byte 252 | Byte 254 | all are fill |
| 110 | 0 | 3e | Byte 248 | Byte 250 | Byte 252 | Byte 254 | all are fill |
| 111 | 0 | 39 | Byte 248 | Byte 250 | Byte 252 | Byte 254 | all are fill |
| 112 | 0 | 35 | Byte 256 | Byte 258 | Byte 260 | Byte 262 | all are fill |
| 113 | 0 | 31 | Byte 256 | Byte 258 | Byte 260 | Byte 262 | all are fill |
| 114 | 0 | 2d | Byte 256 | Byte 258 | Byte 260 | Byte 262 | all are fill |
| 115 | 0 | 29 | Byte 256 | Byte 258 | Byte 260 | Byte 262 | all are fill |
| 116 | 0 | 26 | Byte 264 | Byte 266 | Byte 268 | Byte 270 | all are fill |
| 117 | 0 | 23 | Byte 264 | Byte 266 | Byte 268 | Byte 270 | all are fill |
| 118 | 0 | 20 | Byte 264 | Byte 266 | Byte 268 | Byte 270 | all are fill |
| 119 | 0 | 1d | Byte 264 | Byte 266 | Byte 268 | Byte 270 | all are fill |
| 120 | 0 | 1b | Byte 272 | Byte 274 | Byte 276 | Byte 278 | all are fill |
| 121 | 0 | 19 | Byte 272 | Byte 274 | Byte 276 | Byte 278 | all are fill |
| 122 | 0 | 17 | Byte 272 | Byte 274 | Byte 276 | Byte 278 | all are fill |
| 123 | 0 | 15 | Byte 272 | Byte 274 | Byte 276 | Byte 278 | all are fill |
| 124 | 0 | 13 | Byte 280 | Byte 282 | Byte 284 | Byte 286 | all are fill |
| 125 | 0 | 12 | Byte 280 | Byte 282 | Byte 284 | Byte 286 | all are fill |
| 126 | 0 | 10 | Byte 280 | Byte 282 | Byte 284 | Byte 286 | all are fill |
| 127 | 1 | fff | Byte 280 | Byte 282 | Byte 284 | Byte 286 | all are fill |

For this example, Byte 248 means Byte 248 and Byte 249. Since Bit Compression was not on, the word length in the ELS science packet is already 16 bit, so two bytes make up the word. Due to space, I have indicated the offset of just the first of two bytes.

For this example, steps 0 through 59 are written as high range data and bytes 60 through 123 are written as low range data. Bytes 124 through 127 are ignored. When the data is written into the science packet, the last energy compressed step written is a flyback step. This flyback step should not be included within **ELSSCIH** / **ELSSCIL**. In the above example, this is steps 124, 125, 126, and 127.

For this example, this matrix is written into the data record with a **dr_time** stamp of $00 * 3600000 + 23 * 60000 + 57238 = 1437238$. Then, since the Time Compression bit is 1, a second data record is written identical to the first except that **dr_time** is $1437238 + 4000 = 1441238$.