VANILLA MANUAL

A guide for software and data users

Preliminary Draft

June 2000

Table of Contents

Version and Contacts

This manual documents the Vanilla software.

Present software version: Vanilla 3.10 (6/16/00)

All examples used in this manual are from the Mars Global Surveyor Thermal Emission Spectrometer [complete listing of sample data is contained in the appendix]. In addition, all headers and labels are in PDS compliant format. Vanilla was developed at Arizona State University by Noel Gorelick, Saadat Anwar, Kamran Qazi, and Michael Weiss-Malik (the programming gurus). This manual was written by Kelly Bender (a Vanilla user, but not a programmer) and Noel Gorelick. Questions regarding Vanilla can be directed to vanilla@tes.asu.edu.

Glossary

ASCII

DATASET

Fixed-length record

Fixed-length array

Fragment

Key

Key block

PDS

Table

Vanilla

Variable-length record

Overview

Vanilla is a database query tool that allows for easy storage and retrieval of fixed and variable length data. It uses a table format and allows connection between tables via a primary key. The Vanilla search function is run in a command line format and outputs in ASCII. Vanilla is easy to learn, user friendly, and easily applied to large datasets.

Prior to using the Vanilla software, the user needs to carefully assess the data that will be put into the Vanilla database format. Many decisions have to be made regarding the segregation of the data into different tables, the assignment of column headers, and the identification of fundamental data links between the various tables that will be used. Once these issues have been laid out, this manual will help in the actual setup of the database in Vanilla format. Changes are readily accommodated by Vanilla. It is always possible to add additional tables as they become necessary (or useful), so the results of the initial database layout activity can be modified over time.

Using the MGS-TES as an example, the instrument outputs a data stream that contains radiance data, bolometric data, instrument telemetry, a record of the instrument settings used for each observation, and a time stamp for every piece of data collected. So just from the raw instrument output the following tables were devised: radiance data (rad), bolometric data (bol), telemetry (tlm), and observation parameters (obs). Time is stored in each of the tables and provides the link among them. In addition to the raw data tables, other information was needed for use in analysis of the data. In the case of MGS-TES, geometric information was provided by the project for use in calculating such parameters as footprint latitude and longitude. The calculated geometry data was put into a table (geo). Science analysis results and data quality flags are examples of later additions to the MGS-TES Vanilla database.

Table Structure

Fragments

A fragment is the smallest unit of data that Vanilla considers a "file" and therefore has a filename and contains a header at the start of the fragment prior to the data. There is no limit to the number of rows or columns contained in a fragment. Fragments with no data records are bad and will cause the Vanilla program to have problems.

Tables

A table is a collection of one or more related fragments. All fragments in a table must be sorted in ascending order based on the PRIMARY_KEY (discussed in the key section below). Vanilla assumes all files with the same name to be fragments of a single table sorted lexicographically on filename. It also assumes that all fragments in a single table contain identical keys/columns.

Headers

Each fragment is prefixed with an ASCII header in PDS 3.0 format. The format of this header consists of sets of keyword=value pairs, followed by the keyword END. A sample header is given below: [lines are numbered for easy reference – numbering is NOT part of the PDS format]

```
1       PDS_VERSION_ID                = PDS3
2       FILE_NAME                     = "OBS04101.DAT"
3       RECORD_TYPE                   = FIXED_LENGTH
4       RECORD_BYTES                  = 39
5       FILE_RECORDS                  = 1245
6       LABEL_RECORDS                 = 35
7       ^TABLE                        = 36
8       SPACECRAFT_ID                 = MGS
9       INSTRUMENT_ID                 = TES
10      MISSION_PHASE_NAME            = "MAPPING"
11      TARGET_NAME                   = MARS
12      PRODUCT_ID                    = "TES04101"
13      PRODUCER_ID                   = MGS_TES_TEAM
14      DATA_SET_ID                   = "MGS-M-TES-3-TSDR-V1.0"
15      PRODUCT_RELEASE_DATE          = 1998-08-18
16      PRODUCT_CREATION_TIME         = 1998-08-18T17:30:00
17      START_TIME                    = 1997-10-26T08:33:44.293
18      STOP_TIME                     = 1997-10-29T06:43:30.274
19      SPACECRAFT_CLOCK_START_COUNT  = 562322042
20      SPACECRAFT_CLOCK_STOP_COUNT   = 562574628
21      START_ORBIT_NUMBER            = 28
22      STOP_ORBIT_NUMBER             = 29

23      OBJECT                        = TABLE
```

```
24          NAME                        = OBS
25          INTERCHANGE_FORMAT          = BINARY
26          PRIMARY_KEY                 = (
                                        "SPACECRAFT_CLOCK_START_COUNT",
                                        "DETECTOR_NUMBER" )
27          START_PRIMARY_KEY           = ( 562322042, 1 )
28          STOP_PRIMARY_KEY            = ( 562574628, 6 )
29          ROWS                        = 1210

30          ^STRUCTURE                  = "OBS.FMT"
31      END_OBJECT                      = TABLE
32      END
```

The above header consists of three primary parts: a description of the whole fragment, a pointer to the binary data, and a set of nested PDS objects that identify the contents and layout of the fragment.

Line 1 identifies the PDS format version for the entire header. Line 2 is the fragment filename. Fragment (or file) names are comprised of three parts: the table name, the fragment number and an extension. Table names are defined in the OBJECT=TABLE section of the header (lines 24). The table name can be any combination of letters, is case insensitive, and can be any length (TES uses three letters). Fragment numbers must be a numeric string. There is no limit on the length of the fragment numbers. The extension will be .DAT or .VAR depending on the data type. The fixed-length records are stored in files with a .DAT extension. The variable length records that are referenced by an individual .DAT file can be found in a file with the same name, but with a .VAR extension. The .DAT extension can take the form .[Dd][Aa][Tt] or .[Tt][Aa][Bb].

Lines 3 - 6 of the header describe the overall structure of the fragment and in this case indicate that it consists of 1,245 fixed-length records, 39 bytes in length. The entire ASCII header is padded with white space to occupy an integral number of records of the 39 byte length. The header record number is reported on line 6.

The keyword ^TABLE is a pointer to the start of the binary data (line 7). The number given with this keyword is the record number of the start of the data. In this case the record number is 36, which starts at byte 1404 counting from byte zero (35 records * 39 bytes/record).

Lines 8 through 22 are PDS required entries used for archival and data distribution purposes.

Nested object items (lines 23 – 31) identifies the table that the fragment is part of (line 24), the data format (line 25), the key element used for connection among the various tables (line 26), the start and stop values of the key element in the fragment (lines 27 and 28), and the number of rows of data contained in the fragment (29). The keyword ^STRUCTURE (line 30) points to a separate file which details the individual column descriptions. The OBJECT=COLUMN listings are kept in a separate format file (extension .FMT) that the Vanilla program will access in order to read the data. By separating the .FMT files for the different tables of data, the user is provided with an easy

reference to the contents of each table. Knowledge of the individual column names is necessary for utilizing the search capabilities of Vanilla.

Columns

All data columns have a name. Occurrences of the same column name in different tables are assumed by Vanilla to be related. In other words, column descriptions must be unique but can be used in different tables. For example, NAME = DETECTOR is used in all TES tables, but the column data type and description are identical everywhere.

The following data types are allowable for use in Vanilla columns:

A. 1, 2, or 4 byte signed or unsigned integer

B. 4 or 8 byte real number

C. fixed length string

D. fixed length array of form A, B, or C

E. 1, 2, or 4 byte bit string of signed or unsigned integer bit fields

F. variable length array of form A, B, or C with maximum array length of 32 kbytes

Vanilla search functions can be used to constrain data extraction in columns with data types A, B, C, D, or E. Variable length arrays (type F) can not be constrained during data extraction. More information about the search capabilities of Vanilla is presented later in this document.

Object listings for all columns in a table are contained in the ^STRUCTURE file referenced by each data fragment. Part of a sample STRUCTURE file (obs.fmt) is shown below. [see Appendix for complete *.fmt examples.] The STRUCTURE file lists the table name, the total number of columns contained in the table/fragment, the number of bytes per row (all rows must be the same), a description of the table, and an OBJECT=COLUMN entry for each column [only one is listed below].

```
NAME                    = OBS
COLUMNS                 = 6
ROW_BYTES               = 14
DESCRIPTION             = " The OBS table stores the state of the
                        instrument at the start of each observation.
                        One OBS record is generated for each
                        observation."

OBJECT                  = COLUMN
    NAME                = SPACECRAFT_CLOCK_START_COUNT
    DATA_TYPE           = MSB_UNSIGNED_INTEGER
    START_BYTE          = 1
```

```
    BYTES               = 4
    ALIAS_NAME          = sclk_time
    DESCRIPTION         = "The value of the spacecraft clock at the
                        beginning of the observation"
END_OBJECT              = COLUMN

OBJECT                  = COLUMN
    NAME                = PNT_ANGLE
    DATA_TYPE           = MSB_INTEGER
    START_BYTE          = 12
    BYTES               = 2
    SCALING_FACTOR      = .046875
    DESCRIPTION         = "Scan mirror pointing angle, degrees from
                        nadir."
END_OBJECT              = COLUMN
```

Each OBJECT=COLUMN listing includes the column name, data type, starting position (in bytes), size (in bytes), scaling factors if applicable, and description. A scaling factor is used to convert from the stored value to useful units. A scaling offset may also be included, but if not included, should be assumed to be zero. Scaling factors and offsets should be applied as follows:

scaled_value = (stored_value * scaling_factor) + scaling_offset

Only fixed length data can be scaled and offset. Scaling factors and offsets are applied automatically by Vanilla during data extraction and automatically convert the data type to floating point.

Descriptions are provided for every column. These descriptions are surrounded by quotes and may span several lines.

In some cases the column is a fixed-length array of related, homogeneous values (such as temperatures or voltages). For that case, the column listing also includes the BYTES term to indicate the size of the array, and two fields (ITEMS and ITEM_SIZE) to describe the number and size of a single element in the array. For fixed-length arrays, column "padding" (leaving unused bytes between data bytes) is allowed, as the start bytes are listed in the column information.

The following column listing indicates that the column is a homogeneous array of 6, 2-byte integers.

```
   OBJECT               = COLUMN
        NAME            = INTERFEROGRAM_MAXIMUM
        DATA_TYPE       = MSB_INTEGER
        START_BYTE      = 29
        BYTES           = 12
        ITEMS           = 6
        ITEM_BYTES      = 2
        SCALING_FACTOR  = .000152587890625
        DESCRIPTION     = "Array of 6 interferogram maximum values"
   END_OBJECT           = COLUMN
```

Variable Length Files

Variable length data are stored in files separate from the fixed-length data and use .VAR extensions. For every .VAR fragment there will exist a .DAT fragment with the same name. The .DAT fragment will contain all the necessary headers and a "pointer" column. Pointer columns contain the position of the variable length data, in bytes, from the start of the relevent .VAR file. A position value of -1 in a pointer column indicates that there are no variable length data for that record.

Additional keywords in the OBJECT=COLUMN listing are used to identify it as a pointer to a variable length column, and describe the data contained in the variable length records. These keywords are:

```
VAR_DATA_TYPE
VAR_ITEM_BYTES
VAR_RECORD_TYPE
```

The VAR_DATA_TYPE and VAR_ITEM_BYTES keywords are similar to the PDS keywords DATA_TYPE and ITEM_BYTES, but refer to the structure of the variable-length data. The VAR_RECORD_TYPE keyword identifies the overall format of the variable-length record. This keyword has two possible values:

```
VAR_RECORD_TYPE = VAX_VARIABLE_LENGTH
VAR_RECORD_TYPE = Q15
```

The value VAX_VARIABLE_LENGTH indicates that the variable-length record has the size of the record in bytes, as a 2-byte integer, both before and after the record. This corresponds to the VAX/VMS variable-length record format.

The Q15 format is very similar to the VAX_VARIABLE_LENGTH format; however it is only used to store floating point values in a compact representation. This format is an array of floating point mantissas stored as 2-byte signed integers. These mantissas share a scaling exponent that is stored as the first item in the record as another 2-byte signed integer. All the elements in the array must be scaled by the exponent, by multiplying them by 2 to the power (exp-15). Just like the VAX_VARIABLE_LENGTH records, the Q15 records are also stored with the size of the record in bytes, as a 2-byte integer, both before and after the record.

A sample variable length fragment header is shown below.

```
OBJECT                  = COLUMN
    NAME                = RAW_RADIANCE
    DATA_TYPE           = MSB_UNSIGNED_INTEGER
    START_BYTE          = 9
    BYTES               = 4
    VAR_DATA_TYPE       = MSB_INTEGER
    VAR_ITEM_BYTES      = 2
    VAR_RECORD_TYPE     = Q15
    ALIAS_NAME          = raw_rad
    DESCRIPTION         = "Raw spectral radiance"
    UNIT                = "transformed volts"
```

```
END_OBJECT                  = COLUMN
```

See Figure 1 for an illustration of a variable length record and how it related to the column listing. See Figure 2 for a diagram of the Q15 variable length record.

Linking Tables

Keys

To access the various tables of data, there must exist a fundamental logical element(s) common to the entire dataset. This fundamental element is called a key and must be numeric in value. A key may have more than one element, in which case it is termed a composite key.

Some tables may have just a single element key while others may have composite keys. The key is, in effect, the column heading for the first column (or columns) of data. Each row of data in a fragment/table will have a unique value for the key. Different fragments/tables with the same key are assumed to be related. Hence, keys provide the link between the different tables of data.

All data headers will contain the key (termed PRIMARY_KEY in PDS compliant format) and the start and stop values for the group of data to which the header belongs. For example:

```
PRIMARY_KEY          = ( "SPACECRAFT_CLOCK_START_COUNT",
                       "DETECTOR_NUMBER" )
START_PRIMARY_KEY    = ( 562322042, 1 )
STOP_PRIMARY_KEY     = ( 562574628, 6 )
```

The PRIMARY_KEY for any fragment may contain different key elements from other fragments; however all keys must be a partial subset of the longest composite key and all key elements must occur in the same order as the longest composite key's elements.

Key Blocks

For any fragment with a composite key, it is possible for a set of rows to contain the same value for the first element of the key and unique values for the second element. For example, using the PRIMARY_KEY above, for any single time value there can exist up to six different rows of data because there are six possible values for detector. Rows of data must be sorted in ascending order based on the PRIMARY_KEY. So, for the above example, the data must start with detector row 1 and end with detector row 6. A group of data rows with the same value for the first key is termed a key block.

An integral number of key blocks are combined to make a fragment. Key blocks cannot span fragment boundaries. The key blocks composing the fragment must be sorted in ascending order based on the PRIMARY_KEY.

Dataset

For Vanilla to access data it must have a list of the table names and the locations of their fragments. This information is stored in a file named DATASET (case sensitive). The following formats are allowable:

1 One or more fragment filenames for fragments resident in the directory containing the DATASET file (for example obs00001.dat)

2 One or more table names for tables resident in the directory containing the DATASET file (for example obs). Vanilla will automatically assume that all fragments with the same table name are related.

3 One or more path/table name or path/fragment name entries (for example /mapping/data/p1/obs or /mapping/data/p1/obs00001.dat)

4 One or more path listings that direct Vanilla to a location where another DATASET file, with appropriate contents, is in residence (for example /mapping/data/p1 – and p1 contains a DATASET file).

Vanilla uses DATASET to find all available fragments. It then reads the header of the first fragment of each table to find the column definitions (assuming that all fragments with the same table name will contain the same columns). Once Vanilla has accessed this information it is able to search the data as desired by the user. Vanilla can only access tables that are listed in the DATASET file. Nonexistent tables listed in DATASET are ignored by Vanilla.

Data Searches

The real power of Vanilla is its search capabilities. Searches are done using a command line input. Data is output in ASCII format and can be read into other tools for data visualization and analysis.

The usage for Vanilla is:

>vanilla directory –fields “col1 col2 . . .” –select “select1 select2 . . .”

The meaning of each of the arguments is explained below.

directory

This argument is required and must be the absolute or relative path to a directory containing a DATASET file.

-fields “col1 col2 . . .”

This argument is required and identifies the table columns to output. The list of columns must be presented as a single string, and so, must be enclosed in quotes if more than one column is given. The format of a column identifier is:

table.column[index]

Where the ‘table’ and ‘[index]’ portions are optional. Column identifiers are separated by spaces, so no spaces are allowed within a column identifier.

The table prefix is only necessary when multiple tables contain columns with the same name (such as key columns). If the table prefix is not specified, the first table listed in the DATASET file that contains the named column is assumed.

It is possible for some columns in PDS tables to be specified as containing an array of homogeneous data elements. For these array columns, the optional [index] is used to specify which element(s) of the array to extract. The index can be a single number, indicating a single element of the array, or a range of numbers (specified by [low:high]), indicating multiple consecutive elements. If the user specifies the name of an array column and does not specify an index or leaves it blank (eg: column[]), the entire array is output.

Some columns are composed of multiple bit fields. If only the bit column is specified, the entire “bit word” is output as an integer number. In most cases this is probably not desirable. To extract a single bit field, the column should be specified by column:bit_column.

All variable length arrays require at least an empty index (ie: name[]). Leaving off the index extracts the variable length data pointer (the position of the variable length data in its .VAR file).

If columns from multiple tables are specified, an inner join is performed between all the tables involved and only those records that exist in all of the tables specified are output. Searches that use columns in the -fields portion that access tables with no keys in common will return zero records. If the search uses column names that do not exist in the fragments available from the DATASET, then Vanilla returns no records. This is allowable and does not cause any problems.

-select "select1 select2 . . . "

This argument is optional and specifies a selection criteria that a record must meet before it is output. Like the –fields argument, all the selection criteria must be presented to Vanilla as a single string and so must be enclosed in quotes. The column(s) used in the -select portion do not have to be the same ones used in the -fields portion. The format for a selection is as follows:

table.column[index] lowvalue highvalue

Like the –fields options, the 'table' and '[index]' potions are optional and carry the same meaning. However, with or without the '[index]' portion, the column identifier must specify only a single data element (eg: the [low:high] fomat for index is not allowed, and omitting the index is not allowed for arrays).

The 'lowvalue' and 'highvalue' portions of the selection specify a range that the column value must lie within before a record is output. Records that don't meet all the selection criteria are discarded. The ranges are inclusive; a value must satisfy the following relation for the record to be considered for output:

lowvalue <= column value <= highvalue

If the column contains a character value, the comparison is lexicographical.

Appendix

This appendix contains the sample data set used as examples in this document. The tables are all contained in a single directory along with the necessary format files – as reflected by the DATASET file.

Directory Contents

```
DATASET
geo.fmt
geo07000.dat
obs.fmt
obs07000.dat
rad.fmt
rad07000.dat
rad07000.var
tlm.fmt
tlm07000.dat
```

File Contents

DATASET

```
obs
geo
rad
tlm
```

geo.fmt

```
NAME                = GEO
COLUMNS             = 7
ROW_BYTES           = 15
DESCRIPTION         = "The GEO table contains information about the
                      sun/spacecraft/target geometry in a format that is
                      easily searchable.  These values are computed for
                      every scan other than those used to calibrate the
                      instrument.  If a viewing vector does not
                      intersect the target body (i.e., an atmospheric
                      observation), then most of the geometry is
                      calculated relative to the point on the viewing
                      vector closest to the body (i.e., the tangent
                      point).  If the closest point lies behind the
                      spacecraft, fill values are used."

OBJECT              = COLUMN
   NAME             = SPACECRAFT_CLOCK_START_COUNT
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 1
```

```
   BYTES            = 4
   ALIAS_NAME       = sclk_time
   DESCRIPTION      = "The value of the spacecraft clock at the
                      beginning of the observation"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = DETECTOR_NUMBER
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 5
   BYTES            = 1
   ALIAS_NAME       = detector
   DESCRIPTION      = "The number of the spectrometer detector that
                      made the observation. Detectors are numbered from
                      1 to 6"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = LONGITUDE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 6
   BYTES            = 2
   SCALING_FACTOR   = 0.01
   DESCRIPTION      = "Areocentric west longitude of target point"
   UNIT             = "DEGREE"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = LATITUDE
   DATA_TYPE        = MSB_INTEGER
   START_BYTE       = 8
   BYTES            = 2
   SCALING_FACTOR   = 0.01
   DESCRIPTION      = "Areocentric latitude of target point"
   UNIT             = "DEGREE"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = PHASE_ANGLE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 10
   BYTES            = 2
   SCALING_FACTOR   = 0.01
   ALIAS_NAME       = phase
   DESCRIPTION      = "Angle between the spacecraft, the target point
                      and the sun"
   UNIT             = "DEGREE"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = EMISSION_ANGLE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 12
   BYTES            = 2
   SCALING_FACTOR   = 0.01
   ALIAS_NAME       = emission
   DESCRIPTION      = "Angle between the spacecraft, the target point
```

```
                    and the surface normal vector at the target"
   UNIT             = "DEGREE"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = INCIDENCE_ANGLE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 14
   BYTES            = 2
   SCALING_FACTOR   = 0.01
   ALIAS_NAME       = incidence
   DESCRIPTION      = "Angle between the sun, the target point and the
                    surface normal vector at the target"
   UNIT             = "DEGREE"
END_OBJECT          = COLUMN
```

obs.fmt

```
NAME                = OBS
COLUMNS             = 6
ROW_BYTES           = 14
DESCRIPTION         = "The OBS table stores the state of the
                    instrument at the start of each observation.  One
                    OBS record is generated for each observation."

OBJECT              = COLUMN
   NAME             = SPACECRAFT_CLOCK_START_COUNT
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 1
   BYTES            = 4
   ALIAS_NAME       = sclk_time
   DESCRIPTION      = "The value of the spacecraft clock at the
                    beginning of the observation"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = ORBIT_NUMBER
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 5
   BYTES            = 2
   ALIAS_NAME       = orbit
   DESCRIPTION      = "The number of the orbital revolution of the
                    spacecraft around Mars for the observation"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = INSTRUMENT_TIME_COUNT
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 7
   BYTES            = 4
   ALIAS_NAME       = ick
   DESCRIPTION      = "The number of two-second intervals that have
                    elapsed since the start of the orbit. The two-
                    second interval is the smallest time unit defined
                    by the instrument and equals the time to complete
                    a single length scan."
```

```
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = TEMPORAL_AVERAGE_COUNT
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 11
   BYTES            = 1
   ALIAS_NAME       = tic
   DESCRIPTION      = "The number of two-second scans averaged into
                      this observation.  Valid values are 1, 2 and 4"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = MIRROR_POINTING_ANGLE
   DATA_TYPE        = MSB_INTEGER
   START_BYTE       = 12
   BYTES            = 2
   SCALING_FACTOR   = .046875
   ALIAS_NAME       = pnt_angle
   DESCRIPTION      = "Scan mirror pointing angle, degrees from nadir
                      about the spacecraft's +Y axis."
   UNIT             = "DEGREE"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = IMC_COUNT
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 14
   BYTES            = 1
   ALIAS_NAME       = pnt_imc
   DESCRIPTION      = "The number of image motion compensation steps
                      used."
END_OBJECT          = COLUMN
```

rad.fmt

```
NAME                = RAD
COLUMNS             = 8
ROW_BYTES           = 20
DESCRIPTION         = "
                      The RAD table contains the raw and calibrated
                      observed radiances. For each observation there can
                      be up to 6 RAD records, one for each active
                      spectrometer detector.  If the Temporal
                      Integration Count (OBS Table,
                      TEMPORAL_AVERAGE_COUNT) is greater than 1, then
                      the data represent the average of the measurements
                      from that many scans.

                      The instrument can apply a programmable spectral
                      mask to the raw data causing neighboring channels
                      to be averaged; however, this feature is used only
                      when downlink bandwidth is limited.  When
                      spectrally masked data are received, the averaged-
                      out channels are replaced with the averaged value
```

```
                    to expand the spectra back to its original size.
                    The spectral-mask that was used to perform the
                    averaging is kept in this table.

                    The raw spectra are compressed for downlink.  The
                    original bit-packed compression header, containing
                    the size of the compressed data and the
                    compression mode used, is kept in this table in
                    order to be used to evaluate the performance of
                    the compressor."

OBJECT              = COLUMN
   NAME             = SPACECRAFT_CLOCK_START_COUNT
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 1
   BYTES            = 4
   ALIAS_NAME       = sclk_time
   DESCRIPTION      = "The value of the spacecraft clock at the
                    beginning of the observation"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = DETECTOR_NUMBER
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 5
   BYTES            = 1
   ALIAS_NAME       = detector
   DESCRIPTION      = "The number of the spectrometer detector that
                    made the observation. Detectors are numbered from
                    1 to 6"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = SPECTRAL_MASK
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 6
   BYTES            = 1
   ALIAS_NAME       = spectral_mask
   DESCRIPTION      = "ID number of spectral mask applied.  See
                    ancillary Masks table"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = COMPRESSION_MODE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 7
   BYTES            = 2
   ALIAS_NAME       = cmode
   DESCRIPTION      = "16-bit compression header of original data
                    containing the size and compression mode of the
                    original compressed data.  See TES Users Guide."
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = RAW_RADIANCE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 9
```

```
   BYTES            = 4
   VAR_DATA_TYPE    = MSB_INTEGER
   VAR_ITEM_BYTES   = 2
   VAR_RECORD_TYP   = Q15
   E
   ALIAS_NAME       = raw_rad
   DESCRIPTION      = "Raw spectral radiance"
   UNIT             = "transformed volts"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = CALIBRATED_RADIANCE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 13
   BYTES            = 4
   VAR_DATA_TYPE    = MSB_INTEGER
   VAR_ITEM_BYTES   = 2
   VAR_RECORD_TYP   = Q15
   E
   ALIAS_NAME       = cal_rad
   DESCRIPTION      = "Calibrated spectral radiance"
   UNIT             = "watts cm-2 steradian-1 wavenumber-1"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = DETECTOR_TEMPERATURE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 17
   BYTES            = 2
   ALIAS_NAME       = tdet
   DESCRIPTION      = "Derived temperature of the detector, used to
                      remove instrument radiance in calibration
                      algorithm"
   UNIT             = "K"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = TARGET_TEMPERATURE
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 19
   BYTES            = 2
   ALIAS_NAME       = target_temp
   DESCRIPTION      = "Derived temperature of the observed target"
   UNIT             = "K"
END_OBJECT          = COLUMN
```

tlm.fmt

```
NAME                = TLM
COLUMNS             = 5
ROW_BYTES           = 64
DESCRIPTION         = "
                      The TLM table stores the auxiliary observation
                      parameters downlinked with the long packet format
                      (see OBS Table, DATA_PACKET_TYPE). Records in the
                      TLM table occur at a frequency less than or equal
```

```
                    to the frequency of OBS records; that is, one (or
                    none) per observation."

OBJECT              = COLUMN
   NAME             = SPACECRAFT_CLOCK_START_COUNT
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 1
   BYTES            = 4
   ALIAS_NAME       = sclk_time
   DESCRIPTION      = "The value of the spacecraft clock at the
                    beginning of the observation"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = AUXILIARY_DIAGNOSTIC_TEMPS
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 5
   BYTES            = 24
   ITEMS            = 12
   ITEM_BYTES       = 2
   SCALING_FACTOR   = 0.01
   ALIAS_NAME       = aux_temps
   DESCRIPTION      = "Array of 12 auxiliary temperatures, Read from
                    internal instrument thermistors.
                    1: T5 - Black Body 1
                    2: T6 - Black Body 2
                    3: T7 - Black Body 3
                    4: T8 - Bolometric Black Body Reference (spare)
                    5: T9 - Electronics
                    6: T10 - Power Supply
                    7: T11 - Telescope Field Stop
                    8: T12 - Interferometer Fixed Mirror
                    9: T13 - Interferometer Beamsplitter
                    10: T14 - Interferometer Motor
                    11: T15 - Primary Mirror
                    12: T16 - Secondary Mirror"
   UNIT             = "K"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = INTERFEROGRAM_MAXIMUM
   DATA_TYPE        = MSB_INTEGER
   START_BYTE       = 29
   BYTES            = 12
   ITEMS            = 6
   ITEM_BYTES       = 2
   SCALING_FACTOR   = 0.000152587890625
   ALIAS_NAME       = ifgm_max
   DESCRIPTION      = "Array of 6 interferogram maximum values, one
                    for each spectrometer detector. Scaling factor is
                    5.0/32768 V"
   UNIT             = "VOLTS"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = INTERFEROGRAM_MINIMUM
   DATA_TYPE        = MSB_INTEGER
```

```
   START_BYTE       = 41
   BYTES            = 12
   ITEMS            = 6
   ITEM_BYTES       = 2
   SCALING_FACTOR   = 0.000152587890625
   ALIAS_NAME       = ifgm_min
   DESCRIPTION      = "Array of 6 interferogram minimum values, one
                      for each spectrometer detector. Scaling factor is
                      5.0/32768 V"
   UNIT             = "VOLTS"
END_OBJECT          = COLUMN

OBJECT              = COLUMN
   NAME             = ONBOARD_PROCESSING_EVENT_LOG
   DATA_TYPE        = MSB_UNSIGNED_INTEGER
   START_BYTE       = 53
   BYTES            = 12
   ITEMS            = 6
   ITEM_BYTES       = 2
   ALIAS_NAME       = dsp_log
   DESCRIPTION      = "Array of digital signal processor event logs,
                      16-bit mask, one for each spectrometer detector.
                      See TES User's Guide for details"
END_OBJECT          = COLUMN
```

Figures

Figure 1

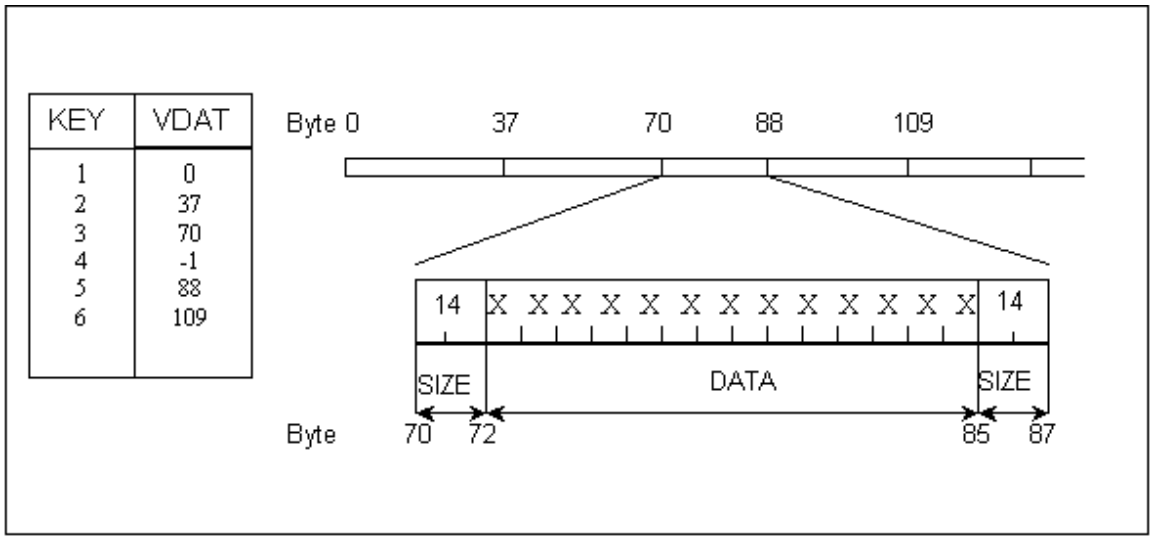

Figure 1. This figure illustrates the use of variable-length records, and how they relate to the fixed-length records. In this example, the table contains 2 columns, one of which is a pointer to the variable-length records. The table shows 6 rows, but only 5 of the rows actually point to variable-length records. The fourth record contains -1 in the pointer column, indicating that there are no variable-length data for that row. The two column listings are shown below.

```
OBJECT                  = COLUMN
    NAME                = KEY
    DATA_TYPE           = ASCII_INTEGER
    BYTES               = 1
END_OBJECT              = COLUMN

OBJECT                  = COLUMN
    NAME                = VDATA
    DATA_TYPE           = ASCII_INTEGER
    BYTES               = 2
    VAR_ITEM_BYTES      = 1
    VAR_RECORD_TYPE     = CHARACTER
    VAR_DATA_TYPE       = VAX_VARIABLE_LENGTH
END_OBJECT              = COLUMN
```

Figure 2

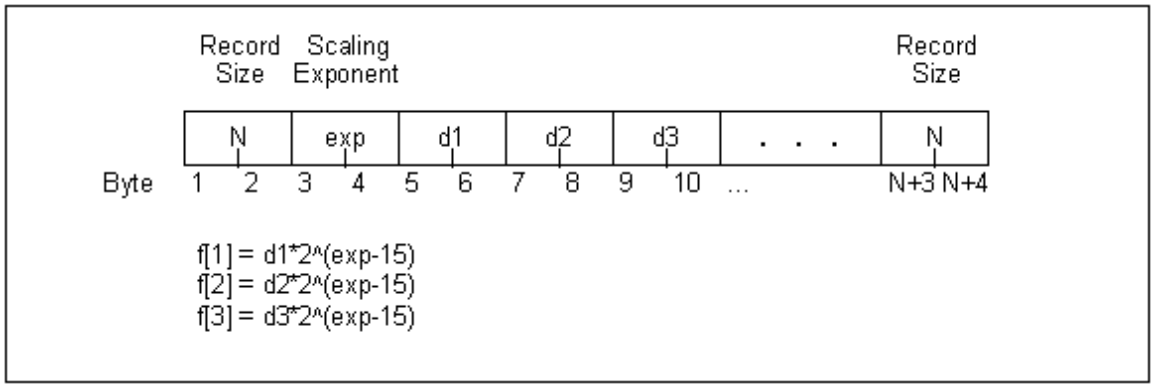

Figure 2. A diagram of a complete Q15 variable length record.