

SU-IDD-15-03561-V3.2

February 28, 2005

**SCANNER IDFS Design Document
Version 3.2**

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

SU-IDD-15-03561-V3.2

February 28, 2005

SCANNER IDFS DESIGN DOCUMENT

Version 3.2

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 | 08/23/02 | <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. |
| Version 3.0 | 10/24/03 | <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. |
| Version 3.1 | 12/15/03 | <ul style="list-style-type: none"> • Modified tolerance values for -12V Power Line in Section 7.1. |
| Version 3.2 | 02/28/05 | <ul style="list-style-type: none"> • Added equations in Section 6.1 for unit conversion of currents and temperature. • Updated Appendix A for VIDF changes. |

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 Scanner Engineering and Housekeeping Data..... | 4 |
| 3.0 | Notes for Scanner Data..... | 6 |
| 4.0 | Introduction to IDFS | 7 |
| 5.0 | Breakdown of Scanner Data into Virtual Instruments | 10 |
| 6.0 | Definition for Virtual Instrument SCANENG8..... | 11 |
| 6.1 | Notes for SCANENG8 virtual instrument..... | 13 |
| 6.1.1 | Scanner Temperature Monitor..... | 13 |
| 6.1.2 | Scanner Threshold CW Reference and Scanner Threshold CCW Reference..... | 13 |
| 6.1.3 | Scanner Coast Current Reference and Scanner Ramp Current Reference | 13 |
| 6.2 | SCANENG8 Header Record Format..... | 14 |
| 6.2.1 | Setting of <i>data_lat</i> header record element..... | 14 |
| 6.2.2 | Setting of <i>n_sample</i> header record element..... | 14 |
| 6.2.3 | Setting of <i>scan_index</i> header record element | 15 |
| 6.2.4 | Setting of <i>sensor_index</i> header record element | 15 |
| 6.2.5 | Setting of <i>d_qual</i> header record element..... | 15 |
| 6.2.6 | Setting of <i>mode_index</i> header record element..... | 16 |
| 6.3 | SCANENG8 Data Record Format..... | 17 |
| 6.3.1 | Setting of <i>dr_time</i> data record element..... | 17 |
| 6.3.2 | Setting of <i>spin</i> data record element | 17 |
| 6.3.3 | Setting of <i>sun_sen</i> data record element | 18 |
| 6.3.4 | Setting of <i>data_array</i> data record element | 19 |
| 7.0 | Definition for Virtual Instrument SCANENGS..... | 21 |
| 7.1 | Notes for SCANENGS virtual instrument..... | 23 |
| 7.2 | SCANENGS Header Record Format | 24 |
| 7.2.1 | Setting of <i>data_lat</i> header record element..... | 24 |
| 7.2.2 | Setting of <i>n_sample</i> header record element..... | 24 |
| 7.2.3 | Setting of <i>scan_index</i> header record element | 25 |
| 7.2.4 | Setting of <i>sensor_index</i> header record element | 25 |
| 7.2.5 | Setting of <i>d_qual</i> header record element..... | 25 |
| 7.2.6 | Setting of <i>mode_index</i> header record element..... | 26 |
| 7.3 | SCANENGS Data Record Format | 28 |
| 7.3.1 | Setting of <i>dr_time</i> data record element..... | 28 |
| 7.3.2 | Setting of <i>spin</i> data record element | 28 |
| 7.3.3 | Setting of <i>sun_sen</i> data record element | 29 |
| 7.3.4 | Setting of <i>data_array</i> data record element | 30 |
| 8.0 | Definition for Virtual Instrument SCANPOS..... | 32 |

| | | |
|--|---|-------------|
| 8.1 | Notes for SCANPOS virtual instrument..... | 33 |
| 8.2 | SCANPOS Header Record Format..... | 34 |
| 8.2.1 | Setting of <i>data_accum</i> header record element..... | 34 |
| 8.2.2 | Setting of <i>n_sample</i> header record element..... | 35 |
| 8.2.3 | Setting of <i>scan_index</i> header record element..... | 35 |
| 8.2.4 | Setting of <i>sensor_index</i> header record element..... | 35 |
| 8.2.5 | Setting of <i>d_qual</i> header record element..... | 35 |
| 8.2.6 | Setting of <i>mode_index</i> header record element..... | 36 |
| 8.3 | SCANPOS Data Record Format..... | 37 |
| 8.3.1 | Setting of <i>dr_time</i> data record element..... | 37 |
| 8.3.2 | Setting of <i>spin</i> data record element..... | 37 |
| 8.3.3 | Setting of <i>sun_sen</i> data record element..... | 38 |
| 8.3.4 | Setting of <i>nss</i> data record element..... | 39 |
| 8.3.5 | Setting of <i>data_array</i> data record element..... | 40 |
| Appendix A - Virtual Instrument Description Files (VIDF) for Scanner..... | | A-1 |
| SCANENG8 VIDF File..... | | A-1 |
| SCANENG5 VIDF File..... | | A-6 |
| SCANPOS VIDF File..... | | A-11 |
| Appendix B - Scanner Instrument Operation Scenarios..... | | B-1 |

LIST OF FIGURES

| | | |
|-----------------|---|----|
| <i>Figure 1</i> | <i>Bit Designation</i> | 4 |
| <i>Figure 2</i> | <i>IDFS sensor data array</i> | 9 |
| <i>Figure 3</i> | <i>IDFS calibration data array</i> | 9 |
| <i>Figure 4</i> | <i>Data Array for SCANENGS</i> | 31 |
| <i>Figure 5</i> | <i>Data Array for SCANPOS when nss equals 1</i> | 40 |
| <i>Figure 6</i> | <i>Data Array for SCANPOS when nss equals 2</i> | 40 |
| <i>Figure 7</i> | <i>Data Array for SCANPOS when nss equals 4</i> | 41 |

LIST OF TABLES

| | |
|---|-----------|
| <i>Table 2-1 Scanner Source Data Packets.....</i> | <i>5</i> |
| <i>Table 6-1 SCANENG8 Sensor Definition</i> | <i>11</i> |
| <i>Table 6-2 SCANENG8 Status Flag Definition.....</i> | <i>12</i> |
| <i>Table 6-3 SCANENG8 Header Record Definition.....</i> | <i>14</i> |
| <i>Table 6-4 SCANENG8 Data Record Definition.....</i> | <i>17</i> |
| <i>Table 7-1 SCANENG8 Sensor Definition</i> | <i>21</i> |
| <i>Table 7-2 SCANENG8 Status Flag Definition.....</i> | <i>22</i> |
| <i>Table 7-3 SCANENG8 Header Record Definition.....</i> | <i>24</i> |
| <i>Table 7-4 SCANENG8 Data Record Definition.....</i> | <i>28</i> |
| <i>Table 8-1 SCANPOS Sensor Definition.....</i> | <i>32</i> |
| <i>Table 8-2 SCANPOS Status Flag Definition.....</i> | <i>32</i> |
| <i>Table 8-3 SCANPOS Header Record Definition</i> | <i>34</i> |
| <i>Table 8-4 SCANPOS Data Record Definition</i> | <i>37</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) |
| IRQ | Interrupt Request |
| 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 IDFS production code for the Scanning Unit, hereafter referred to as the Scanner, will read the Scanner-specific and Main Unit-specific data files generated by the APAF in order to create 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 Scanner 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 Scanner-specific IDFS production code (Section 2.0).
- 2) Provide some background information explaining how operational characteristics of the Main Unit 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 Scanner, with a brief explanation of the decision process utilized to break up the Scanner data into the various streams (Section 5.0).
- 5) Provide a section for each of the IDFS virtual instruments defined for Scanner, with each section containing information that describes (Sections 6.0 – 8.0):
 - 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 Scanner 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 Scanner 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 Scanner Engineering 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 Scanner data. These two packets are referred to throughout this document as the Scanner Information Packet and the Main Unit Housekeeping Packet, respectively. The Main Unit Housekeeping Packet contains housekeeping data not only for Scanner, but also for NPI, NPD, ELS, IMA and for the main unit itself.

Table 2-1 summarizes the information contained in the Scanner Information 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 Scanner Information packet and a Main Unit 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:

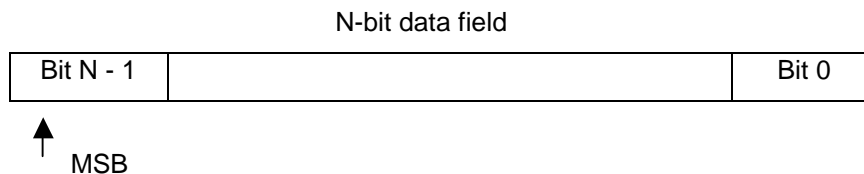


Figure 1 Bit Designation

The overall structure of the packet is the same for the Scanner Information 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 Scanner 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 Scanner Information Data or the Main Unit Housekeeping Data |

3.0 Notes for Scanner Data

The reader of this document is referred to Appendix B for a detailed explanation of operational scenarios for the Scanner.

It is imperative that an absolute time tag be incorporated into the packets so that correlative science analysis can be performed properly and accurately for all ASPERA-3 instruments.

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 Scanner 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 Scanner Data into Virtual Instruments

The Scanner data will be broken down into the following three virtual instruments:

1. **SCANENG8** - Scanner 8-Bit Engineering Monitor Data
2. **SCANENG8S** - Scanner Engineering Status Data
3. **SCANPOS** – Scanner Position / Orientation Data

The following sections will describe the data that comprises each of the three virtual instruments defined for the Scanner data. The data was subdivided into the three virtual instruments based upon data category (Engineering Data vs. Science Data) and word size.

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 **SCANENG8**

The Scanner-specific 8-bit monitor and reference values returned in the Main Unit Housekeeping packet form the virtual instrument **SCANENG8**. The first four characters in the acronym are intended to identify the parent instrument (Scanner), 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.

Table 6-1 SCANENG8 Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Packet Field Name | Packet Byte Position |
|---------------------------|-----------------------------------|----------------------|----------------------------|-----------------------------|
| 0 | Scanner Temperature Monitor | HSKP | scanner_temp_sensor | 22 |
| 1 | Scanner VREFMC | HSKP | scanner_vrefmc | 97 |
| 2 | Scanner Coast Current Reference | HSKP | scanner_coast_current_ref | 100 |
| 3 | Scanner Ramp Current Reference | HSKP | scanner_ramp_current_ref | 101 |
| 4 | Scanner Threshold CW Reference | HSKP | scanner_treshold_cw_ref | 102 |
| 5 | Scanner Threshold CCW Reference | HSKP | scanner_treshold_ccw_ref | 103 |
| 6 | Scanner Threshold Wheel Reference | HSKP | scanner_treshold_wheel_ref | 104 |
| 7 | Scanner Position | HSKP | scanner_position | 105 |

Table 6-2 SCANENG8 Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Packet Field Name | Packet Byte Position | Packet Bit Position |
|--------------------|-------------------------|----------------------|--------------------------|-----------------------------|----------------------------|
| 0 | SW Version – Upper Byte | HSKP | sw_version | 24 | 0-7 |
| 1 | SW Version – Lower Byte | HSKP | sw_version | 25 | 0-7 |
| 2 | Software Mode | HSKP | sw_mode | 106 | 0-7 |
| 3 | Scanner Speed | HSKP | scanner_speed | 99 | 0-1 |

6.1 Notes for SCANENG8 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 Scanner Temperature Monitor

The Scanner Temperature Monitor is represented two different ways: by an 8-bit telemetry number and by the temperature generated. The Scanner Temperature Monitor is converted to a temperature through the formula:

$$\text{Temperature Monitor [Deg.C]} = 1.5686 * \text{BMON} - 263.3098$$

where BMON is the value returned in telemetry and ranges from 0 to 255.

6.1.2 Scanner Threshold CW Reference and Scanner Threshold CCW Reference

The Scanner Threshold CW Reference and the Scanner Threshold CCW Reference values are both represented two different ways: by an 8-bit telemetry number and by the reference value generated. The Scanner Threshold CW Reference and the Scanner Threshold CCW Reference bytes are converted to a reference value through the formula:

$$\text{Tresh} = (\text{BMON} / 255) * 5\text{V}$$

where BMON is the value returned in telemetry and ranges from 0 to 255.

6.1.3 Scanner Coast Current Reference and Scanner Ramp Current Reference

The Scanner Coast Current Reference and Scanner Ramp Current Reference values are both represented two different ways: by an 8-bit telemetry number and by the current generated. The Scanner Coast Current Reference and Scanner Ramp Current Reference bytes are converted to a current value through the formula:

$$\text{I28} = 0.4207 * \ln(\text{BMON}) - 1.7492$$

where I28 represents the current in the +28V power line entering the instrument and BMON is the value returned in telemetry, ranging from 0 to 255.

6.2 SCANENG8 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 SCANENG8 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 | 4 |
| 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 | 8 |
| 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 *data_lat* header record element

There is a telecommand that can be used to set the Main Unit Housekeeping packet generation frequency. There is a note which states that the delay that is specified is defined in terms of seconds. However, this rate defines how fast the packets can be generated, not the rate at which the data was acquired. Therefore, the value of 31250 microseconds reflects the fastest instrument cycle time of all instruments that return data in the Main Unit Housekeeping packet.

6.2.2 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.

6.2.3 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
```

6.2.4 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 8 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
sensor_index [6]    6
sensor_index [7]    7
```

6.2.5 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 or +5V monitor value is not within tolerance |
| 3 | Bad Data | +30V monitor value and (+12V or +5V monitor value) are not within tolerance |

Since all IDFS sensors will be returned per data record, **n_sen** is set to 8 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 |
| d_qual [6] | 0 |
| d_qual [7] | 0 |

6.2.6 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 bytes 2 and 3, 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 bytes 2 and 3:

| Status Byte | 2 Software Mode | 3 Scanner Speed |
|--------------------------|--|--|
| State Definitions | 0 = Undefined 1 = Booting 2 = Safe 3 = Prom 4 = Normal | 0 = Stopped 1 = 32 second scan 2 = 64 second scan 3 = 128 second scan |

6.3 SCANENG8 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 SCANENG8 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 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.

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 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 |

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 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.

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 **SCANENG8** should be laid down according to the layout provided below:

| Array Position | IDFS Sensor | Packet Source | Packet Byte Position | Comments |
|----------------|-------------|---------------|----------------------|----------|
| data_array [4] | 0 | HSKP | 22 | 1 value |
| data_array [5] | 1 | HSKP | 97 | 1 value |

| Array Position | IDFS Sensor | Packet Source | Packet Byte Position | Comments |
|-----------------------|--------------------|----------------------|-----------------------------|-----------------|
| data_array [6] | 2 | HSKP | 100 | 1 value |
| data_array [7] | 3 | HSKP | 101 | 1 value |
| data_array [8] | 4 | HSKP | 102 | 1 value |
| data_array [9] | 5 | HSKP | 103 | 1 value |
| data_array [10] | 6 | HSKP | 104 | 1 value |
| data_array [11] | 7 | HSKP | 105 | 1 value |

7.0 Definition for Virtual Instrument SCANENGS

The Scanner-specific instrument status monitors returned in the Main Unit Housekeeping packet form the virtual instrument **SCANENGS**. The first four characters in the acronym are intended to identify the parent instrument (Scanner), 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 SCANENGS Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Packet Field Name | Packet Byte Position | Packet Bit Position |
|---------------------------|---------------------------------|----------------------|----------------------------|-----------------------------|----------------------------|
| 0 | Scanner CCW End Position Status | HSKP | scanner_status_ccw_end_pos | 98 | 7 |
| 1 | Scanner CW End Position Status | HSKP | scanner_status_cw_end_pos | 98 | 6 |
| 2 | Scanner Position Clock Status | HSKP | scanner_status_pos_clock | 98 | 5 |
| 3 | Scanner Direction Status | HSKP | scanner_status_direction | 98 | 4 |
| 4 | Scanner State | HSKP | scanner_status_state | 98 | 2-3 |
| 5 | Scanner Lost Step | HSKP | Lost step | 98 | 1 |
| 6 | Scanner Initialized | HSKP | scanner_initialized | 98 | 0 |
| 7 | Scanner +30V Status | HSKP | scanner_plus_30v_on_off | 99 | 7 |
| 8 | Scanner Setup Mode | HSKP | scanner_setup_mode | 99 | 4 |
| 9 | Scanner Setup Direction | HSKP | scanner_setup_direction | 99 | 3 |
| 10 | Scanner Speed | HSKP | scanner_speed | 99 | 0-1 |
| 11 | +30V Enable | HSKP | hk_v_plus_30v | 35 | n/a |
| 12 | -12V Enable | HSKP | hk_v_minus_12v | 37 | n/a |
| 13 | +12V Enable | HSKP | hk_v_plus_12v | 34 | n/a |
| 14 | -5V Enable | HSKP | hk_v_minus_5v | 38 | n/a |
| 15 | +5V Enable | HSKP | hk_v_plus_5v | 36 | n/a |

Table 7-2 SCANENGS Status Flag Definition

| Status Byte | Status Byte Name | Packet Source | Packet Field Name | Packet Byte Position | Packet Bit Position |
|--------------------|-------------------------|----------------------|--------------------------|-----------------------------|----------------------------|
| 0 | SW Version – Upper Byte | HSKP | sw_version | 24 | 0-7 |
| 1 | SW Version – Lower Byte | HSKP | sw_version | 25 | 0-7 |
| 2 | Software Mode | HSKP | sw_mode | 106 | 0-7 |
| 0 | Scanner Speed | HSKP | scanner_speed | 99 | 0-1 |

The **SCANENGS** virtual instrument is defined to hold status values that are of interest to aid in the understanding of the current operational status of the Scanner. However, only eleven of the IDFS sensors defined above (sensors 0 through 10) are true status bits being returned in telemetry. The other IDFS sensors (sensors 11 through 15) must determine their single-bit status 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 SCANENGS 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 7.3.

The **SCANENGS** virtual instrument is defined to hold status values that are of interest to aid in the understanding of the current operational status of the Scanner. However, only eleven of the IDFS sensors defined above (sensors 0 through 10) are true status bits being returned in telemetry. The other IDFS sensors (sensors 11 through 15) must determine their single-bit status values based upon the 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 11 through 15:

| IDFS Sensor Name | Telemetry Monitor | Determination for Bit Setting |
|------------------|-------------------|--|
| +30V Enable | hk_v_plus_30v | if $+29.96V \leq \text{monitor} \leq +30.04V$ 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 SCANENGS 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 SCANENGS 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 | 4 |
| 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 | 16 |
| 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 |

7.2.1 Setting of *data_lat* header record element

There is a telecommand that can be used to set the Main Unit Housekeeping packet generation frequency. There is a note which states that the delay that is specified is defined in terms of seconds. However, this rate defines how fast the packets can be generated, not the rate at which the data was acquired. Therefore, the value of 31250 microseconds reflects the fastest instrument cycle time of all instruments that return data in the Main Unit Housekeeping packet.

7.2.2 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.

7.2.3 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
```

7.2.4 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 16 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
sensor_index [6]    6
sensor_index [7]    7
sensor_index [8]    8
sensor_index [9]    9
sensor_index [10]   10
sensor_index [11]   11
sensor_index [12]   12
sensor_index [13]   13
sensor_index [14]   14
sensor_index [15]   15
```

7.2.5 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 or +5V monitor value is not within tolerance |
| 3 | Bad Data | +30V monitor value and (+12V or +5V monitor value) are not within tolerance |

Since all IDFS sensors will be returned per data record, **n_sen** is set to 16 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
d_qual [6]      0
d_qual [7]      0
d_qual [8]      0
d_qual [9]      0
d_qual [10]     0
d_qual [11]     0
d_qual [12]     0
d_qual [13]     0
d_qual [14]     0
d_qual [15]     0

```

7.2.6 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 bytes 2 and 3, 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 bytes 2 and 3:

| Status Byte | 2 Software Mode | 3 Scanner Speed |
|--------------------------|--|--|
| State Definitions | 0 = Undefined 1 = Booting 2 = Safe 3 = Prom 4 = Normal | 0 = Stopped 1 = 32 second scan 2 = 64 second scan 3 = 128 second scan |

7.3 SCANENGS 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 SCANENGS 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

SCANENGS basically consists of single-bit and multi-bit quantities that are packed together into an 8-bit word before being placed into the **data_array**. Since IDFS sensors 4 and 10 are 2-bit quantities, all single-bit quantities must be written as a 2-bit quantity. Therefore, 4 IDFS sensors are packed into a single 8-bit word.

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 **SCANENGS** is defined accordingly:

| Array Position | IDFS Sensor | Comments |
|----------------|-------------|--------------------|
| data_array [4] | 0 – 3 | 1 value per sensor |
| data_array [5] | 4 – 7 | 1 value per sensor |
| data_array [6] | 8 – 11 | 1 value per sensor |
| data_array [7] | 12 – 15 | 1 value per sensor |

Figure 4 defines how the 4 IDFS sensors are packed within each 8-bit value.

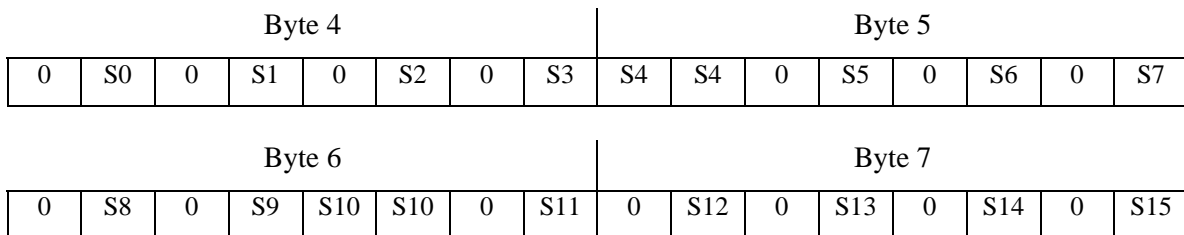


Figure 4 Data Array for SCANENGS

Since the sensor values will be defined as 2-bit quantities in the VIDF file, it is imperative that the top bit of single-bit telemetry values be zero-filled.

8.0 Definition for Virtual Instrument SCANPOS

The data that is returned in the Scanner Information Packet form the virtual instrument **SCANPOS**. The first four characters in the acronym are intended to identify the parent instrument (Scanner) and the next three characters (POS) indicate that the data defines the Scanner position / orientation.

The IDFS sensor definitions for the virtual instrument are defined in Table 8-1 below. The status flag definitions for the virtual instrument are defined in Table 8-2. 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 Scanner Information Packet, the value for the **Data type** parameter is 5 and the packets are referred to as Engineering telemetry packets.

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

Table 8-1 SCANPOS Sensor Definition

| IDFS Sensor Number | Definition | Packet Source | Data Type | Data Subtype | Packet Field Name | Packet Byte Position |
|---------------------------|-------------------|----------------------|------------------|---------------------|--------------------------|-----------------------------|
| 0 | Scanner Position | SCI | 5 | 1 | Data | 28 |

Table 8-2 SCANPOS 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 | 5 | 1 | SW version | 16 | 0-7 |
| 1 | SW Version – Lower Byte | SCI | 5 | 1 | SW version | 17 | 0-7 |
| 2 | Software Mode | HSKP | - | - | sw_mode | 106 | 0-7 |
| 3 | Scanner Speed | HSKP | - | - | scanner_speed | 99 | 0-1 |

8.1 Notes for SCANPOS 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.

When the scanner is parked, Scanner Information packets should not be generated.

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 Science packets. The parameter **RICE compression enabled** (byte 27, bit 0) 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.

The number of positions reported in the Scanner Information packet is based upon the scanner speed. When the scanner is operating in the 32 second mode, a single Scanner Information packet is generated and the packet contains 1024 positions. When the scanner is operating in the 64 second mode, a single Scanner Information packet is generated and the packet contains 2048 positions. When the scanner is operating in the 128 second mode, two Scanner Information packets are generated, with each packet containing 2048 positions, resulting in data being reported for 4096 total positions.

8.2 SCANPOS 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 8-3 SCANPOS 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 | -3 |
| i_mode | no. of status flags returned in mode_index | 4 |
| data_accum | sample accumulation time | DATA |
| data_lat | sample dead time (μ sec) | 0 |
| 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 | 1 |
| 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 *data_accum* header record element

Scanner information is returned once per scanner cycle. Therefore, the accumulation rate depends upon the scanner speed, which is reported in the **Scanner speed** parameter (byte 27, bits 4-5) in the Scanner Information packet. Based upon scanner speed, the following values should be used:

| Scanner Speed Value | Speed (milliseconds) | Data_accum (milliseconds) |
|---------------------|----------------------|---------------------------|
| 1 | 32000 | 32000 / 1024 |
| 2 | 64000 | 64000 / 2048 |
| 3 | 128000 | 128000 / 4096 |

The times that are shown in the table above are in milliseconds since the value for **time_units** indicates that the accumulation time is being expressed in terms of milliseconds. In all cases listed, **data_accum** equates to 31.25 milliseconds. Notice that there is no entry for when the scanner_speed value is 0. That is because when the scanner is parked, Scanner Information packets should not be generated.

8.2.2 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, the number of measurements that are stacked is dependent upon the scanner speed. When the scanner is operating in the 32 second mode, 1024 positions are packed into a single sensor set and **n_sample** should be set to 1024. When the scanner is operating in the 64 second mode, 2048 positions are packed into a single sensor set and **n_sample** should be set to 2048. When the scanner is operating in the 128 second mode, 4096 positions are packed into a single sensor set and **n_sample** should be set to 4096.

8.2.3 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
```

8.2.4 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 1 and the values for **sensor_index** should be set accordingly:

```
sensor_index [0]    0
```

8.2.5 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 or +5V monitor value is not within tolerance |
| 3 | Bad Data | +30V monitor value and (+12V or +5V monitor value) are not within tolerance |

| Value | Meaning | Criteria for Setting of Value |
|-------|---------------|---|
| 4 | Unknown State | Time related Main Unit Housekeeping packet is not available |

Since all IDFS sensors will be returned per data record, **n_sen** is set to 1 and the values for **d_qual** should be set accordingly under nominal conditions:

d_qual [0] 0

8.2.6 Setting of *mode_index* header record element

Mode_index is an array of **i_mode** values. Table 8-2 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 bytes 2 and 3, 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 bytes 2 and 3:

| Status Byte | 2 Software Mode | 3 Scanner Speed |
|--------------------------|--|--|
| State Definitions | 0 = Undefined 1 = Booting 2 = Safe 3 = Prom 4 = Normal | 0 = Stopped 1 = 32 second scan 2 = 64 second scan 3 = 128 second scan |

8.3 SCANPOS 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 8-4 SCANPOS 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 | DATA |
| 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 Scanner Information 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 27, bits 4-5 in the Scanner Information 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 27, bit 6 in the Scanner Information 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 26 in the Scanner Information 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

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

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

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

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

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

from the SCET time.

8.3.4 Setting of *nss* data record element

For this virtual instrument, the number of measurements that are stacked in a single sensor set is dependent upon the scanner speed. The number of measurements can vary between 1024, 2048 and 4096. Since the data records are fixed-size records, the usage of multiple sensor sets within a data record is being defined to avoid padding out empty space within **data_array**.

When the scanner is operating in the 32 second mode, 1024 positions are packed into a single sensor set. Four sensor sets should be defined per data record and **nss** should be set to 4. When the scanner is operating in the 64 second mode, 2048 positions are packed into a single sensor set. Two sensor sets should be defined per data record and **nss** should be set to 2. When the scanner is operating in the 128 second mode, 4096 positions are packed into a single sensor set. One sensor set should be defined per data record and **nss** should be set to 1. By utilizing this approach, all data records will contain 4096 positions. ($4 * 1024 = 4096$, $2 * 2048 = 4096$, $1 * 4096 = 4096$).

8.3.5 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 **SCANPOS** is dependent upon the number of sensor sets that are defined per data record, which is specified by **nss**. When **nss** is set to one, the **data_array** for **SCANPOS** should be laid down according to the layout provided in Figure 5:

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Comments |
|-----------------------------|-------------|---------------|-----------|--------------|----------------------|----------|
| data_array [4 thru (N-1)+4] | 0 | SCI | 5 | 1 | 28 | N values |

Figure 5 Data Array for SCANPOS when nss equals 1

When **nss** is set to two, the **data_array** for **SCANPOS** should be laid down according to the layout provided in Figure 6:

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Comments |
|----------------------------------|-------------|---------------|-----------|--------------|----------------------|----------|
| data_array [4 thru (N-1)+4] | 0 | SCI | 5 | 1 | 28 | N values |
| data_array [(N+4) thru (2N-1)+4] | 0 | SCI | 5 | 1 | 28 | N values |

Figure 6 Data Array for SCANPOS when nss equals 2

When **nss** is set to four, the **data_array** for **SCANPOS** should be laid down according to the layout provided in Figure 7:

| Array Position | IDFS Sensor | Packet Source | Data Type | Data Subtype | Packet Byte Position | Comments |
|-----------------------------------|-------------|---------------|-----------|--------------|----------------------|----------|
| data_array [4 thru (N-1)+4] | 0 | SCI | 5 | 1 | 28 | N values |
| data_array [(N+4) thru (2N-1)+4] | 0 | SCI | 5 | 1 | 28 | N values |
| data_array [(2N+4) thru (3N-1)+4] | 0 | SCI | 5 | 1 | 28 | N values |
| data_array [(3N+4) thru (4N-1)+4] | 0 | SCI | 5 | 1 | 28 | N values |

Figure 7 Data Array for SCANPOS when nss equals 4

In all 3 tables shown above, the variable **N** represents the value **n_sample**, as defined in the header record.

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

SCANENG8 VIDF File

```

vidf v3_SCANENG8 {
    float version = 3.0;           /* version      */
    string mission = "MARS";       /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft   */
    string experiment = "ASPERA-3"; /* exp_desc    */
    string instrument = "SCANNER"; /* 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 the Scanner that are returned in the
* Main Unit Housekeeping Packet.
*
* 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: Factor which converts sensor index into fraction of
*          voltage range.
* TABLE 2: Maximum control input voltage [volts]. (final unit for
*          threshold sensors 4 and 5)
* TABLE 3: Convert Temperature Value to degrees C, current reference
*          sensors to milliamps.
* TABLE 4: ASCII definitions of status states
*
* Data_len is set using the equation
* 20 + 8 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 = 4;           /* n_qual       */
    int n_cal_sets = 0;      /* cal_sets     */
    int n_tbls = 5;          /* num_tbls     */
    int n_consts = 0;        /* num_consts   */
    int n_status = 4;        /* status       */
    int n_sensors = 8;       /* sen          */
    int swp_len = 1;         /* swp_len      */

```

```

int max_nss = 1; /* max_nss */
int data_len = 32; /* 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 */
};
struct Status3 {
    string name = "Scanner Speed"; /* name */
    int state = 4; /* 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 = "Scanner 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 Sensor1 {
    string name = "Scanner VREFMC"; /* 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 = "Scanner Coast Current 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 = "Scanner Ramp Current 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 Sensor4 {
    string name = "Scanner Threshold CW 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 Sensor5 {
    string name = "Scanner Threshold CCW 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 Sensor6 {
    string name = "Scanner Threshold Wheel 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 Sensor7 {
    string name = "Scanner Position"; /* 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 = 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 [8] = {              /* format      */
        2, 2, 2, 2, 2,              /* 000 - 004  */
        2, 2, 2                    /* 005 - 007  */
    };
    int offset [8] = {              /* offsets     */
        0, 0, 0, 0, 0,              /* 000 - 004  */
        0, 0, 0                    /* 005 - 007  */
    };
    int scale [2] = {0, 0};         /* scale factor */
    int values [2] = {0, 1};        /* values      */
};
struct Table1 {
    int tbl_sca_sz = 1;             /* tbl_sca_sz  */
    int tbl_ele_sz = 1;             /* 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.
*/
    int tbl_var = 1; /* tbl_var */
    int tbl_expand = 0; /* tbl_expand */
    int crit_act_sz = 0; /* crit_act_sz */
    int format [8] = { /* format */
        -1, -1, 1, 1, 1, /* 000 - 004 */
        1, -1, -1 /* 005 - 007 */
    };
    int offset [8] = { /* offsets */
        -1, -1, 0, 0, 0, /* 000 - 004 */
        0, -1, -1 /* 005 - 007 */
    };
    int scale [1] = {0}; /* scale factor */
    int values [1] = {255}; /* 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 [8] = { /* format */
        -1, -1, 1, 1, 1, /* 000 - 004 */
        1, -1, -1 /* 005 - 007 */
    };
    int offset [8] = { /* offsets */
        -1, -1, 0, 0, 0, /* 000 - 004 */
        0, -1, -1 /* 005 - 007 */
    };
    int scale [1] = {0}; /* scale factor */
    int values [1] = {5}; /* values */
};
struct Table3 {
    int tbl_sca_sz = 4; /* tbl_sca_sz */
    int tbl_ele_sz = 4; /* tbl_ele_sz */
    int tbl_type = 0; /* tbl_type */
}
/*
* Table 3
* This table contains the sets of polynomial coefficients which
* converts 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 [8] = { /* format */
        2, -1, 2, 2, -1, /* 000 - 004 */
        -1, -1, -1 /* 005 - 007 */
    };

```

```

};
int offset [8] = {
    0, -1, 2, 2, -1,
    -1, -1, -1
};
int scale [4] = {-4, -4, -4, -4};
int values [4] = {-2633098, 15686,
                 -17492, 4207
};
};
struct Table4 {
    int tbl_sca_sz = 0;
    int tbl_ele_sz = 9;
    int tbl_type = 1;
};
/*
 * Table 4
 * ASCII definitions of the status states
 */
int tbl_var = 4;
int tbl_expand = 0;
int crit_act_sz = 0;
int format [4] = {-1, -1, 0, 0};
int offset [4] = {-1, -1, 0, 5};
string values [9] = {
    "Undefined", "Booting", "Safe",
    "Prom", "Normal", "Stop",
    "32 sec.", "64 sec.", "128 sec."
};
};
}

```

SCANENGS VIDF File

```

vidf v3_SCANENGS {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft   */
    string experiment = "ASPERA-3";    /* exp_desc     */
    string instrument = "SCANNER";     /* 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 status indicators that are returned in the
* Main Unit Housekeeping packet which are of interest to aid in
* the understanding of the current operational status of the Scanner.
*
* 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 + (16 sensors / 4 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 = 4;                   /* status       */
    int n_sensors = 16;                 /* sen          */
    int swp_len = 1;                    /* swp_len      */
    int max_nss = 1;                    /* max_nss      */
    int data_len = 28;                  /* 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     */
};
struct Status3 {
    string name = "Scanner Speed";                      /* name      */
    int state = 4;                                       /* 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 = "Scanner CCW End Position Status";    /* 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 = "Scanner CW End Position Status";    /* 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 = "Scanner Position Clock Status";     /* 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 = "Scanner Direction Status";          /* 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 = "Scanner State";                     /* name      */
    int d_type = 0;                                    /* d_type    */
    int status = 1;                                    /* status    */
    int tdw_len = 2;                                   /* tdw_len   */
    int time_offset = 0;                              /* time_offset */
};
struct Sensor5 {
    string name = "Scanner Lost Step";                 /* name      */
    int d_type = 0;                                    /* d_type    */
    int status = 1;                                    /* status    */
    int tdw_len = 1;                                   /* tdw_len   */
};

```

```

    int time_offset = 0;                                /* time_offset */
};
struct Sensor6 {
    string name = "Scanner Initialized";                /* name          */
    int d_type = 0;                                    /* d_type        */
    int status = 1;                                    /* status        */
    int tdw_len = 1;                                   /* tdw_len       */
    int time_offset = 0;                               /* time_offset   */
};
struct Sensor7 {
    string name = "Scanner +30V Status";                /* name          */
    int d_type = 0;                                    /* d_type        */
    int status = 1;                                    /* status        */
    int tdw_len = 1;                                   /* tdw_len       */
    int time_offset = 0;                               /* time_offset   */
};
struct Sensor8 {
    string name = "Scanner Setup Mode";                /* name          */
    int d_type = 0;                                    /* d_type        */
    int status = 1;                                    /* status        */
    int tdw_len = 1;                                   /* tdw_len       */
    int time_offset = 0;                               /* time_offset   */
};
struct Sensor9 {
    string name = "Scanner Setup Direction";           /* name          */
    int d_type = 0;                                    /* d_type        */
    int status = 1;                                    /* status        */
    int tdw_len = 1;                                   /* tdw_len       */
    int time_offset = 0;                               /* time_offset   */
};
struct Sensor10 {
    string name = "Scanner Speed";                    /* name          */
    int d_type = 0;                                    /* d_type        */
    int status = 1;                                    /* status        */
    int tdw_len = 2;                                   /* tdw_len       */
    int time_offset = 0;                               /* time_offset   */
};
struct Sensor11 {
    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 Sensor12 {
    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 Sensor13 {
    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 Sensor14 {
    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 Sensor15 {
    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 = 26;               /* 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 [16] = {                /* format       */
        0, 0, 0, 0, 0, 0,             /* 000 - 005   */
        0, 0, 0, 0, 0, 0,             /* 006 - 011   */
        0, 0, 0, 0,                   /* 012 - 015   */
    };
    int offset [16] = {                /* offsets      */
        0, 2, 4, 6, 8, 12,            /* 000 - 005   */
        14, 16, 18, 6, 20, 24,        /* 006 - 011   */
        24, 24, 24, 24,                /* 012 - 015   */
    };
    string values [26] = {             /* values       */
        "TBD_Sen0", "TBD_Sen0",        /* 000 - 001   */
        "TBD_Sen1", "TBD_Sen1",        /* 002 - 003   */
        "TBD_Sen2", "TBD_Sen2",        /* 004 - 005   */
        "0 - 180", "180 - 0",          /* 006 - 007   */
        "Not Busy", "Ramp Up",         /* 008 - 009   */
        "Full Speed Move", "Ramp Down", /* 010 - 011   */
        "TBD_Sen5", "TBD_Sen5",        /* 012 - 013   */
        "TBD_Sen6", "TBD_Sen6",        /* 014 - 015   */
        "TBD_Sen7", "TBD_Sen7",        /* 016 - 017   */
        "Normal", "Manual",            /* 018 - 019   */
        "Stop", "32 seconds",          /* 020 - 021   */
        "64 seconds", "128 seconds",    /* 022 - 023   */
        "Disabled", "Enabled"          /* 024 - 025   */
    };
};
};

```

```
struct Table1 {
    int tbl_sca_sz = 0;           /* tbl_sca_sz */
    int tbl_ele_sz = 9;         /* tbl_ele_sz */
    int tbl_type = 1;          /* tbl_type */
/*
* Table 1
* This table is an ASCII look-up table which indicates what
* the settings of the status byte value means.
*/
    int tbl_var = 4;           /* tbl_var */
    int tbl_expand = 0;       /* tbl_expand */
    int crit_act_sz = 0;      /* crit_act_sz */
    int format [4] = {-1, -1, 0, 0}; /* format */
    int offset [4] = {-1, -1, 0, 5}; /* offsets */
    string values [9] = {
        "Undefined", "Booting", "Safe",
        "Prom", "Normal", "Stop",
        "32 sec.", "64 sec.", "128 sec."
    };
};
}
```

SCANPOS VIDF File

```

vidf v3_SCANPOS {
    float version = 3.0;                /* version      */
    string mission = "MARS";            /* mission      */
    string spacecraft = "Mars_Express"; /* spacecraft    */
    string experiment = "ASPERA-3";     /* exp_desc     */
    string instrument = "SCANNER";      /* 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 scanner position data that is
*   returned in the Scanner Information packet by the Main Unit.
*
*   The following is a list of tables which are to be included in
*   this vidf.
*
*   TABLE 0:  ASCII definitions of status states
*
*   Data_len is set using the equation
*       32 + 4096 positions + 4 (nanosecond timing)
*
*   32 is used instead of 20 since hdr_off[max_nss]
*/
    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 = 1;                     /* num_tbls     */
    int n_consts = 0;                   /* num_consts   */
    int n_status = 4;                   /* status       */
    int n_sensors = 1;                  /* sen          */
    int swp_len = 1;                     /* swp_len      */
    int max_packing = 4096;              /* max_packing  */
    int max_nss = 4;                    /* max_nss      */
    int data_len = 4132;                 /* 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 */
};
struct Status3 {
    string name = "Scanner Speed";                   /* name */
    int state = 4;                                    /* 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 = "Scanner Position";                 /* 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 = 0;                               /* tbl_sca_sz */
    int tbl_ele_sz = 9;                               /* tbl_ele_sz */
    int tbl_type = 1;                                 /* tbl_type */
};
/*
 * Table 0
 * 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 [4] = {-1, -1, 0, 0};                 /* format */
    int offset [4] = {-1, -1, 0, 5};                 /* offsets */
    string values [9] = {
        "Undefined", "Bootng", "Safe",               /* 000 - 002 */
        "Prom", "Normal", "Stop",                    /* 003 - 005 */
        "32 sec.", "64 sec.", "128 sec."              /* 006 - 008 */
    };
};
}

```

Appendix B - Scanner Instrument Operation Scenarios

At the present time, a detailed explanation of the operational scenarios for the Scanner is not available. Documentation will be updated as the information is gathered.