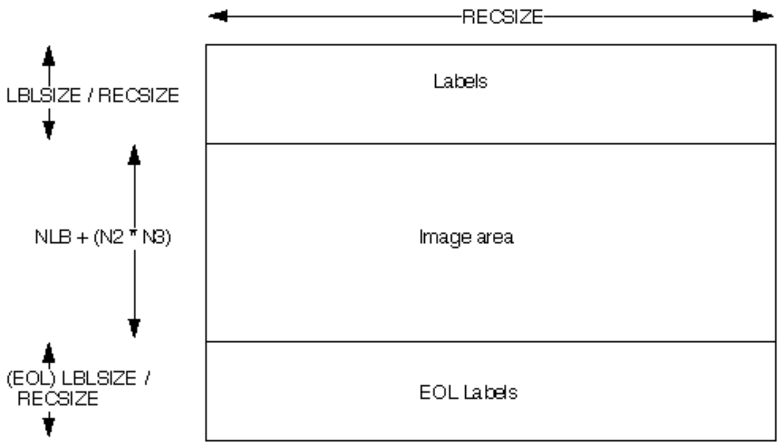
The VICAR file format

By Robert G. Deen

This document describes the format of VICAR files. It applies to files created with version 8.0 or later of VICAR. Files created earlier may not have some sections (like property labels) or some system label items (like the various HOST-related items), but they still conform to this specification. The proper application of defaults given below will ensure that all VICAR files may be read using this spec. Any VICAR files written out must include all the system label items defined below.

Overview

The basic structure of a VICAR file is shown below.



A VICAR file consists of two major parts: the labels, which describe what the file is, and the image area, which contains the actual image. The labels are potentially split into two parts, one at the beginning of the file, and one at the end. Normally, only the labels at the front of the file will be present. However, of the EOL keyword in the system label (described below) is equal to 1, then the EOL labels (End Of file Labels) are present. This happens if the labels expand beyond the space allocated for them.

The VICAR file is treated as a series of fixed-length records, of size RECSIZE (see below). The image area always starts at a record boundary, so there may be unused space at the end of the label, before the actual image data starts.

Labels

The label consists of a sequence of "keyword=value" pairs that describe the image, and is made up entirely of ASCII characters. Each keyword-value pair is separated by spaces. Keywords are strings, up to 32 characters in length, and consist of uppercase characters, underscores (_), and numbers (but should start with a letter). Values may be integer, real, or strings, and may be multiple (e.g. an array of 5 integers, but types cannot be mixed in a single value). Spaces may appear on either side of the equals character (=), but are not normally present.

The first keyword is always LBLSIZE, which specifies the size of the label area in bytes. LBLSIZE is always a multiple of RECSIZE, even if the labels don't fill up the record. If the labels end before LBLSIZE is reached (the normal case), then a 0 byte terminates the label string. If the labels are exactly LBLSIZE bytes long, a null terminator is *not necessarily* present. The size of the label string is determined by the occurrence of the first 0 byte, or LBLSIZE bytes, whichever is smaller.

If the system keyword EOL has the value 1, then End-Of-file Labels exist at the end of the image area (see above). The EOL labels, if present, start with another LBLSIZE keyword, which is treated exactly the same as the main LBLSIZE keyword. The length of the EOL labels is the smaller of the length to the first 0 byte or the EOL's LBLSIZE. Note that the main LBLSIZE does *not* include the size of the EOL labels. In order to read in the full label string, simply read in the EOL labels, strip off the LBLSIZE keyword, and append the rest to the end of the main label string.

The label is divided into three logical parts: System labels, Property labels, and History labels, in that order. These parts are described later in this section.

Label Values

The label values may be of three types: integer, real, or string.

- Integer: A sequence of digits (0-9), with an optional sign (+/-). There must be no embedded blanks in the integer, including between the sign and the number. In C, an integer should be created with the %d format in sprintf().
- Real: A sequence of digits (0-9) including a decimal point (.), an optional sign(+/-), and an optional exponent (one of the letters EeDd followed by a base-10 exponent in integer format). The letter E is greatly preferred for indicating the exponent. There must be no embedded blanks in the real number. The number must contain either a decimal point or an exponent, or else it will be considered an integer. The number of significant digits is variable, so the number may be read as either single or double precision. In C, a real number should be created with the %g format in sprintf().
- String: A string is a sequence of ASCII characters enclosed in single quotes ('). A single quote may be included in the string by doubling it (i.e. 'can''t'). The quotes enclosing the string value may be discarded if the string contains no blanks or special characters, and contains at least one letter that cannot be interpreted as a number (i.e. it does not consist entirely of the letters E and D). However, this is rarely done, and any labels written should enclose strings in quotes.

A keyword may have more than one value by enclosing the values in parentheses and separating the values with commas. The collection of values is treated like an array for that keyword. All values in a multi-valued label item must be of the same type. Spaces may exist around the parentheses or the commas, but are not normally present.

Examples

```
LBLSIZE=1024
FORMAT='BYTE'
LATITUDE=45.3
COORDS=(5.7,-3.2E+2)
COMMENTS=('Wow, this is a comment!', 'This can''t be real')
EXTRA_SPACES =    (    1,   2,3,        4      ,    -5  )
```

System Labels

System labels describe the format of the image and how to access it. They are always the first labels in the file. The system labels extend from the beginning of the file until the first PROPERTY or TASK keyword, or until the end of the label (if there are no property or history labels).

Any program attempting to read a VICAR file should be able to accept (and ignore) system label items it doesn't understand, as new system label items are added from time to time.

Some system label items are mandatory, while others are optional. The mandatory ones are mentioned in the descriptions below. However, when writing a new file, *all* system label items should normally be included.

The currently defined system label items are listed below. They generally appear in the order listed, but there is no guarantee that the items will be in any particular order, except that LBLSIZE must always be first. So, any program that reads the label must be able to handle any order of label items.

- LBLSIZE, integer, mandatory: The size of the label storage area, in bytes. It is always the first thing in the file. This label will appear twice if EOL labels are present; once at the beginning of the file and once at the beginning of the EOL labels. The size specified applies only to the section (main or EOL) that the LBLSIZE item is in.
- FORMAT, string, mandatory: The data type of the pixels in the image. Valid values are:

    - BYTE: one byte unsigned integer, range 0 to 255.
    - HALF: two byte signed integer, range -32768 to 32767.
    - FULL: four byte signed integer.
    - REAL: single-precision floating point number.
    - DOUB: double-precision floating point number.
    - COMP: complex number, composed of two REALs in the order (real, imaginary).

The following values are obsolete, but may appear in some older images:

    - WORD: same as HALF
    - LONG: same as FULL
    - COMPLEX: same as COMP

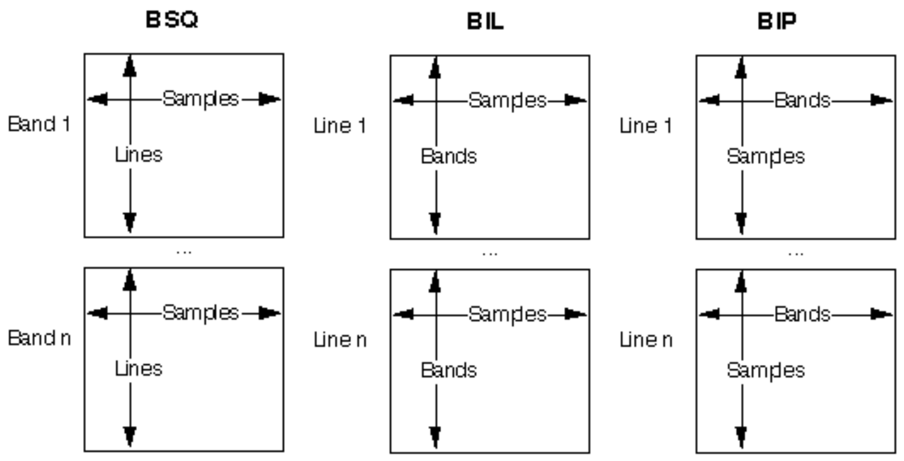
- TYPE, string: The kind of file this is. TYPE defaults to IMAGE. The valid values may very well be expanded in the future, but currently they are:

    - IMAGE: standard VICAR image file. This is the only type fully described in this document.
    - PARMS: very old-style parameter file.
    - PARM: old-style parameter file.
    - PARAM: current parameter file, used to hold input parameters for one VICAR program that were generated by another. Created by the x/zvpopen and x/zvpout routines in the VICAR Run-Time Library.
    - GRAPH1: IBIS Graphics-1 file.
    - GRAPH2: IBIS Graphics-2 file.
    - GRAPH3: IBIS Graphics-3 file.
    - TABULAR: IBIS Tabular file.

- BUFSIZ, integer, mandatory: This label item is obsolete. It formerly defined the size of the internal buffer to use when reading the image, but it is no longer used. It still must be present for historical reasons, however. When creating a new file, just set BUFSIZ equal to RECSIZE.
- DIM, integer: The number of dimensions in the file, which is always equal to 3. Some older images may have a DIM of 2, in which case some labels will not be present. Note that the dimension is 3 even if N3=1 (e.g. there is only one band in a BSQ file). The default is 3.
- EOL, integer: A flag indicating the existence of EOL labels (see above). If EOL=1, the labels are present. If EOL=0 (or is absent), no EOL labels are present, and the entire label string is at the front of the file.
- RECSIZE, integer, mandatory: The size in bytes of each record in the VICAR file. It may be calculated with the formula NBB + N1*pixel_size, where pixel_size is the size of each pixel computed using FORMAT (for the pixel type) and the INTFMT or REALFMT (for the host representation) labels.
- ORG, string: The organization of the file. While N1 is always the fastest-varying dimension, and N3 is the slowest, the terms Samples, Lines, and Bands may be interpreted in different ways. ORG specifies which interpretation to use, and defaults to BSQ. The valid values are:

    - BSQ: Band SeQuential. The file is a sequence of bands. Each band is made up of lines, which are in turn made up of samples. So, N1=Samples, N2=Lines, and N3=Bands. This is the most common case.
    - BIL: Band Interleaved by Line. The file is a sequence of lines. Each line is made up of bands, which are in turn made up of samples. So, N1=Samples, N2=Bands, and N3=Lines.
    - BIP: Band Interleaved by Pixel. The file is a sequence of lines. Each line is made up of samples, which are in turn made up of bands. So, N1=Bands, N2=Samples, and N3=Lines.

The three organizations are depicted graphically below.



- NL, integer, mandatory: The number of lines in the image (same as N2 for BSQ or N3 for BIL and BIP).
- NS, integer, mandatory: The number of samples in the image (same as N1 for BSQ and BIL or N2 for BIP).
- NB, integer, mandatory: The number of bands in the image (same as N3 for BSQ, N2 for BIL, or N1 for BIP).
- N1, integer: The size (in pixels) of the first (fastest-varying) dimension. If not present, it defaults to NS or NB, as appropriate.

- N2, integer: The size of the second dimension. If not present, it defaults to NL, NS, or NB, as appropriate.
- N3, integer: The size of the third (slowest-varying) dimension. If not present, it defaults to NL or NB, as appropriate.
- N4, integer: This item was to have been used for four-dimensional files, but this has not yet been implemented. It defaults to 0.
- NBB, integer: The number of bytes of binary prefix before each record. Each and every record consists of the pixels of the fastest-varying dimension, optionally preceded by a binary prefix. The size (in bytes, not pixels) of this binary prefix is given by NBB, which defaults to 0. The binary prefix and the binary header (see NLB) together make up the binary label. The format of data in the binary label is application-defined. The BLTYPE label is intended to identify the format of the binary label, but it is new and not widely used. Generally, the binary label should be ignored unless the format of the data is known beforehand.
- NLB, integer: The number of lines (records) of binary header at the top of the file. The optional binary header occurs once in the file, between the main labels and the image data. It is not repeated per third dimension. The size of the binary header in bytes is given by NLB * RECSIZE, since NLB is a line count. NLB defaults to 0. Note that the binary header also includes space reserved for the binary prefix (NBB), since NBB goes into RECSIZE. The binary header and the binary prefix (see NBB) together make up the binary label. The format of data in the binary label is application-defined. The BLTYPE label is intended to identify the format of the binary label, but it is new and not widely used. Generally, the binary label should be ignored unless the format of the data is known beforehand.
- HOST, string: The type of computer used to generate the image. It is used only for documentation; the INTFMT and REALFMT labels are used to determine the format of the pixels. Nevertheless, it should be kept consistent with INTFMT and REALFMT. HOST defaults to VAX-VMS. The value may be anything, as new computer types are occasionally added, but as of this writing, the possible values are:

    - ALLIANT: Alliant FX series computer.
    - CRAY: Cray (port is incomplete, and Cray format is not yet supported).
    - DECSTATN: DECstation (any DEC MIPS-based RISC machine) running Ultrix.
    - HP-700: HP 9000 Series 700 workstation.
    - MAC-AUX: Macintosh running A/UX.
    - MAC-MPW: Macintosh running native mode with Mac Programmers Workbench.
    - SGI: Silicon Graphics workstation.
    - SUN-3: Sun 3, any model.
    - SUN-4: Sun 4 or SPARCstation, or clone such as Solbourne.
    - TEK: Tektronix workstation.
    - VAX-VMS: VAX running VMS.

- INTFMT, string: The format used to represent integer pixels (BYTE, HALF, and FULL) in the file. If INTFMT is not present, it defaults to LOW. Note that INTFMT should be present even if the pixels are a floating-point type. The valid values are:

    - HIGH: High byte first, big endian. Used for all other hosts (except for Cray, which is unimplemented).
    - LOW: Low byte first, little endian. Used for hosts VAX-VMS and DECSTATN.

- REALFMT, string: The format used to represent floating-point pixels (REAL, DOUB, and COMP) in the file. If REALFMT is not present, it defaults to VAX. Note that REALFMT should be present even if the pixels are an integral type. The valid values are:

    - IEEE: IEEE 754 format, with the high-order bytes (containing the exponent) first. Used for all other hosts (except for Cray, which is unimplemented).

  - RIEEE: Reverse IEEE format. Just like IEEE, except the bytes are reversed, with the exponent last. Used for host DECSTATN only.
  - VAX: VAX format. Single precision is in VAX F format, double precision is in VAX D format. Used for host VAX-VMS only.

- BHOST, string: The type of computer used to generate the binary label. It can take the same values with the same meanings as HOST. The reason BHOST is separate is that the data in the binary label may be in a different host representation than the pixels.
- BINTFMT, string: The format used to represent integers in the binary label. It can take the same values with the same meanings as INTFMT. The reason BINTFMT is separate is that the data in the binary label may be in a different host representation than the pixels.
- BREALFMT, string: The format used to represent floating-point data in the binary label. It can take the same values with the same meanings as REALFMT. The reason BREALFMT is separate is that the data in the binary label may be in a different host representation than the pixels.
- BLTYPE, string: The type of the binary label. This is not a data type, but is a string identifying the kind of binary label in the file. It is used for documentation, and so application programs can process the binary label correctly, without having to be told what kind it is. BLTYPE is new, and is not yet used by any applications. It defaults to a null string. The valid values are maintained in a name registry by the VICAR system programmer, which will document the actual data layout for each BLTYPE. As of this writing, there are no names yet registered.

Example

The system label for a typical file is shown below. Although carriage returns have been inserted for clarity, none actually exist in the file.

```
LBLSIZE=1024  FORMAT='BYTE'  TYPE='IMAGE'  BUFSIZ=20480  DIM=3  EOL=0
RECSIZE=512  ORG='BSQ'  NL=512  NS=512  NB=1  N1=512  N2=512  N3=1
N4=0
NBB=0  NLB=0  HOST='VAX-VMS'  INTFMT='LOW'  REALFMT='VAX'
BHOST='VAX-VMS'  BINTFMT='LOW'  BREALFMT='VAX'  BLTYPE=''
```

Property Labels

Property labels describe properties of the image in the image domain. Property labels do not describe the physical layout of the image; that is handlded by the system labels. They do contain other current information about the file, such as the map projection used, a lookup table, or latitude/longitude information for the image.

Property labels are divided into named sets called properties. Each property is made up of zero or more label items that contain the actual property information. The name space for each property is independent, so the same label item keyword may be used in more than one property. Only one property of a given name may exist.

Property labels are located between the system and the history labels. They start with the first occurrence of the keyword PROPERTY, and end with the first occurrence of the keyword TASK or the end of the labels (if there are no history labels). It is quite possible that no property labels exist in a file, in which case there would be no PROPERTY keywords.

Each property begins with a PROPERTY keyword, which has a string value. This value is the name of the property set. The PROPERTY keyword is followed by the label items that make up the property. The set continues until the next PROPERTY keyword, or the end of the property labels.

Label items within a property must not use the keywords DAT_TIM, LBLSIZE, PROPERTY, TASK, or USER.

Any program attempting to read a VICAR file should be able to accept (and ignore) properties or property label items it doesn't understand, as new properties and label items are added from time to time. A simple display program could ignore the property labels completely.

The valid property names, and the keywords that make up each property, are defined in a name registry maintained by the VICAR system programmer. As of this writing, no names have been registered, as property labels are a recent addition to VICAR files.

Example

Below is an example of what a property label with two properties might look like. **IMPORTANT: This is not a real property label! The property names and items shown below have not been standardized. This is an example only!** Also, carriage returns have been inserted for clarity, and do not exist in the label.

```
PROPERTY='MAP'  PROJECTION='mercator'  LAT=34.2  LON=177.221
PROPERTY='LUT'  RED=(1,2,3,4,5,6,7,8)  GREEN=(8,7,6,5,4,3,2,1)
BLUE=(1,1,1,3,5,7,8,8)
```

History Labels

History labels describe the processing history of the image. Each processing step has an entry (called a task) in the history label. Each task can optionally have label items further describing the task (such as parameters to the program). History labels do not describe the physical layout of the image; that is handlded by the system labels. They should contain only historical information; however, they often contain current state information that should be in a property label, since property labels are new and not yet well utilized.

History labels are divided into sets called tasks. Each task is made up of three mandatory label items, and zero or more label items that contain additional history information. The name space for each task is independent, so the same label item keyword may be used in more than one task. Each task has a task name associated with it, which is the name of the program that created that part of the history label. However, the task names are not unique. Several tasks may have the same name. Each occurrence of the task name is called an instance, so the task name and the instance combine to uniquely identify the task set.

History labels are located after the system and the property labels. They start with the first occurrence of the keyword TASK, and end with the end of the labels. It is possible, although highly unlikely, that no history labels exist in a file, in which case there would be no TASK keywords.

Each history task begins with a TASK keyword, which has a string value. This value is the name of the task. The instance is derived by counting the number of previous TASK keywords with the same task name; it is not stored explicityl in the label. The TASK keyword is followed by a USER and a DAT_TIM keyword. USER is a string specifying the username of the account that ran the program. The machine the program was run on is not available; it is assumed that the username is enough to identify the user. DAT_TIM is a string specifying the date the program was run, in the format Www Mmm dd hh:mm:ss yyyy, where Www is the three-letter day of the week, Mmm is the three-letter month, and the rest are digits. Time is in 24-hour format, and day of the month (dd) must be two digits (although the first may be a blank instead of a zero).

Following the USER and DAT_TIM keywords are the optional label items with further history information. The task set continues until the next TASK keyword, or until the end of the labels. The actual contents of the additional keywords are application-defined.

Label items within a task must not use the keywords DAT_TIM, LBLSIZE, PROPERTY, TASK, or USER.

Any program attempting to read a VICAR file should be able to accept (and ignore) tasks or task label items it doesn't understand, as new tasks and label items are added frequently. A simple display program could ignore the history labels completely.

Example

Below is an example of a typical history label with several tasks in it. Although carriage returns have been inserted for clarity, none actually exist in the file.

```
TASK='GEN'  USER='RGD059'  DAT_TIM='Thu Sep 24 17:31:50 1992'  IVAL=0.0
SINC=1.0  LINC=1.0  BINC=1.0  MODULO=0.0  TASK='COPY'  USER='RGD059'
DAT_TIM='Thu Sep 24 17:31:54 1992'  TASK='LABEL'  USER='RGD059'
DAT_TIM='Thu Sep 24 17:32:54 1992'  TASK='F2'  USER='RGD059'
DAT_TIM='Thu Sep 24 17:33:07 1992'  FUNCTION='in1+10'  TASK='STRETCH'
USER='RGD059'  DAT_TIM='Thu Sep 24 17:33:55 1992'
PARMS='AUTO-STRETCH:      0 to      0 and    138 to    255'
```

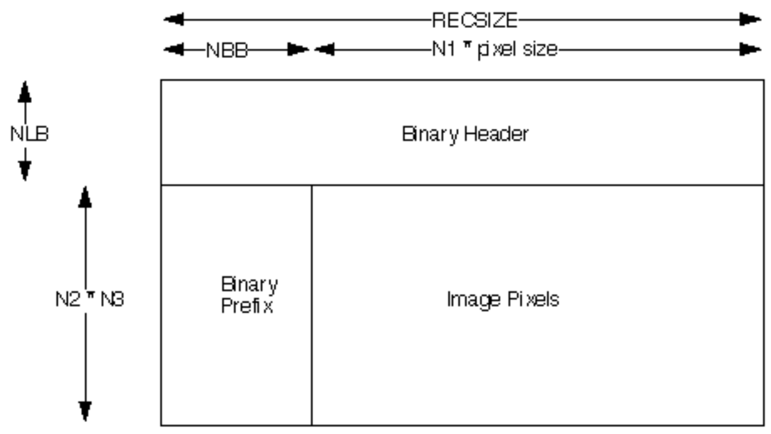
Image area

Following the labels (or between the label parts if there are EOL labels) is the image area. The structure and content of the image area are described in this section.

Image Organization

The image area is made up of records RECSIZE in length. Each record contains one line of data (for BSQ), i.e. one set of N1 pixels, plus the binary prefix, if any. If NBB=0, the binary prefix does not exist. A set of N2 records comprises a band (for BSQ), and a set of N3 bands makes up the image. The image is optionally preceded by NLB records of binary header. If NLB=0, the binary header does not exist.

The structure of the image area is shown below.



Pixel Types

Image pixels are always represented in a binary format, not in ASCII. This makes reading and writing the image more efficient, but makes it harder to transfer images between different machines. The pixel representation is determined by two factors: the data type (FORMAT label), and the host representation (INTFMT and REALFMT labels). For binary labels, the data type is application-defined, while the host representation is specified in the BINTFMT and BREALFMT labels (which may be different than INTFMT and REALFMT).

Data is typically stored in the native host representation for whatever machine the image was created on. Any program that reads a VICAR file *must* be able to translate that representation into the one used by the machine the program is running on. A program that writes a VICAR file does not need to do translation; it can write out in the native format (although it can translate if desired). The VICAR Run-Time Library typically performs the input translation automatically.

The integer data types are BYTE, HALF, and FULL. In both currently supported INTFMTs (HIGH and LOW), BYTE is a single-byte, unsigned value in the range 0-255. HALF is a two-byte, two's-complement signed value in the range -32768 - +32767, while FULL is a four-byte, two's-compelement signed value in the range -2147483648 - +2147483647. For INTFMT=HIGH, the high-order byte is first for HALF and FULL, while for INTFMT=LOW the low-order byte is first and all the bytes are swapped (i.e. 4321 instead of 1234). The representations for BYTE are identical in HIGH and LOW.

The floating-point data types are REAL, DOUB, and COMP. Type COMP is for complex numbers, and consists of two REAL numbers in the order (real, imaginary). There are three currently supported REALFMTs, IEEE, RIEEE, and VAX. IEEE is the IEEE-754 standard floating point format. REAL is a single-precision value, while DOUB is a double-precision value. See the IEEE-754 documentation for a definition of the standard. RIEEE is exactly like IEEE, except the bytes are stored in reverse order (as in HIGH vs. LOW, so they are in the order 4321 or 87654321 instead of 1234 or 12345678). VAX is the floating-point format used by Digital Equipment Corp.'s VAX series of computers. REAL is stored in VAX F floating-point format, while DOUB is stored in VAX D floating-point format. See the documentation provided by DEC for a definition of the floating-point formats.

Binary Labels

Binary labels are the least well-defined part of the VICAR file format. Binary labels consist of two parts: binary headers, which occur once at the top of the file, and binary prefixes, which occur before every image record. For most purposes, especially for simple display programs, binary labels can be ignored. Most of the time, they are not even present.

The data types and semantics of information in the binary label are defined by the application programs that use them. The user has to know what kind of binary label is present in order to use the correct application to make use of it. An attempt has been made to solve this problem through the addition of the BLTYPE system label item, but it is new and not yet in use at the time of this writing.

No attempt in this document is made to describe the various kinds of binary label. The documentation for the individual programs involved will describe the format used.

The data in a binary label is usually stored in a binary format, although an application could, of course, decide to store it in ASCII. All the binary label data must be in a single host representation, given by the BINTFMT and BREALFMT system labels. It is important to realize that the host for the pixels is not necessarily the same as the host for the binary labels, so make sure BINTFMT and BREALFMT are used instead of INTFMT and REALFMT for interpreting binary labels. As with pixels, any program that reads a binary label must be able to do host translation, while a program that writes a binary label does not have to translate (although some kinds of binary labels may be defined to be in VAX format for backwards compatibility).

Last Update: September 25, 1992

Questions about the VICAR file format can be directed to Bob Deen.

This document can be found on-line at: http://www-mipl.jpl.nasa.gov/vicar/vic_file_fmt.html

Operational Science Analysis

Document Review: CL 96-0842 on 22-May-96 by Charlotte Marsh