



VICAR Quick-Start Guide

Version 3.1

2018-10-04

Prepared by:

Walter Bunch

Robert Deen, VICAR Cognizant Engineer



Jet Propulsion Laboratory

California Institute of Technology

Pasadena, California

Copyright 2016 California Institute of Technology. Government sponsorship acknowledged.

Table of Contents

1. Introduction.....	5
1.1. What VICAR Is.....	5
1.2. What VICAR Isn't.....	5
1.3. What this Guide Is.....	6
1.4. Brief History of VICAR	6
1.5. VICAR File Format	7
1.6. Users of VICAR.....	8
1.6.1. Historic.....	8
1.6.2. Current.....	8
1.7. Components of VICAR in this Release	9
1.8. Motivation for Release.....	9
1.9. Obtaining VICAR	10
1.10. Supported Platforms.....	11
2. Getting Started with VICAR.....	12
2.1. Documentation Status	12
2.1.1. General Guides	12
2.1.2. VICAR User's Guide.....	12
2.1.3. VICAR File Format	12
2.1.4. VICAR Run-Time Library Reference Manual.....	13
2.1.5. VICAR Porting Guide.....	13
2.1.6. Building and Delivering VICAR Applications	14
2.1.7. Application Program Help (PDF files)	14
2.2. Building and/or Installing VICAR.....	15
2.3. Starting up VICAR.....	15
2.4. Simple Aliveness Test.....	16
2.5. Shell VICAR Syntax	19
2.5.1. Pathname.....	20
2.5.2. Subcommands	20
2.5.3. Positional and key=value Parameters	20
2.5.4. Keywords	21
2.5.5. Multivalued Parameters	21
2.5.6. Strings and Quoting	21
2.5.7. Output Parameters.....	22
2.5.8. TCL Procedures.....	23
2.6. Xvd Image Display.....	23
2.6.1. Mac-specific X-windows configuration	23
2.6.2. Running xvd	23

VICAR Quick-Start Guide

2.7.	File Format Conversion	25
2.7.1.	Transcoder.....	25
2.7.2.	GDAL VICAR Plugin	26
2.8.	Most Important General VICAR Programs	27
2.8.1.	F2	27
2.8.2.	LABEL	27
2.8.3.	CFORM	27
2.8.4.	DIFPIC.....	27
2.8.5.	VICCUB.....	28
2.8.6.	STRETCH.....	28
2.8.7.	GEN	28
2.8.8.	SIZE	28
2.8.9.	FLOT	28
2.8.10.	HIST	28
2.8.11.	MAXMIN	28
2.8.12.	GETLAB.....	28
2.9.	Image Based Information System (IBIS)	28
3.	Getting Started with Development.....	30
3.1.	Building a Program.....	30
3.2.	Java	30
4.	Use Cases - Doing Something with VICAR	32
4.1.	Landsat 7 Mosaic	32
4.2.	SRTM Mosaic.....	36
4.3.	Landsat 8 Multi-Spectral Analysis	40
4.4.	Landsat 8 Pan Sharpening	45
4.5.	Using ISIS with VICAR to Process Galileo Europa Imagery	48
4.6.	Neptune's Satellite Proteus (1989N1)	53
5.	List of Programs.....	57
5.1.	Categories	57
5.1.1.	Utilities.....	57
5.1.2.	Displaying Images, Text, and Graphics	57
5.1.3.	Generic Tools	57
5.1.4.	Image Registration and Mosaicking	58
5.1.5.	Calibrating the Camera and Target	58
5.1.6.	Miscellaneous	58
5.1.7.	Multispectral Data	58
5.1.8.	Graphics and Tabular Data	58
5.1.9.	Project-Specific Programs.....	58
5.2.	Program Listing.....	59
5.2.1.	Utilities.....	59
5.2.2.	Displaying images, text, and graphics	60
5.2.3.	Generic tools.....	61
5.2.4.	Image registration and mosaicking	64

VICAR Quick-Start Guide

5.2.5.	Calibrating the camera and target	66
5.2.6.	Miscellaneous	67
5.2.7.	Multispectral data	67
5.2.8.	Graphics and tabular data	68
5.2.9.	Project-specific Programs.....	69
6.	<i>Acronym List</i>	71
7.	<i>References</i>	73

1. Introduction

1.1. *What VICAR Is*

VICAR stands for Video Image Communication and Retrieval. It is an image processing system developed by the (Multimission) Image Processing Lab (IPL or MIPL), at NASA's Jet Propulsion Laboratory (JPL).

VICAR has its origins in the mid-1960's (see the brief history, below), which makes it quite probably the oldest continuously used image processing system in the world.

VICAR was developed for use with JPL's planetary missions, from Surveyor in 1966 up to the present day. The majority of JPL's planetary missions that have cameras use VICAR in some way. This continues to this day, with Mars 2020 expected to make heavy use of it, just as MER, MSL, Phoenix, and Insight are.

Fundamentally, VICAR is a command-line-oriented system. It consists of about 350 application programs that run the gamut from trivial to highly complex. The power of VICAR comes in the way these applications can be combined together in scripts to do more complicated processing in a systematic way.

Another critical component of VICAR is its handling of metadata, or labels. Labels are pieces of metadata, in KEYWORD=VALUE format, that are attached to the image. They describe things about the image, such as the conditions under which the image was taken (e.g. temperature, pointing, mapping parameters) and the processing history of the image. These labels are first-class citizens in VICAR; they are almost as important as the image itself! They are what make a VICAR file a scientifically useful image, instead of just a pretty picture.

1.2. *What VICAR Isn't*

To be blunt, VICAR is not Photoshop. Although there are some GUI (Graphical User Interface) elements, notably the "xvd" display program, fundamentally VICAR is a command-line system, not a GUI. It does not have the glitz or interactivity of Photoshop. It is not anywhere near as easy to use.

If you want to make an image pretty, or enhance it in standard ways, use Photoshop. It is far better suited to experimentation with imaging techniques, and excels at improving how images look.

However, if you want to radiometrically or photometrically correct an image of a moon of Jupiter, or create a map of the surface, or do a variety of things that require maintaining precise scientific calibration of the data, VICAR is a better bet. It is designed specifically for this kind of work. VICAR is also well suited to systematic, production work, where you do the same thing to a whole set of images.

VICAR is also not ISIS (Integrated Software for Imagers and Spectrometers). ISIS is a package from USGS (US Geological Survey) that shares a lot of common roots with VICAR. While there are some

similarities, there are also differences - VICAR is better at some things, ISIS is better at others. If you want to work with Voyager or Galileo data, for example, use VICAR. One thing ISIS is very much better at is documentation - ISIS puts a priority on it, while VICAR has not (as discussed more below). You will have to put more effort into learning and using VICAR. We hope though that the effort will be rewarding, and worthwhile. Note that one of the use case examples in this manual involves using ISIS together with VICAR.

1.3. *What this Guide Is*

This Guide is intended to be an up-to-date, quick-start document that gets you pointed in the right direction. It is not a full-on user's guide.

Frankly, the system-level documentation for VICAR stinks. Most of it has not been updated in decades. While much of it is still accurate as far as it goes, there are a lot of newer features (for example, shell-VICAR) that are not discussed in the documentation. This Guide will try to plug those holes, pointing at what's good - or not - in the older documents while describing some of the newer features.

As bad as the system-level documentation is, the individual program documentation is generally pretty good, describing in detail what the programs do and how they do it.

1.4. *Brief History of VICAR*

Note to historians: the history in this section has been pulled together from several, sometimes contradictory sources. We have attempted to weave a coherent narrative but this should not be considered authoritative; go to the primary sources instead.

The seeds for image processing at JPL were sown in the early 1960's. Bob Nathan proposed image processing at JPL in 1962/3. By 1964/5, Fred Billingsley (the first person to publish using the word "pixel") and Roger Brandt had developed a Video Film Converter (digitizer), and Howard Frieden developed code to process Ranger data on an IBM 7094.

The first published reference to VICAR came in 1966. VICAR was written by Stan Bressler, Frieden, Nathan, Billingsley et al, for IBM 360 computers, based on experience with the previous work. The first documented use of VICAR was for Surveyor, again in 1966. The JPL Image Processing Lab was also formed at this time.

We believe, but cannot prove, that this makes VICAR the oldest continuously used image processing system in the world. We celebrated its 50th birthday in 2016.

The first "Open Source" delivery of VICAR was in 1971. This was to an outfit called COSMIC, which was the clearinghouse for NASA software at the time. VICAR continued to be delivered in source code form until the mid-1990's, when growing concerns over ITAR made it harder to justify source code release.

The 1970's saw the introduction of interactive processing (on IBM/TSO), as well as the development of IBIS (Image-Based Information System), which is still a part of VICAR.

1984 was an important year in VICAR history. In that year VICAR was converted from IBM 360 computers, to VAX/VMS. The VICAR core was redesigned to support the VMS conversion. However, much of the application code survived the transmission, providing continuity of the code base. In addition, the VICAR file format was redesigned to its current state (this is sometimes called VICAR2, but more commonly the 2 is dropped).

This transition also saw adoption of the Transportable Applications Executive (TAE) from NASA-Goddard as the command-line parser, scripting language, and batch processor. (ISIS also adopted TAE and used it for several decades). TAE is still included as part of VICAR, although its use has declined precipitously in recent years.

Finally, IPL was reorganized to become the Multimission Image Processing Lab (MIPL), in recognition of the increasing number of missions supported by VICAR.

The early 1990's saw VICAR ported to Unix. Unlike the VMS transition, which was a hard-cut from IBM to VMS, this was a port, with both VMS and Unix being supported simultaneously for a long time. Nearly 20 flavors of Unix were supported at various levels in the 1990's and early 2000's; as the industry consolidated around Linux most of these were dropped (currently Linux, Solaris, and Mac OS X are the only supported operating systems).

The early 1990's also saw the introduction of "shell-VICAR", which allowed VICAR programs to be run directly from the Unix command line. This reduced reliance on TAE and opened up the entire world of Unix scripting languages (e.g. sh, csh, perl, python, ...).

The "xvd" display program was developed in 1994 by Bob Deen. This X-windows/Motif program swept aside all the older display technologies, and is still in active use.

The 2000's saw the porting of VICAR to Mac OS X (2004), as well as a new generation of Java-based display tools (notably Marsviewer, by Nicholas Toole, in 2003). The early 2000's also saw the introduction of the Java-based "transcoder" by Steve Levoe, used for metadata-preserving file format conversion.

2005 was the end of an era, as the last VMS machine was decommissioned. However, VMS had been dwindling in popularity for years before that.

Finally, in 2015 the VICAR core was again released as Open Source.

1.5. VICAR File Format

The VICAR file format is intentionally simple, designed to make it easy to process images. It consists of an ASCII header for the labels (in KEYWORD=VALUE format), followed by a simple raster of pixels, potentially with multiple bands (bands are often used for multispectral data, including simple RGB color). There are a few optional complexities (e.g. binary prefixes); these are addressed in the VICAR File Format [1] document.

The labels may be continued at the end of the raster, if there is not room at the beginning. This makes metadata handling very efficient. If the label expands beyond the allotted space, it can be continued at the end of the file, rather than rewriting the image to make more room.

Labels come in three categories. System labels describe the layout of the file itself, and are the same across all files. Property labels describe the current contents of the file. History labels contain information about what processing was done to the file.

VICAR files are uncompressed. This makes random-access reads and writes easy, as well as I/O through one's own code (not using the file access library). Interestingly, the industry has moved toward more and more compression while making disk space cheaper and cheaper. There is a quasi-experimental compression mode embedded in the I/O package but it is used only rarely.

VICAR supports very large files, much bigger than the 2GB typical of many formats. The only limit is that each *dimension* must be $< 2^{31}$ (~2 billion). VICAR files also support a wide range of data types: byte, short int (16-bits), long int (32-bits), float, double, and complex.

The VICAR file format is compatible with both PDS 3 and PDS 4. PDS is the Planetary Data System, used to archive mission data. Many missions have supplied data to PDS 3 in VICAR format, with detached or attached PDS labels: MSL, MER, Phoenix, Cassini, Galileo, Voyager, Magellan, MEX (HRSC), and many older missions. The PDS label skips over (ignores) the VICAR label, while VICAR is capable of skipping over an attached PDS label. This dual-label capability is very important; it means processing programs are still able to run on PDS-archived data.

Importantly, the simplicity of the VICAR format (as long as binary prefixes are not used) enables it to be compatible with the much more restrictive PDS 4 as well. It is one of the few image formats that PDS 4 will accept.

1.6. Users of VICAR

1.6.1. Historic

VICAR has been used with most JPL planetary missions that have a camera. From the Ranger, Surveyor, and Mariner series, to Voyager, Viking, Magellan, and Galileo, to Mars Pathfinder, VICAR played a major role. The primary exception has been the more recent Mars orbiters, where VICAR saw little use.

VICAR has also been used in other contexts as well. AVIRIS was an airplane-mounted camera, NEAT was a telescope-based asteroid tracker, and the Cartographic group at JPL used (and still does use) VICAR for Earth maps of Landsat, GOES, AVHRR, ASTER, Geosy, Meteosat, MODIS, Quickbird and Worldview data, among others.

1.6.2. Current

VICAR has a long history, but is very much an active system. Some of the current users are discussed here.

The biggest current users are the Mars surface missions. The MIPL ground image processing systems for the MER and MSL rovers are based entirely on VICAR. The recent Phoenix mission and upcoming InSight and Mars 2020 missions are similarly VICAR-based. This code is in the critical path for operations, creating stereo terrains and mosaics used to drive and operate the

rovers. Unfortunately, the Mars-specific code is not being included in the Open Source release at this time.

AFIDS (Automatic Fusion of Image Data System) is a state-of-the-art Earth mosaic/cartography system developed by JPL. It handles automated subpixel registration, orthorectification, and huge (> 2GB) mosaics. It integrates many open source tools with VICAR core processing. AFIDS makes extensive use of the GeoTIFF standard to aid in cartographic projections of image data. It also supports NITF (National Imagery Transmission Format), and thus sees extensive use by the Department of Defense. Efforts are underway to bring this capability to the planetary world.

Cassini uses VICAR for telemetry processing, data validation and analysis. Users also do mapping, photometric analysis, and navigation (pointing correction) using VICAR.

DLR Berlin uses VICAR extensively, for HRSC (Mars Express), VMC (Venus Express), ISS and VIMS (Cassini), and Dawn framing camera, and for stereo processing of LROC (LRO), MDIS (Messenger), and OSIRIS (Rosetta).

The PDS Rings node used VICAR for reprocessing of Voyager data. A different team is currently proposing other Voyager reprocessing using VICAR to the NASA PDART (ROSES) call.

VICAR is also used for Earth processing, including classification/segmentation, change detection, large mosaics, multi-band processing detecting thermal anomalies, and cloud detection using various instruments.

As described above, the PDS Data Archive holds extensive collections of data in VICAR format.

1.7. *Components of VICAR in this Release*

The following represents some of the major components included in the Open Source release.

- Almost 350 application programs (see Section 5 for a list)
- Command-line parsing (shell-VICAR) and optional environment (TAE)
- VICAR-format Image I/O library, in both C/C++/Fortran, and Java versions
- "xvd" image display program
- File format conversion utility ("transcoder"), which converts between most common file formats (including VICAR, PDS, ISIS, and FITS, as well as industry standards like JPEG, PNG etc), and preserves metadata (at least for some conversions).
- A VICAR GDAL plugin that enables GDAL processing of VICAR files.
- IBIS (Image-Based Information System) for handling large tabular data sets
- Java-based JadeDisplay image display library and JADIS stereo image display library (both already open source'd separately, but included here).

1.8. *Motivation for Release*

There are several reasons for VICAR being released again in 2015 as open source.

VICAR had a long history of open source, up until the mid-1990's.

Almost all users or potential users want/need source code. We were negotiating deals with almost all users to get source code anyway. This was inefficient; a blanket authorization would be much easier.

There was no longer a need to keep it proprietary. ITAR has become somewhat more lenient of late, with most VICAR code clearly not covered. The parts that are questionable (such as telemetry processors) have been removed from the Open Source delivery.

JPL is encouraging Open Source much more now than before. It used to be very difficult to get approvals for release, and anything that was released had to go through Open Channel, which was not a convenient distribution mechanism. Now the process has been streamlined, and modern venues like SourceForge and GitHub are now allowed for distribution.

VICAR is a grab bag. Some parts of it are sleek and modern and cutting-edge, used daily today. Other parts are old and creaky, haven't been touched in decades, and may or may not even work anymore. It is important to get code for older missions out there for posterity so that others can process it. The older missions are a treasure trove of data, but JPL does not have funding to work with that data. Providing the code gives other researchers that opportunity. Even if a piece of code doesn't work (say, due to a missing database), access to the source means the programs can be fixed, or algorithms can be extracted and used in other contexts.

Finally, VICAR does not have the user base it used to. Open Source is the only way to get any of that back.

This all crystallized during discussions at the First Planetary Data Workshop in Flagstaff in 2012. There we realized we had to do this; it simply took some time to pull it all together.

Note that at this time, we are delivering VICAR old-school: as a downloadable tarball. We are not supporting a collaborative SourceForge or GitHub kind of development environment. While that is something we are working toward, the reality is that if we waited for that to happen we'd never get the code released.

We do, however, request that you submit any changes or enhancements you make back to us, so we can include them in the next version of VICAR. We cannot guarantee to include all (or even any) changes, but we want to do as much as we can given our resource constraints. Assuming the changes don't break anything important, we would like to incorporate them back into the mainline code base for the next release.

1.9. *Obtaining VICAR*

The Open Source page for VICAR is:

http://www-mipl.jpl.nasa.gov/vicar_open.html

That page will tell you where the current repository is.

At first, we were handling the Open Source version in the traditional release manner: download a tarball which has everything, and do what you want with it (within the licensing terms of course). Now we also are providing the VICAR source via GitHub.

1.10. Supported Platforms

VICAR is officially supported on the following platforms:

- Linux (32-bits)
- Linux (64-bits)

That means we have done full regression and validation testing on it (or at least on the parts we use regularly).

In addition, VICAR is known to work on:

- Solaris 10 (well, it *was* known to work – we don't build Solaris anymore)
- Mac OS X (both natively and via a centos7 Docker image; this used to work in 32-bit Mac OS, but now only in 64-bit)
- Windows 10 (not the Home version, via a centos7 Docker image)

We simply don't have the resources to fully test these platforms. However, all tests that we *have* done, show it works.

Given that the entire package is *caveat emptor* - we make no warranty express or implied - then in reality all the platforms above can be considered supported.

2. Getting Started with VICAR

This section provides an overview of the available VICAR documentation, pointing out what is current and what is not. It then shows how to set up VICAR and do a simple aliveness test. Next is a brief overview of three important new areas not covered by the existing documentation: shell command line, image display with xvd, and the transcoder. It finishes up with a short description of the most important general-purpose VICAR programs.

2.1. *Documentation Status*

2.1.1. General Guides

As mentioned in the introduction, the VICAR documentation leaves much to be desired. This section will help you navigate what we have, and find the good bits.

2.1.2. VICAR User's Guide

The VICAR User's Guide [5] was written in 1994. It contains information about both the VMS and Unix versions of VICAR. Unix support was "new" at the time. There was no shell-VICAR concept yet, so TAE was the only command-line processor.

Still, it provides a reasonable description of how to use VICAR with TAE (which is still possible). If you concentrate on the Unix parts and ignore VMS, it is still valid as far as it goes.

However, it should be noted that it is generally far easier to write VICAR programs in a standard Unix scripting language (e.g. sh, csh, perl, python... there are many) and use standard Unix job control (background processing, cron jobs, etc) to run systematic jobs. TAE can be used, especially if you have heritage code, but we at MIPL rarely use it ourselves any more.

Note that tapes are no longer supported in VICAR.

2.1.3. VICAR File Format

This document [1], written in 1994/5, is still perfectly valid and current, with a few exceptions noted below.

The most important recent addition by far is the ability to skip over a PDS3 or ODL label in order to get at the VICAR label. This capability, added for MER, allows for dual-labeled files... one with a PDS3 or ODL label, followed by a VICAR label.

The VICAR I/O packages look for "PDS_VERSION_ID" or "ODL_VERSION_ID" at the start of a file (they are functionally equivalent; MER and PHX data use PDS_VERSION_ID while MSL uses ODL_VERSION_ID). If this is found, the PDS/ODL label is parsed just enough to look for a "^IMAGE_HEADER" keyword. The value is an integer followed by a unit. The unit can be either <BYTES> or <RECORDS>. If bytes, that many bytes are skipped from the beginning of the file. If records, then the "RECORD_BYTES" keyword is looked for, the values are multiplied together, and that many bytes are skipped.

Once these bytes are skipped, the file is treated exactly like any other VICAR file, starting at that point. The PDS/ODL label is never again referenced or read.

Note that there is NO support for writing these attached labels in VICAR; output files are always pure VICAR. These files can be created using the Transcoder (described later).

The second update is the list of supported platform names. For a current list see the declaration of `host_table` at the top of `rtl/source/xvhost.c`. Note that "JAVA" (HIGH, IEEE) is also supported even though it is not in that table.

The final recent addition is the possibility of compressed images. Compressed images are not really standard VICAR, but there is some support for them built in. If the `COMPRESS` keyword is present, the value describes the type of compression. Currently the only implemented types are `BASIC` and `BASIC2`, which are variants of simple run-length encoding (good for sparse data sets with lots of 0's).

Note however that support for compression is disabled by default; you must define `RTL_USE_COMPRESSION` to 1 in `rtl/inc/xvmaininc.h` before compiling to enable it.

There is a complete absence of documentation for compression; even the source code is not well documented. If you want to use compression, see `rtl/source/basic_compression.c`.

Compression is not supported and not recommended for use. It is mentioned here only because it exists.

2.1.4. VICAR Run-Time Library Reference Manual

The VICAR Run-Time Library is the C/C++/Fortran image I/O and parameter processing library. It is the true core of VICAR. The RTL Reference Manual [3] is up to date, with the exception of two new routines.

The routines `xvplabel/zvplabel` and `xvplabel2/zvplabel2` are new since the RTL Reference Manual was written. These write the program parameters out to the VICAR history label. They are quite important and `zvplabel()` is called in every Mars program in order to preserve parameters. It really should be called in *every* program at some point. The difference is that `zvplabel2()` writes out all parameters, while `zvplabel()` writes out only the non-defaulted (i.e. specified by the user) parameters.

For calling sequences for these routines, see the comments at the top of `rtl/source/xvplabel.c`.

2.1.5. VICAR Porting Guide

The VICAR Porting Guide [6] was written to help application programmers during the port from VMS to Unix. At the time, it also served as an update to the RTL Reference Manual. However, most of the still-relevant information has since been transferred to the RTL Reference Manual (especially in section 2, Programming Practice).

There may be some residual historical interest in the Porting Guide. In addition, there are a number of VICAR programs that were never ported to Unix due to perceived lack of need; if any of these were ever ported the Guide would be helpful. (These unported programs are not included in the VICAR Open Source release).

2.1.6. Building and Delivering VICAR Applications

This document [4] describes the application build system (vimake) and the packer (vpack, which packs source code into .com files - similar in concept to tar files).

The document is still up to date and useful as far as it goes. However, there are additional vimake commands that have been added since it was written. Most of these are LIB_* macros, but there are others.

The best source of documentation for these is the vimake templates themselves. If you come across an undocumented macro in an imake file, look at util/imake_unix.tmpl and util/imake.config. Search for the macro; the comments nearby should describe the purpose of the macro.

Note that the list of "external" libraries (meaning not developed by MIPL; these are accessed by the LIB_* macros) has been pared down greatly for the Open Source delivery. Only those external libraries needed for the Open Source code are included.

2.1.7. Application Program Help (PDF files)

Each VICAR application program has associated with it a .pdf file of the same base name (thus the program "label" has "label.pdf"). These files are **NOT** Adobe Portable Document File PDF's!!! They are plain text files.

In VICAR, PDF means Parameter Definition File. Unfortunately, Adobe chose the same name we had been using already for years.

The VICAR PDF files contain program-readable descriptions of each program parameter - data type, valid values, default, etc. They also - more importantly - contain the help for the program.

The PDF help has three sections. The first is overall program documentation. The second, starting with a ".level1" line, contains a short description of each parameter. The third, starting with ".level2", contains a complete description of each parameter.

In general, the PDF help is good, describing the program, its operation, algorithms, parameters etc. in detail. The PDF help should be the primary source of information for any given program.

However, many PDF's were written in the VMS days, so examples often use VMS file paths, etc. These should be easily translatable to Unix equivalents.

Many more PDF's were written before shell-VICAR. So almost all examples use TAE command-line syntax. See the discussion below about the shell syntax to translate these to work outside of TAE.

The PDF help is extracted into HTML as part of the build process, and this is included in the built VICAR tree in the `$V2TOP/html/vichelp` directory.

Note: PDF files come in two distinct flavors: "process" and "procedure" (distinguished by the first line in the file). You will interact mostly with process PDF's (which wrap application programs). See Section 2.5.8 for a discussion of procedure PDF's. Both contain help, however.

There is also an old command-line menu system that can help find programs. To access it, start up TAE (type "vicar") and then type "menu". The menu has not been kept up to date, but it may still be useful to some.

2.2. Building and/or Installing VICAR

VICAR is distributed with a collection of third-party libraries, called "externals." VICAR can be installed from pre-built VICAR and externals binaries. The VICAR source also can be built with the pre-built externals binaries.

For VICAR build and/or installation instructions, see the companion document `VICAR_build_3.0.pdf`. VICAR can be built and installed anywhere, but the rest of this document assumes `/usr/local/vicar/v3.0`

2.3. Starting up VICAR

VICAR requires a number of environment variables to run, even from the shell. These are set up by the `vicset1.csh` and `vicset2.csh` scripts.

VICAR is designed around the `csh` (or `tcsh`) shell. The startup scripts are all for `csh`. If you use a different shell for VICAR, you may need to write your own setup script to hand-set a few of the variables. This is not a supported configuration, but the best bet is to just try it and see what is needed.

Before running `vicset1/2` you have to tell it where the top of the VICAR tree is. This is the directory that contains "`vicset1.csh`". The actual name varies across distribution tars, and with git download. Obviously, insert this location in the `setenv` command below.

```
setenv V2TOP /usr/local/vicar/v3.0/vicar_open_3.0
source $V2TOP/vicset1.csh
source $V2TOP/vicset2.csh
```

The `v3.0` directory is where the source tar was unpacked and built, and also where the external directory was unpacked. See `VICAR_build_3.0.pdf` for details. Note that the `csh` files above are source'd rather than being executed. This is so they can set shell and environment variables which survive after the scripts are done.

Why are there two scripts? `Vicset1` is the primary one, and sets up environment variables and other things that are inherited by subshells. `Vicset2` sets up aliases, which are not inherited. Therefore, it is recommended that you put the following in your `~/.cshrc` file:

VICAR Quick-Start Guide

```
if ($?V2TOP != 0) then
    source $V2TOP/vicset2.csh
endif
```

That will ensure that subshells get the full VICAR environment, if it was set in the parent (without disturbing anything if you did not set up VICAR). However, it is not *required* that you do the above; most subshells do not need the aliases set up by vicset2.

If you want to set up VICAR by default in your .cshrc then the following is recommended:

```
if ($?V2TOP == 0) then
    setenv V2TOP /usr/local/vicar/v3.0/vicar_open_3.0
    source $V2TOP/vicset1.csh
    source $V2TOP/vicset2.csh
else
    source $V2TOP/vicset2.csh
endif
```

2.4. Simple Aliveness Test

After building VICAR, or installing a pre-built VICAR, the following will execute a small set of programs that test the basics of VICAR. While this is not even close to an exhaustive test, if these programs work then it is likely that the build generally succeeded. Lines starting with % are lines you type (without the %); the rest shows output.

This assumes you have done the VICAR setup in the previous section.

```
% $R2LIB/gen a
Beginning VICAR task GEN
GEN Version 6
GEN task completed
% $R2LIB/list a
Beginning VICAR task LIST
```

BYTE	samples are interpreted as								BYTE	data			
Task:GEN	User:rgd				Date_Time:Tue Jun 9 20:59:51 2015								
Samp	1	3	5	7	9								
Line													
1	0	1	2	3	4	5	6	7	8	9			
2	1	2	3	4	5	6	7	8	9	10			

VICAR Quick-Start Guide

3	2	3	4	5	6	7	8	9	10	11
4	3	4	5	6	7	8	9	10	11	12
5	4	5	6	7	8	9	10	11	12	13
6	5	6	7	8	9	10	11	12	13	14
7	6	7	8	9	10	11	12	13	14	15
8	7	8	9	10	11	12	13	14	15	16
9	8	9	10	11	12	13	14	15	16	17
10	9	10	11	12	13	14	15	16	17	18

% \$R2LIB/copy a b

Beginning VICAR task COPY

COPY VERSION 12-JUL-1993

% \$R2LIB/label -list b

Beginning VICAR task LABEL

LABEL version 15-Nov-2010

***** File b *****

3 dimensional IMAGE file

File organization is BSQ

Pixels are in BYTE format from a SUN-SOLR host

1 bands

10 lines per band

10 samples per line

0 lines of binary header

0 bytes of binary prefix per line

---- Task: GEN -- User: rgd -- Tue Jun 9 20:59:51 2015 ----

IVAL=0.0

SINC=1.0

LINC=1.0

BINC=1.0

MODULO=0.0

---- Task: COPY -- User: rgd -- Tue Jun 9 21:00:06 2015 ----

VICAR Quick-Start Guide

```
*****
```

```
% $R2LIB/list b
```

```
Beginning VICAR task LIST
```

```

    BYTE      samples are interpreted as    BYTE    data
Task:GEN      User:rgd      Date_Time:Tue Jun  9 20:59:51 2015
Task:COPY     User:rgd      Date_Time:Tue Jun  9 21:00:06 2015
    Samp      1          3          5          7          9
Line
    1          0   1   2   3   4   5   6   7   8   9
    2          1   2   3   4   5   6   7   8   9  10
    3          2   3   4   5   6   7   8   9  10  11
    4          3   4   5   6   7   8   9  10  11  12
    5          4   5   6   7   8   9  10  11  12  13
    6          5   6   7   8   9  10  11  12  13  14
    7          6   7   8   9  10  11  12  13  14  15
    8          7   8   9  10  11  12  13  14  15  16
    9          8   9  10  11  12  13  14  15  16  17
   10          9  10  11  12  13  14  15  16  17  18

```

```
% $R2LIB/gen c 1024 1024
```

```
Beginning VICAR task GEN
```

```
GEN Version 6
```

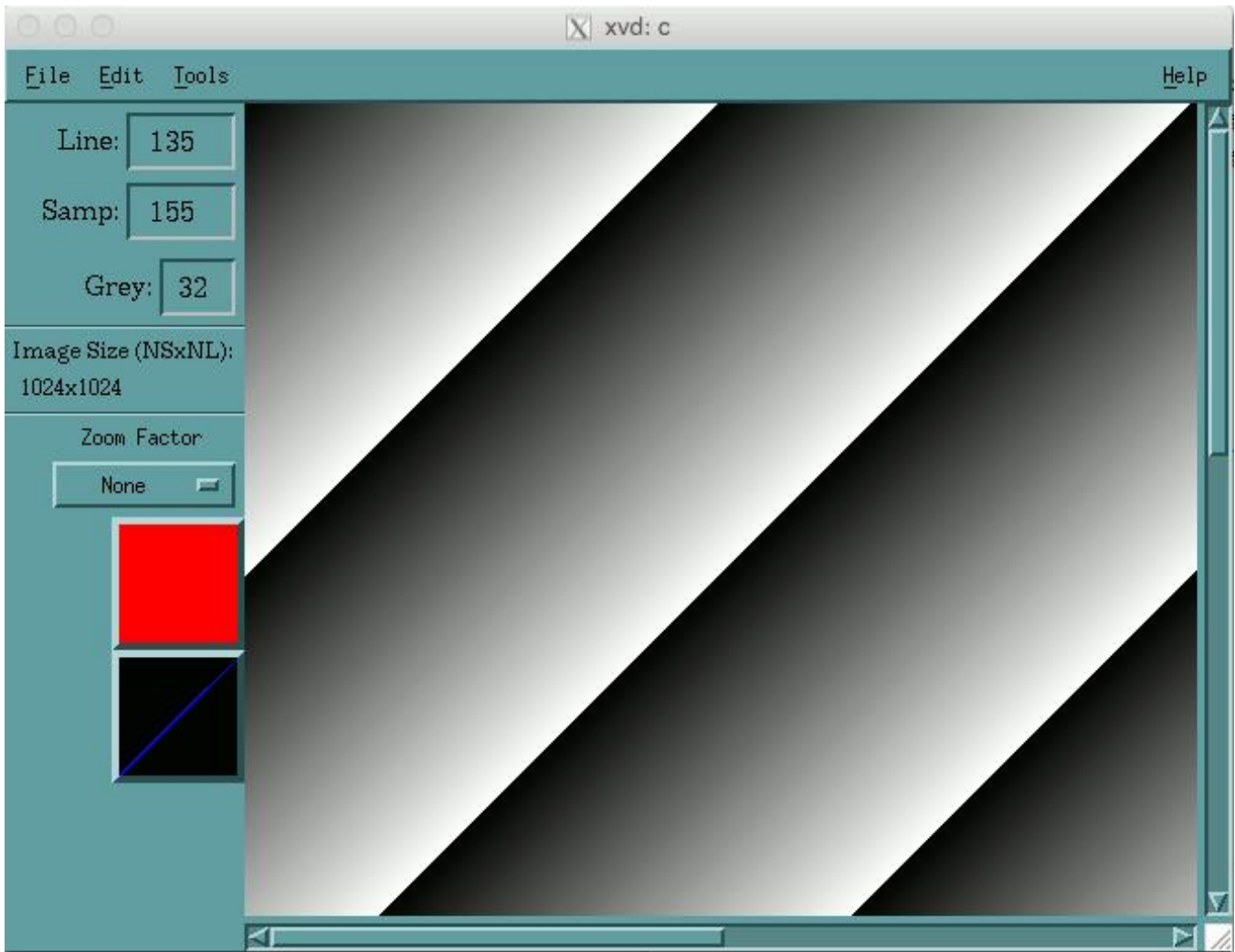
```
GEN task completed
```

```
% xvd c &
```

```
[1] 11255
```

```
%
```

The last command fires up the xvd image display program. It should display a diagonal ramp pattern looking like this:



If these commands do not work, check the build log for errors, and build again if necessary. If you continue to have problems, contact us and we will try to help - we do not have troubleshooting documentation yet.

One may infer from these examples that filename extensions are not required. Indeed that is the case: VICAR programs do not expect or enforce any filename convention. Any extension can be used, or none at all. Most of the time a .vic or .VIC extension is preferred to indicate it's a VICAR file, but sometimes .red/.grn/blu are used, or many other things. Many PDS holdings use .IMG, although this author's preference is to use .IMG for PDS-format files and .VIC for vicar.

2.5. *Shell VICAR Syntax*

As mentioned previously, shell-VICAR allows programs to be executed directly from the Unix shell, without needing TAE. This allows any normal Unix scripting language to be used with VICAR programs. ("Procedure" PDF's using the TCL language are handled differently; see Section 2.5.8).

This section describes how shell-VICAR syntax differs from TAE syntax. This will help translate examples in the PDF help, or the VICAR User's Guide. It also serves as a reference for how to

construct command lines. All the examples in here will run if typed in order as shown; the "gen" command generates files so no inputs are needed.

2.5.1. Pathname

TAE knows where to find programs automatically. Not so with the shell; you generally have to specify \$R2LIB/ (or other directory, but \$R2LIB is by far the most common) to run programs. It is certainly possible to put \$R2LIB in your \$path to remove this limitation, but we don't generally do so for fear of name collisions with the 350 VICAR applications and standard Unix utilities. It just seems safer to require the \$R2LIB.

```
$R2LIB/gen a
```

Almost all programs are in \$R2LIB. There are some completely unsupported programs in \$R3LIB - we do not test them or support them in *any* way. If you have other parts of the VICAR system there can be more directories, e.g. \$MARSLIB for the Mars programs or \$HWLIB for the DLR extensions to VICAR.

2.5.2. Subcommands

A few programs have "subcommands". The program LABEL is the primary one you will come across, but there are others. In TAE, you put the subcommand right after the command, e.g. label-list. In the shell, you put a space before the "-". So it looks like a Unix -keyword, but it has to be the first parameter.

```
$R2LIB/label -list a
```

2.5.3. Positional and key=value Parameters

All VICAR parameters can be specified using key=value, where key is the name of the parameter in the PDF. However, it is possible, in both TAE and the shell, to omit the key= for the first few parameters. These so-called positional parameters have to be in the same order as in the PDF. As soon as you want to skip a parameter, you have to go to the key=value form. Once you start key=value, you cannot go back to positional on the same command line.

Basically, positional parameters are just a shorthand for the most commonly used parameters. The first two parameters to almost all VICAR programs are INP and OUT.

The key in key=value need not be the entire parameter name in the PDF; it can be shortened as desired, as long as the name is unique. So if a program had parameters ORANGE and OFORM, these could be shortened to OR and OF if desired. Any truncated version of ORANGE could be used, as long as it doesn't become ambiguous with respect to other parameters defined for the program. An actual example is the program F2, which takes several parameters, including FORMAT and FUNCTION. It is common to find scripts using F2 that identify the FUNCTION parameter as just FUN.

```
$R2LIB/gen b 50 50
$R2LIB/copy b c sl=10 nl=20
$R2LIB/copy inp=b out=c sl=10 nl=20
```

Note that it is legal to have spaces on either side of the "=" if desired. This is very useful when dealing with very long filenames; you can say e.g. "inp= NL<tab>" and hit tab and let the shell's filename completion fill in the value for you. Without the space, it would look for a file starting with "inp=NL" which is not what you want; with the space it looks for files starting with "NL" which is what you want.

2.5.4. Keywords

Many programs have "keywords" (not to be confused with the parameter name in key=value). These keywords are parameters with a defined set of valid strings, generally used as flags. These parameters can be specified by key=value but they can also be specified by "-value", like Unix keywords. In TAE, keywords are indicated by an apostrophe before the name: 'value . You will see this a lot in examples, convert them to -value . Keyword names can also abbreviated as long as they remain unique.

```
$R2LIB/label -list b -dump
$R2LIB/gen d 10 10 -real ival=-1 linc=1 sinc=0
$R2LIB/list d -zero
```

2.5.5. Multivalued Parameters

Many VICAR parameters accept more than one value. In TAE, these multivalued parameters are enclosed in parentheses, e.g. irange=(-1,10) . In shell-VICAR, that's what the parser ultimately wants to see. However, parentheses have special meaning to the shell, therefore they must be quoted. This is most often done with backslashes, e.g. irange=\(-1 10\) , but can also be done with quotes: irange="(-1,10)" . Note that values can be separated by either spaces or commas, and spaces are allowed around the parentheses.

```
$R2LIB/cform d e irange=\(-1 10\) orange=\(0 255\) -byte
$R2LIB/gen a.red 1024 1024
$R2LIB/gen a.grn 1024 1024 linc=-1
$R2LIB/gen a.blu 1024 1024 sinc=-1 ival=128
$R2LIB/viccub \( a.red a.grn a.blu \) a.color
```

2.5.6. Strings and Quoting

String parameters can be very tricky due to shell quoting rules. If there are no special characters in the string, then it can be treated like a number with no special handling. But if it contains special characters or spaces, it can get tough.

The shell-vicar parser needs to see double quotes around strings containing spaces or special characters. That means the double quotes themselves have to be quoted. This is often done by putting the entire thing in single quotes outside the double quotes. It can also be done by escaping the double quotes. If the value itself has to have quotes (as is often the case with label -add) it can get really messy (see the last example below, which pops out of shell quoting in order to have a backslash-quoted single quote be part of the string itself... whew!)

VICAR Quick-Start Guide

```
$R2LIB/f2 e f func='"in1*2"'
$R2LIB/label -add f g item='"key=value test=1.5"'
$R2LIB/label -add g h item='"key=\' \'space value\' \' test=1.5"'
```

Note that if you see the message:

```
[TAE-POSERR] Positional values may not follow values specified by
name.
```

it often means the quotes were messed up somehow.

The trick with quoting is to think about what the shell-vicar parser itself needs to see, and then back up to what needs to be specified on the shell to get there.

2.5.7. Output Parameters

A few programs have output parameters. For example, getlab will return the value of a label item, which can be used by the script. Output parameters are written to a file specified by V2PARAM_FILE (by default a file in /tmp named with the process ID to avoid collisions). This file can then be accessed via the v2param program.

```
$R2LIB/label -list h
$R2LIB/getlab h test -real
v2param itm_name
1.5
set x = `v2param itm_name`
$R2LIB/getlab h key -string
v2param itm_name
space value
$R2LIB/getlab h key -string -inst itm_inst=1 itm_task=label
v2param itm_name
value
setenv NAME `v2param itm_name`
```

The shell variable x or environment variable NAME can then be used elsewhere in the script.

Note that when using v2param, the keyword you specify is the name of the parameter with type "name" in the PDF. So in the case of getlab, you always use v2param with itm_name; the actual parameter name you're getting is in the call to getlab.

```
$R2LIB/gen i 10 10
$R2LIB/maxmin i
more `v2param -file`
```

```
setenv MAX `v2param MAXIVAL`
```

2.5.8. TCL Procedures

PDF files come in two distinct flavors: "process" and "procedure" (distinguished by the first line in the file). The "process" PDF is used for application programs written in Fortran, C, or C++, and is the form we discuss mostly in this guide. The "procedure" PDF is a script, which calls other VICAR programs or scripts. The scripting language, called TCL (TAE Command Language) is defined by TAE and includes if/else, variables, and other usual scripting language features. Procedure PDF's are still in use (AFIDS uses them extensively), although they have been supplanted by standard scripting languages (shell, perl, python, etc) in most situations. The distinction is important in that VICAR procedures are more difficult to use from the shell; the user must invoke them using the "taetm" utility:

```
taetm -s "vicar command line"
```

Note that this is a TAE command line using TAE syntax rules, not shell-VICAR syntax rules. Also important is that the entire command line must look like one "word" to the shell, thus the quotes.

2.6. *Xvd Image Display*

The "xvd" program is a high-performance display program for VICAR and PDS 3 images. It is written in C++ using X-windows and Motif.

2.6.1. Mac-specific X-windows configuration

Xvd needs an X-windows server, which comes for free with Linux, but you have to obtain one for the Mac (at <http://xquartz.macosforge.org/>). The VICAR setup script vicset1.csh will prepend /usr/X11/lib/flat_namespace/ to your DYLD_LIBRARY_PATH, because we have found that some Mac configurations require it (to solve the "non-widget child" error). You also may need to add the following to your /etc/ssh/sshd_config to get X11 forwarding to work:

```
X11Forwarding yes

X11UseLocalhost yes
```

2.6.2. Running xvd

Running xvd is simple, as its location is put in \$PATH for you by vicset1.

```
xvd &
```

This will bring up a file selection window, allowing you to select a file to view.

More commonly, a filename can be given on the command line. This can be a single-band or multi-band (color) file. Alternatively, three files can be given, if the bands are separate:

```
xvd x.vic &
xvd x.red x.grn x.blue &
```

VICAR Quick-Start Guide

The trailing & puts the program in the background, freeing the shell window for other tasks.

There are several options that can be provided to xvd (before the filename):

- min x : Sets the minimum data range for a non-byte image
- max y : Sets the maximum data range for a non-byte image
- fullscreen : sends xvd into full-screen mode. Right-click brings up a menu, allowing you to get out of this mode.
- fit : Does a zoom to fit, making the image fit the window size
- width w : Sets the initial width of the window
- height h : Sets the initial height of the window
- x x : Sets the X position of the window
- y y : Sets the Y position of the window
- xrm resource : Sets an arbitrary Xrm resource string (see the XVd.xres resource file in \$GUILIB for examples)
- help : prints these options to the terminal

Of these, -min and -max are very commonly used, -fit is occasionally used, and the others are rarely used.

The xvd program is pretty self-explanatory and easy to use, so it is not described in detail here, beyond a few small items of note:

- Non-byte data is converted to byte for display using the data range. This is normally the minimum and maximum values in the image, but can be set with the File/Data Range menu or the -min/-max command line options. Stretches are applied *after* the conversion to byte.
- The magnifying glass and cursor stretch options initiate modes that are non-intuitive to get out of. Simply right-click (often command-click on a mac, depending on your X-windows setup) to bring up a pop-up menu that allows you to turn these off.
- If stretch does not seem to work, go to Edit/Preferences and switch to S/W Lookup Table. Some X-windows servers incorrectly advertise the capability to do a hardware stretch, which xvd pays attention to.
- Save As works in a somewhat non-intuitive way; rather than saving xvd's output, it actually calls VICAR programs to manipulate the data in the same way that xvd did. This generally works but can fail with certain types of files (notably, PDS 3 files that are not also VICAR files).

- The Help system and Print options likely will not work, as they are based on first-generation web browsers.

The xvd program supercedes the older VIDS image display system, which is based on the Virtual Raster Display Interface (VRDI). Both of these are still included in the VICAR delivery, but their use is not recommended.

2.7. File Format Conversion

2.7.1. Transcoder

The Transcoder is a powerful Java program that does conversion amongst many common file formats, and can preserve metadata. It is based on the Java Image I/O package, with additional plugins courtesy of VICAR for VICAR, PDS 3, ISIS 2, and FITS images. It also has the beginnings of PDS 4 support.

The transcoder is invoked using the rather awkward command:

```
java jpl.mipl.io.jConvertIIO
```

For most active missions we write wrapper scripts around this for common operations, but these scripts are not currently included with the delivery. An example of such a script, to convert vicar (or really, anything) to PNG, follows:

```
#!/bin/csh
#
# Simple script to convert vicar -> png.
#
set base = ${1:r}
java -Xmx3072m jpl.mipl.io.jConvertIIO inp=$1 out=${base}.png
format=png 2rgb=true oform=byte ri=true
```

Running it with no options prints a (long) help list. Describing every option is beyond the scope of this quick guide, but a few of the most important are described here.

The three most important are `inp=`, `out=`, and `format=`. Using these you can convert any known image format to any other, *without* preserving metadata. For example:

```
java jpl.mipl.io.jConvertIIO inp=file.vic out=file.png format=png
```

For a list of known formats, run it with "plugins" as the (only) argument. The list is quite extensive!

The `2rgb=true` argument will convert a single-band input file to color for those formats that are naturally color (such as jpeg).

Metadata-preserving transformations are controlled by an XSL stylesheet, that says how to convert the metadata between formats. How to write one is beyond the scope of this document,

but several are provided in \$V2TOP/java/jpl/mipl/io/xsl/. The most important of these are VicarToPDSmer*.xsl, VicarToPDSmsl*.xsl, and VicarToPDSphx.xsl. These convert from VICAR to PDS (3) format and are how we create the dual-labeled products for Mars surface operations and archive. Use the highest numbered one available.

For example, here is a script that will create the MSL dual-labeled files, along with a PDS 3 detached label (in the second call):

```
#!/bin/csh
#
# Simple script to transcode (vicar -> pds/odl) an image.
#
set base = ${1:r}
java -Xmx1024m jpl.mipl.io.jConvertIIO inp=$1 out=${base}.IMG
xml=false format=pds embed_vicar_label=true ri=true
xsl=$V2TOP/java/jpl/mipl/io/xsl/VicarToPDSmsl_Blob_ODL12.xsl
pds_label_type=ODL3
java -Xmx1024m jpl.mipl.io.jConvertIIO inp=${base}.IMG
out=${base}.LBL format=pds pds_detached_only=true ri=true
xsl=$V2TOP/java/jpl/mipl/io/xsl/PDSToPDSmsl_Blob_ODL2PDS_10.xsl
pds_label_type=PDS3
```

2.7.2. GDAL VICAR Plugin

GDAL (<http://www.gdal.org/>) is a library and collection of applications used to convert between various image formats, in particular geospatial formats. GDAL's plugin mechanism allows third parties to add support for new file formats. VOS includes a VICAR plugin for GDAL (see [vicar_open_3.0/vicar_gdalplugin/install](#)). If you add the location of the plugin shared library (gdal_vicar.so) to GDAL_DRIVER_PATH, or copy the plugin to your local GDAL installation's plugin directory, then GDAL should be able to find it. If you successfully install the plugin, you can verify it with:

```
gdalinfo --formats | grep -i vicar
```

Which should display:

```
VICAR -raster- (rw+): VICAR dataset (.img)
```

Note that the GDAL VICAR plugin (gdal_vicar.so) links against the VICAR run-time library (librtl.so) and the GDAL shared library (libgdal.so). So both should be found in your LD_LIBRARY_PATH.

2.8. *Most Important General VICAR Programs*

This section briefly describes some of the most important, commonly used, general VICAR programs. The classification as important is entirely the opinion of the author. It is not meant to imply that the other programs are not important! These are simply the programs that get used over and over in scripts and interactive processing.

See the program help for details; this section just points out the programs with a few examples.

2.8.1. F2

The F2 program does general math on an image, and is one of the most powerful generic VICAR programs. The function can be specified with either Fortran or C like syntax; the author generally uses the Fortran syntax. Some examples are below.

Subtract two images with a bias:

```
$R2LIB/f2 \ (a b\ ) c func=' "in1-in2+128"'
```

Subtract off the line number, but only where the value is non-0, and only on band 1. For Mars surface images, this converts a disparity image into a delta-disparity image.

```
$R2LIB/f2 a b func=' "(in1-line)*(in1.ne.0)"' nb=1 sb=1
```

Blank out a 100-pixel radius circle centered at 512,512:

```
$R2LIB/f2 a b func=' "in1*(sqrt((line-512)**2+(samp-512)**2).gt.100)"'
```

2.8.2. LABEL

Does label manipulation on an image. One of the few programs with subcommands. The -list subcommand is one of the most commonly used programs; it prints the label. The -add and -replace subcommands allow modification of the label.

2.8.3. CFORM

Converts data types. Very useful for converting halfword (16-bit integer) to byte in preparation for transcoding to a byte format such as jpeg or png. For example this converts a halfword image with a data range of 0-4095 to byte:

```
$R2LIB/cform a.vic a.vicb irange=\ (0 4095\ ) orange=\ (0 255\ ) -byte
```

2.8.4. DIFPIC

Computes a difference image for two input images. While F2 could be used to compute a difference image, DIFPIC also prints statistics about the differences. Even more statistics are printed if an output file is supplied.

2.8.5. VICCUB

VICCUB is a very simple program that takes 3 inputs and creates a single 3-band output. This is commonly used to create color images out of separate bands, or anaglyphs out of stereo images (using \left right right\) as input creates an anaglyph).

2.8.6. STRETCH

Does contrast enhancement (stretch) on an image. There are many different modes and options, including histogram-based stretches.

2.8.7. GEN

Generates VICAR files from scratch. Not much use in actual processing but (as can be seen from this document) very handy in test scripts and example code.

2.8.8. SIZE

This program resizes images, with or without interpolation.

2.8.9. FLOT

90 and 180 degree rotations and reflections of images.

2.8.10. HIST

Computes and prints histograms and other statistics.

2.8.11. MAXMIN

Computes the maximum and minimum pixel values in an image, and where they are. Notably, the values can be output for use in scripts (for example, setting a data range with cform).

```
$R2LIB/maxmin a
setenv MIN `v2param MINIVAL`
setenv MAX `v2param MAXIVAL`
```

2.8.12. GETLAB

Extracts label items from an image, returning them so they can be used in scripts. See example under "Output Parameters", above.

2.9. *Image Based Information System (IBIS)*

In 1975 Fred Billingsley and Nevin Bryant proposed that image processing technology could be used for registration and processing of multiple data planes over a geographic area. They created a comprehensive geographic information system, called IBIS, that allowed the integration of image data with tables of disparate geographic information. Their original system allowed for tables, graphics and images, but today IBIS only refers to the data table portion. These tabular data resemble a spreadsheet. IBIS files have VICAR labels and are described internally as

```
FORMAT='BYTE' TYPE='TABULAR'
```

VICAR Quick-Start Guide

IBIS works on rows and columns of data. Usually (but not always) columns of data have the same units (size, distance, velocity, geographic coordinates, etc) while rows of data refer to each element in the data set. So by setting up the relationships properly one can reference each cell to match some pixel, or set of pixels, in a corresponding image. By this, one can overlay important geographic inventory data on the image.

IBIS data can be floating point (single or double precision), integer or ASCII text. Internal descriptors are used to keep track of this. Tables can be arbitrarily large (millions of columns by millions of rows). Tables are allowed to have descriptive text headers.

IBIS allows the user to perform just about any mathematical operation on a column or row or any string operation if the data is text. Normally, these operations move data from one or more columns (or rows) to a new column (or row). IBIS tables can be expanded pretty arbitrarily to accommodate new data as development proceeds. It is also possible to extract data from one tabular data set and put it in a new tabular file or to merge it into an existing tabular file (with some limitations). Programs ACOPIN and VQUIC can transform any ASCII text file (with defined separators) into an IBIS Table. Through proper relationships, one can manipulate one or more columns (or rows) to create an output image file. Correspondingly, image data can also be transformed into an IBIS table.

Programs which support IBIS are listed in Section 5.2 below.

3. Getting Started with Development

VICAR is very much an environment in which to write image processing programs. Anything more than a cursory treatment is well beyond the scope of this document. The best suggestion is to look at other programs (generally, the newer the better) and follow their lead - program by example. Especially for the image I/O and parameter processing patterns.

In addition to the Run-Time Library (see the RTL Reference Manual [3]), which contains the core infrastructure for VICAR, there are a whole host of application-level subroutines in p2/sub (with a few in p1/sub). These are generally self-documenting, with help files included in the .com file package, or otherwise described by source-code comments.

If you make changes to VICAR, add capabilities, fix bugs, etc, we would like to hear about them! If possible, contribute the changes back to us and we will do our best to incorporate them in the next version of VICAR.

3.1. *Building a Program*

Building programs is described in the Building and Delivering VICAR Applications [4] document. Only the briefest outline is here.

Building a VICAR program is controlled by the imake file. This is a description of *what* to build, in the form of C preprocessor macro definitions. It does not say *how* to build it; that is the province of the vimake program. During the port from VMS to Unix, this scheme allowed the same build description to be used on both operating systems. The system still proves useful, as different platforms still need different compile options and commands.

This sequence will build a program (in this case gen) in the local directory:

```
cp -r $V2TOP/p2/prog/gen/* .
vimake gen
make -f gen.make
```

3.2. *Java*

There is a fair amount of Java code included with this delivery. There are build scripts in \$V2TOP/util/java* but in general, javac will just work for development of Java code. Or use an IDE.

The primary Java packages are:

io : Contains the transcoder and the image I/O plugins

jade : Contains JadeDisplay, which is the core display widget for Marsviewer. Also contains JADIS, a system for displaying Swing user interface components in stereo. Both packages have been delivered to Open Source previously; the pages below on the Open Channel Foundation contain useful documentation (which we have not yet brought back in to the Open Source delivery). Note that you need not obtain the code from OpenChannel as it is included here.

<http://openchannelfoundation.org/projects/JadeDisplay>

<http://openchannelfoundation.org/projects/JADIS>

mars : Contains classes to manage 3-dimensional vectors, and quaternions.

spice : Contains a Java Native Interface (JNI) wrapper around part of the NAIF/SPICE toolkit.

4. Use Cases - Doing Something with VICAR

4.1. *Landsat 7 Mosaic*

Note that this example may not work in the Solaris and MacOS platforms -- it should be in the next release.

Landsat data is available in GeoTIFF format from the USGS at <http://landsatlook.usgs.gov/>

The Landsat-7 panchromatic (band 8) images are provided as unsigned byte pixels. The script `lsatmos.pdf` below mosaics two band 8 images (from path 042, rows 035 and 036, acquired 2000-07-27) using the program `featherv`, which requires as input an IBIS table holding merely the names of the input files, where those files have GeoTIFF labels. `Featherv` also needs to know the starting line and sample of the output mosaic in the coordinate system of the first input image. The majority of this script is used to calculate that starting line and sample:

procedure

```
local topfile string
local botfile string
local minsl int
local minss int
local maxnl int
local maxns int
local curnl int
local curns int
local fcuro1 real
local fcuros real
local curol int
local curos int
local tval int
local nl int
local ns int
```

body

```
let topfile = "L72042035_03520000321_B80"
let botfile = "L72042036_03620000321_B80"
```

```
! convert GeoTIFF to VICAR
vtiff3o-to &"topfile".TIF &"topfile".vic
vtiff3o-to &"botfile".TIF &"botfile".vic
```

```
! get image dimensions length (nl) and width (ns)
lab2tcl &"topfile".vic v1=maxnl v2=maxns keyword=(nl,ns) 'system
lab2tcl &"botfile".vic v1=curnl v2=curns keyword=(nl,ns) 'system
```


VICAR Quick-Start Guide

```
! calculate offset between top and bottom
ibis-gen offset nc=8 nr=1 deffmt=DOUB
mf3 offset func="c1=0$c2=0"
pixmap (offset,&"botfile".vic) mapcols=(3,4) pixcols=(1,2) 'pixtomap
pixmap (offset,&"topfile".vic) mapcols=(3,4) pixcols=(5,6) 'maptopix
mf3 offset func="c7=c5+1$c8=c6+1"
ibis2tcl offset v1=fcurol v2=fcuros vartype=(-1,-1) ibisloc=(1,7,1,8)

! nudge offset line and sample
if (fcurol>0.0) let fcurol = fcurol+0.5
if (fcurol<0.0) let fcurol = fcurol-0.5
if (fcuros>0.0) let fcuros = fcuros+0.5
if (fcuros<0.0) let fcuros = fcuros-0.5
let curol = $fix(fcurol)
let curos = $fix(fcuros)

! calculate nl, ns
let minsl = 1
let minss = 1
if (curol<minsl) let minsl = curol
if (curos<minss) let minss = curos
let tval = curol+curnl-1
if (tval>maxnl) let maxnl = tval
let tval = curos+curns-1
if (tval>maxns) let maxns = tval
let nl = maxnl-minsl+1
let ns = maxns-minss+1

! create ibis file mosfile for featherv
ibis-gen mosrec1 nr=1 nc=6
format=("A99","FULL","FULL","FULL","FULL","DOUB") +
  data=(0,0,0,0,1.0) datacols=(2,3,4,5,6) +
  string=(&"topfile".vic) strcols=(1)

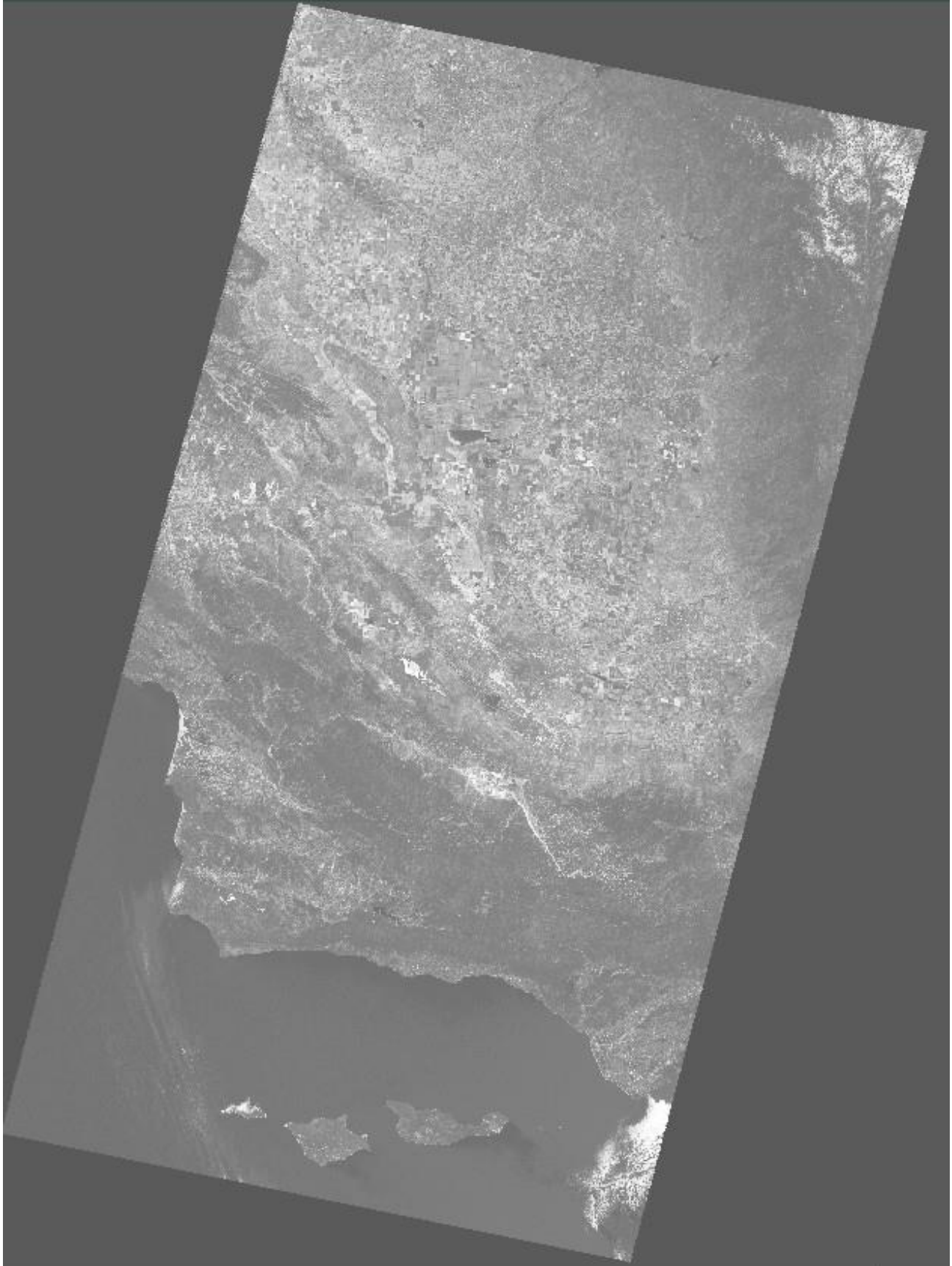
ibis-gen mosrec2 nr=2 nc=6
format=("A99","FULL","FULL","FULL","FULL","DOUB") +
  data=(0,0,0,0,1.0) datacols=(2,3,4,5,6) +
  string=(&"botfile".vic) strcols=(1)

ibis-cat (mosrec1,mosrec2) mosfile

! create the mosaic
featherv +
  inp=(&"topfile".vic,&"botfile".vic,mosfile,&"topfile".vic) +
  out=lsat.vic sl=&minsl ss=&minss nl=&nl ns=&ns +
  dfeather=700 moorefac=3 'factor 'progress 'noramp 'geotiff
```

```
end-proc
```

The mosaic lsat.vic, viewed in xvd, looks like this:



4.2. *SRTM Mosaic*

Note that this example is not functional in the Solaris and MacOS platforms -- it should be in the next release.

SRTM data is available as 1x1 degree images from the USGS at <http://earthexplorer.usgs.gov/>

The images are provided as 3601x3601 signed 16-bit headerless binary files. In this example, a 4x5 degree area (33N to 38N, -121W to -117W) is mosaicked. First, the images must be converted to VICAR format. A parameterized TAE script (srtmlabel.pdf) is used to add VICAR labels to the headerless image files:

```
procedure
  PARM inp TYPE=STRING COUNT=1
  PARM out TYPE=STRING COUNT=1
  PARM wlon TYPE=INT COUNT=1
  PARM slat TYPE=INT COUNT=1

  local wlonp1 int
  local slatp1 int

  body

    let wlonp1 = wlon+1
    let slatp1 = slat+1

    label-create &inp &out 3601 3601 'half

    gtgen inp=&out 'tiecnvrt 'rectfit +
      geotiff= ("ModelTiePointTag=(0,0,0,&wlon,&slatp1,0.0)", +
        "ModelTiePointTag=(3600,0,0,&wlonp1,&slatp1,0.0)", +
        "ModelTiePointTag=(0,3600,0,&wlon,&slat,0.0)", +
        "GTModelTypeGeoKey=2 (ModelTypeGeographic)", +
        "GTRasterTypeGeoKey=2 (RasterPixelIsPoint)", +
        "GeogEllipsoidGeoKey=7030 (Ellipse_WGS84) ")

  end-proc
```

Then a second script (srtmlabelall.pdf) is used to call srtmlabel.pdf on each of the 20 headerless files:

```
procedure
  body

    srtmlabel n33_w118_1arc_v2.bil n33_w118_1arc_v2.vic -118 33
```

VICAR Quick-Start Guide

```
srtmlabel n33_w119_1arc_v2.bil n33_w119_1arc_v2.vic -119 33
srtmlabel n33_w120_1arc_v2.bil n33_w120_1arc_v2.vic -120 33
srtmlabel n33_w121_1arc_v2.bil n33_w121_1arc_v2.vic -121 33
srtmlabel n34_w118_1arc_v2.bil n34_w118_1arc_v2.vic -118 34
srtmlabel n34_w119_1arc_v2.bil n34_w119_1arc_v2.vic -119 34
srtmlabel n34_w120_1arc_v2.bil n34_w120_1arc_v2.vic -120 34
srtmlabel n34_w121_1arc_v2.bil n34_w121_1arc_v2.vic -121 34
srtmlabel n35_w118_1arc_v2.bil n35_w118_1arc_v2.vic -118 35
srtmlabel n35_w119_1arc_v2.bil n35_w119_1arc_v2.vic -119 35
srtmlabel n35_w120_1arc_v2.bil n35_w120_1arc_v2.vic -120 35
srtmlabel n35_w121_1arc_v2.bil n35_w121_1arc_v2.vic -121 35
srtmlabel n36_w118_1arc_v2.bil n36_w118_1arc_v2.vic -118 36
srtmlabel n36_w119_1arc_v2.bil n36_w119_1arc_v2.vic -119 36
srtmlabel n36_w120_1arc_v2.bil n36_w120_1arc_v2.vic -120 36
srtmlabel n36_w121_1arc_v2.bil n36_w121_1arc_v2.vic -121 36
srtmlabel n37_w118_1arc_v2.bil n37_w118_1arc_v2.vic -118 37
srtmlabel n37_w119_1arc_v2.bil n37_w119_1arc_v2.vic -119 37
srtmlabel n37_w120_1arc_v2.bil n37_w120_1arc_v2.vic -120 37
srtmlabel n37_w121_1arc_v2.bil n37_w121_1arc_v2.vic -121 37
```

end-proc

Running `srtmlabelall.pdf` creates the 20 GeoTIFF labeled VICAR image files (*.vic). These are mosaicked together using a third script (`srtmmos.pdf`). It uses `gtappend` to mosaic five images at a time into four vertical columns. Then it uses `gtmss` to mosaic the four columns into a single image "srtm.vic":

```
procedure
body
```

```
gtappend inp=(n37_w121_1arc_v2.vic, +
              n36_w121_1arc_v2.vic, +
              n35_w121_1arc_v2.vic, +
              n34_w121_1arc_v2.vic, +
              n33_w121_1arc_v2.vic) out=w121.vic 'overlap1
```

```
gtappend inp=(n37_w120_1arc_v2.vic, +
              n36_w120_1arc_v2.vic, +
              n35_w120_1arc_v2.vic, +
              n34_w120_1arc_v2.vic, +
              n33_w120_1arc_v2.vic) out=w120.vic 'overlap1
```

```
gtappend inp=(n37_w119_1arc_v2.vic, +
              n36_w119_1arc_v2.vic, +
              n35_w119_1arc_v2.vic, +
              n34_w119_1arc_v2.vic, +
```

VICAR Quick-Start Guide

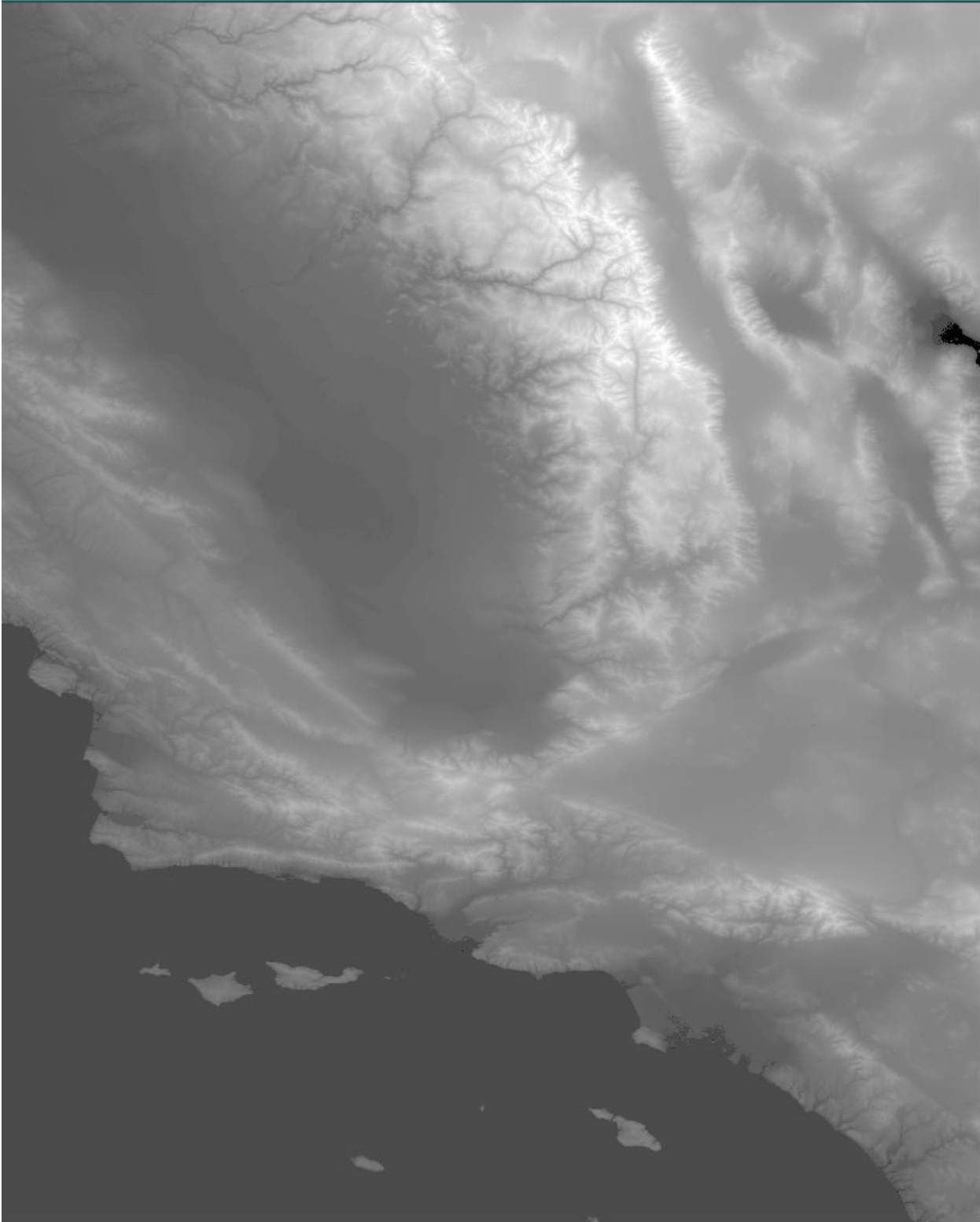
```
        n33_w119_1arc_v2.vic) out=w119.vic 'overlap1

gtappend inp=(n37_w118_1arc_v2.vic, +
              n36_w118_1arc_v2.vic, +
              n35_w118_1arc_v2.vic, +
              n34_w118_1arc_v2.vic, +
              n33_w118_1arc_v2.vic) out=w118.vic 'overlap1

gtmss inp=(w121.vic,w120.vic,w119.vic,w118.vic) out=srtm.vic 'overlap1

end-proc
```

When the 3601x3601 images are mosaicked, they overlap by one pixel. The resulting mosaic is a single-band half-word image with 18001 lines, and 14401 samples per line. Viewing it with xvd and using a Gaussian stretch yields:



4.3. *Landsat 8 Multi-Spectral Analysis*

Materials vary in their absorption, emission, and reflection as a function of frequency, so a multi-spectral image can be used to discern one material from another. Three interesting VICAR programs used with multi-spectral classification are sampler, clusterer, and classifier. Sampler performs sampling on input images and creates an output IBIS file (a table) storing the results. Clusterer performs K-Means clustering on an input IBIS file. Classifier classifies the pixels of the input images (used by clusterer to produce a cluster table) into the classes identified in the cluster table.

A Landsat 8 dataset was retrieved from <http://landsatlook.usgs.gov> (path 42, row 36, acquired 2016-10-03). The VICAR TAE script `msclassls8.pdf` below was used to convert the GeoTIFF format imagery into VICAR format and then generate and view various products. The script was executed from the shell command line with:

```
taetm -s msclassls8.pdf
```

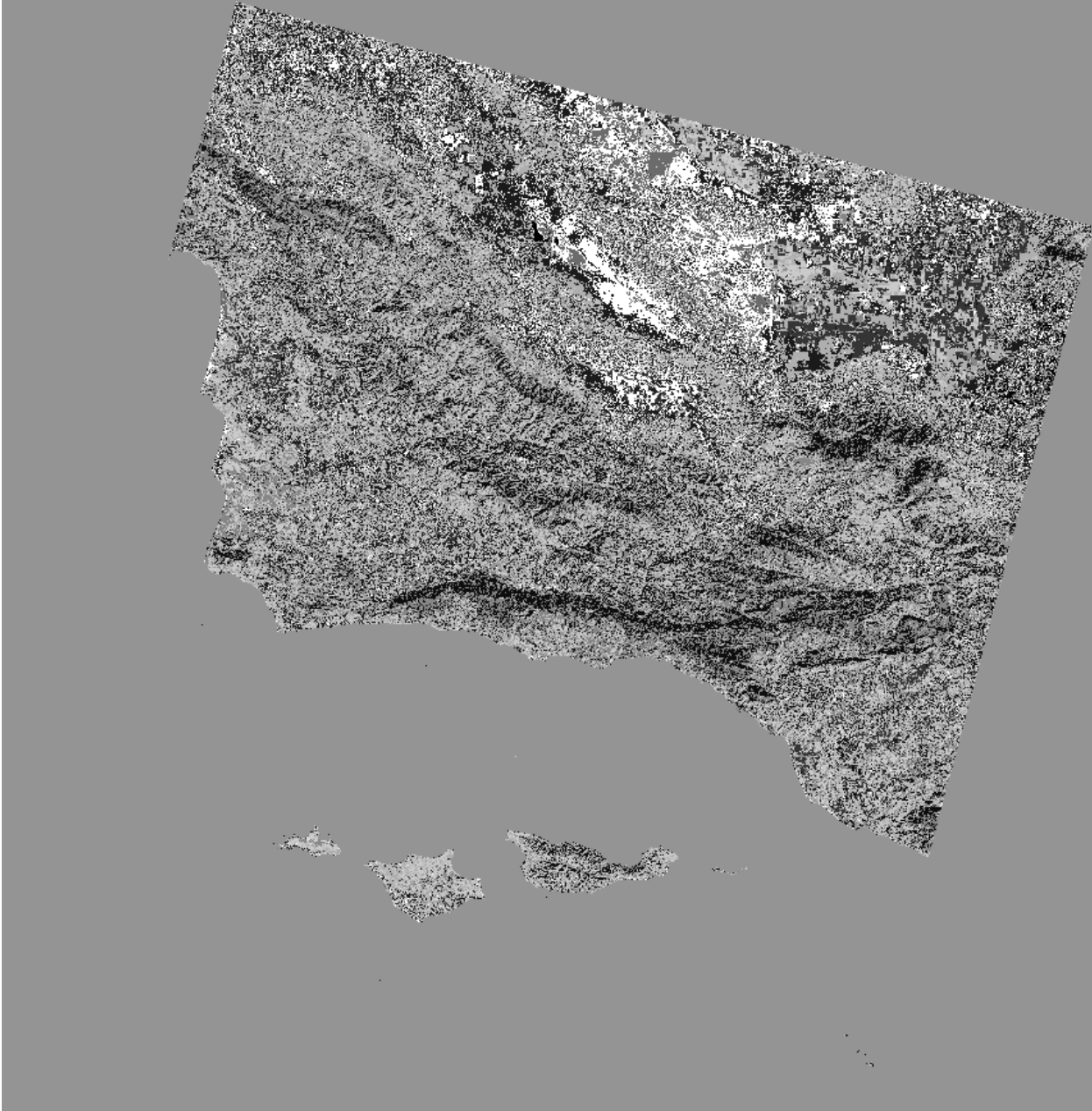
Note that this is a tutorial in VICAR, not multi-spectral analysis. A more rigorous approach to calculating vegetation cover is discussed at <http://grindgis.com/blog/vegetation-indices-arcgis>

For reference, bands 4, 3, and 2 were used to display this RGB image of the scene:



VICAR Quick-Start Guide

Sampler, clusterer, and classifier were used with bands 2 through 6 to generate this statistical class image. Each pixel DN value is a class number in the range 1 to 19, stretched for display here:



Following is the histogram of this image (calculated by hist):

1	79666	
2	2924741	*****
3	3867284	*****
5*	910302	*****
9*	753421	*****
11*	4601105	*****
12	37507636	***** 1
14*	4683045	*****

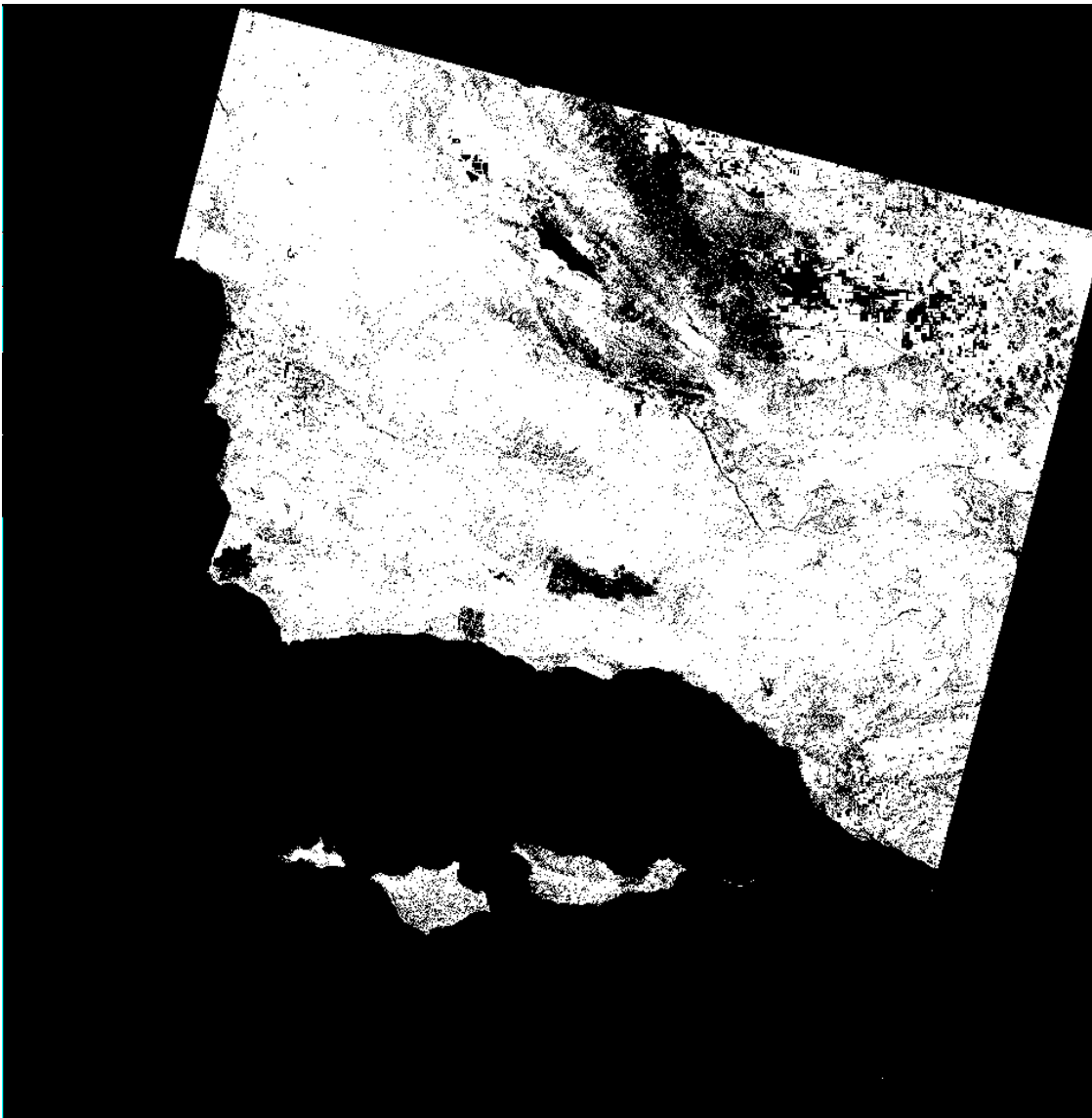
VICAR Quick-Start Guide

```
16* 4853719 ***** 2
19* 2398102 *****
```

Note that there are no pixels in bins 4, 6-8, 10, 13, 15, 16-18, and that bin 12 has the highest frequency, bin 16 the second highest. These 10 pixel classes correspond to some type(s) of landcover, based on the multi-spectral image values. Suppose we want to know which of these classes correspond to vegetation. We could visit the imaged areas to collect some ground truth. We also can use the Normalized Differential Vegetation Index (NDVI, see <http://grindgis.com/blog/vegetation-indices-arcgis>):

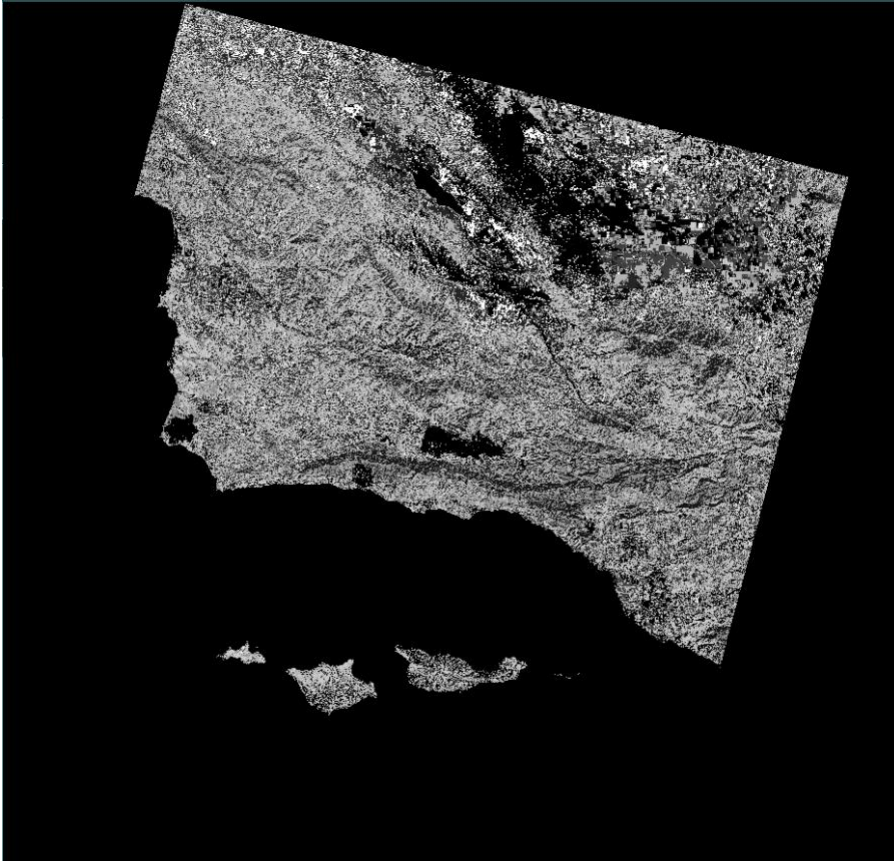
$$\text{NDVI} = (\text{NIR} - \text{RED}) / (\text{NIR} + \text{RED})$$

Using LS8 bands 5 and 4 respectively for NIR and RED, and thresholding the ratio above at 0.1 yields this binary NDVI mask:



VICAR Quick-Start Guide

Using this NDVI mask against the landcover class image yields



And the following histogram (preceded by the original for comparison):

1	79666		
2	2924741	*****	
3	3867284	*****	
5*	910302	*****	
9*	753421	*****	
11*	4601105	*****	
12	37507636	*****	1
14*	4683045	*****	
16*	4853719	*****	2
19*	2398102	*****	
0	41521335	*****	1
1	6430		
2	2484740	*****	
3	3348932	*****	
5*	910195	*****	
9*	229098	**	
11*	4181630	*****	
12	31045		
14*	4140327	*****	
16*	4220075	*****	2
19*	1505214	*****	

VICAR Quick-Start Guide

Bin 0 contains the masked pixels, largely by reducing class 12. Now classes 11, 14, and 16 seem to be most represented. Finally, here is the msclassls8.pdf script:

procedure

```
! Landsat 8 Bands and Wavelengths
! From http://landsat.usgs.gov/best\_spectral\_bands\_to\_use.php
! Band 1 - coastal aerosol          0.43 - 0.45
! Band 2 - blue                     0.45 - 0.51
! Band 3 - green                    0.53 - 0.59
! Band 4 - red                      0.64 - 0.67
! Band 5 - Near Infrared (NIR)      0.85 - 0.88
! Band 6 - Short-wave Infrared (SWIR) 1 1.57 - 1.65
! Band 7 - Short-wave Infrared (SWIR) 2 2.11 - 2.29
! Band 8 - Panchromatic             0.50 - 0.68
! Band 9 - Cirrus                   1.36 - 1.38
! Band 10 - TIRS 1                  10.60 - 11.19
! Band 11 - TIRS 2                  11.5 - 12.51
```

```
! Retrieved from http://landsatlook.usgs.gov/
local ls8b2 string init="LC80420362016277LGN00_B2"
local ls8b3 string init="LC80420362016277LGN00_B3"
local ls8b4 string init="LC80420362016277LGN00_B4"
local ls8b5 string init="LC80420362016277LGN00_B5"
local ls8b6 string init="LC80420362016277LGN00_B6"
```

body

```
! convert GeoTIFF to VICAR
```

```
vtiff3o-to &"ls8b2".TIF &"ls8b2".vic
vtiff3o-to &"ls8b3".TIF &"ls8b3".vic
vtiff3o-to &"ls8b4".TIF &"ls8b4".vic
vtiff3o-to &"ls8b5".TIF &"ls8b5".vic
vtiff3o-to &"ls8b6".TIF &"ls8b6".vic
```

```
! Process LS8 bands 2 through 6 with
!   sampler -> clusterer -> classifier
```

```
sampler inp=(&"ls8b2".vic,&"ls8b3".vic, +
             &"ls8b4".vic,&"ls8b5".vic,&"ls8b6".vic) +
          out=ls8samples.ibis +
          ngridx=200 ngridy=200 seed=30
```

```
clusterer ls8samples.ibis (ls8clusters.ibis, +
                           ls8classifiedsamples.ibis) +
          nclusters=20 conv=0.1 maxi=200
```

VICAR Quick-Start Guide

```
classifier inp=(&"ls8b2".vic,&"ls8b3".vic, +
  &"ls8b4".vic,&"ls8b5".vic,&"ls8b6".vic) +
  classibis=ls8clusters.ibis +
  out=(ls8classifiedResult.img, +
  ls8classifiedResultDist.img)

! Calculate LS8 Normalized Differential
! Vegetation Index
! NDVI = (NIR - RED) / (NIR + RED)
! See http://grindgis.com/blog/vegetation-indices-arcgis
f2 inp=(&"ls8b5".vic,&"ls8b4".vic) out=ls8ndvi.vic +
  func="( (in1-in2)/(in1+in2))>0.1" 'byte

! Mask the veg classes with the NDVI
f2 inp=(ls8classifiedResult.img,ls8ndvi.vic) +
  out=ls8classifiedMasked.img func="in1*in2"

! View RGB of LS8
xvd inp=(&"ls8b4".vic,&"ls8b3".vic,&"ls8b2".vic)

! View LS8 classified result
xvd ls8classifiedResult.img

! View LS8 NDVI
xvd ls8ndvi.vic

! View LS8 ndvi-masked veg
xvd ls8classifiedMasked.img

! Calculate histograms of the LS8 veg and ndvi-masked veg
hist ls8classifiedResult.img
hist ls8classifiedMasked.img

end-proc
```

4.4. Landsat 8 Pan Sharpening

Landsat 8 RGB bands have 30-meter resolution. The panchromatic band has 15-meter resolution. Pan-sharpening the RGB involves conversion from the RGB colorspace to the intensity, saturation, hue (ISH) colorspace, replacing the 30-meter intensity with the 15-meter pan channel, and converting back to RGB. The program RGB2ISH, which converts in either direction, requires all inputs to have the same number of lines and samples. So the 30-m saturation and hue channels need to be scaled up to 15-m before converting back to RGB.

Below are the Pan, original RGB, and sharpened RGB for comparison:



Panchromatic (band 8) Landsat 8 image of Point Magu NAS acquired 2016-10-03. Data retrieved from <http://landsatlook.usgs.gov/>



Original RGB (bands 4, 3, 2)



Pan-Sharpended RGB

The script (pansharp.pdf) used to pan-sharpen the RGB:

procedure

```
local ls8b2 string  init="LC80420362016277LGN00_B2"
local ls8b3 string  init="LC80420362016277LGN00_B3"
local ls8b4 string  init="LC80420362016277LGN00_B4"
local ls8b8 string  init="LC80420362016277LGN00_B8"
```

body

```
! convert GeoTIFF to VICAR
vtiff3o-to &"ls8b2".TIF &"ls8b2".vic
vtiff3o-to &"ls8b3".TIF &"ls8b3".vic
vtiff3o-to &"ls8b4".TIF &"ls8b4".vic
vtiff3o-to &"ls8b8".TIF &"ls8b8".vic

! convert RGB to ISH
rgb2ish inp=(&"ls8b4".vic,&"ls8b3".vic,&"ls8b2".vic) +
          out=(I,S,H)

! scale S and H up to pan resolution
gtsize inp=(S,&"ls8b8".vic) out=SP 'coverref
gtsize inp=(H,&"ls8b8".vic) out=HP 'coverref

! convert ISH to RGB using pan for I
rgb2ish inp=(&"ls8b8".vic,SP,HP) out=(R,G,B) reverse=1

! display pan-sharpened rgb
xvd (R,G,B)

end-proc
```

4.5. Using ISIS with VICAR to Process Galileo Europa Imagery

This is part of a larger procedure that performs a scattered light correction on color Galileo SSI images of Europa. The processing begins with VICAR data from PDS, converts to ISIS3, switches back to VICAR for some filtering, and then back to ISIS3. This assumes that you have ISIS installed. All the commands below are executed from the tcsh shell, in contrast with the VICAR/TAE shell.

1. Retrieve the Europa files from:

http://pds-imaging.jpl.nasa.gov/data/galileo/galileo_orbiter/go_0020/e14/europa/c044098/5000r.img
http://pds-imaging.jpl.nasa.gov/data/galileo/galileo_orbiter/go_0020/e14/europa/c044098/5000r.lbl

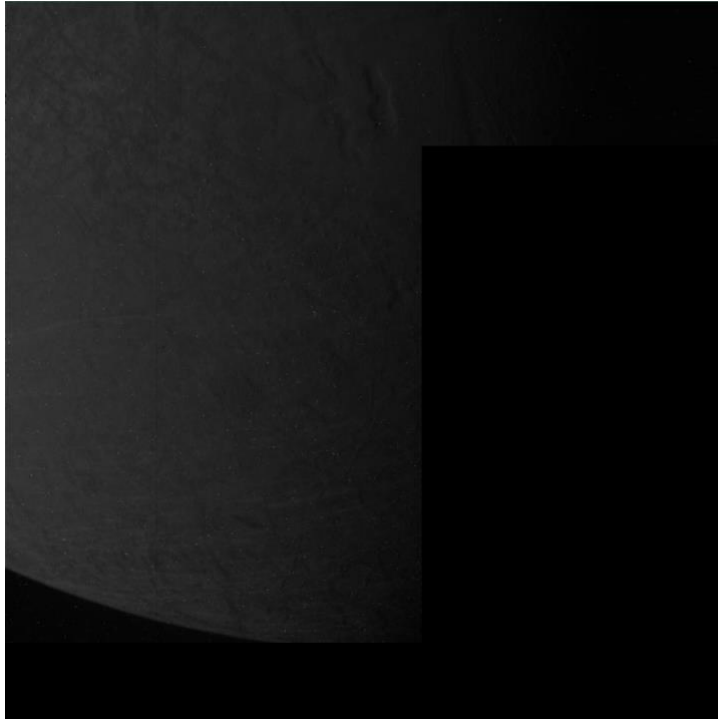
VICAR Quick-Start Guide

2. Define ISISROOT to be where ISIS is installed (contains 3rdParty, bin, config, doc, inc, ...).
3. Retrieve the ISIS Galileo SPICE data:

```
rsync -azv --delete --partial isisdist.astrogeology.usgs.gov::isis3data/data/galileo  
$ISISROOT/../../data
```

4. Use VICAR xvd to view the image:

```
xvd 5000r.img
```



5. Use ISIS gllssi2isis to import the PDS VICAR image to an ISIS cube:

```
gllssi2isis from=5000r.lbl to=5000r.cub
```

6. Use ISIS spiceinit to add camera pointing:

```
spiceinit from=5000r.cub
```

spiceinit updates the cub file and prints the following output:

```
Group = Kernels  
NaifFrameCode      = -77001  
LeapSecond         = $base/kernels/lsk/naif0011.tls  
TargetAttitudeShape = $base/kernels/pck/pck00009.tpc  
TargetPosition     = $base/kernels/spk/de405.bsp  
InstrumentPointing  = $galileo/kernels/ck/CKmerge_type3.plt.bck  
Instrument          = Null  
SpacecraftClock     = $galileo/kernels/sclk/mk00062b.tsc
```

VICAR Quick-Start Guide

```
InstrumentPosition      = $galileo/kernels/spk/s000131a.bsp
InstrumentAddendum      = $galileo/kernels/iak/ssiAddendum004.ti
ShapeModel              = Null
InstrumentPositionQuality = Reconstructed
InstrumentPointingQuality = Reconstructed
CameraVersion           = 1
End_Group
```

7. Use ISIS glssical to calibrate the image:

```
glssical from=5000r.cub to=5000r_cal.cub
```

glssical creates the calibrated image file and prints the following output:

```
Group = RadiometricCalibration
  From              = 5000r.cub
  DarkCurrentFile   = $galileo/calibration/darkcurrent/3f8.dc04.cub
  GainFile          = $galileo/calibration/gain/968f.cal04.cub
  ShutterFile       = $galileo/calibration/shutter/calibration.so02F.cub
  ScaleFactor       = 1.0
  OutputUnits       = I/F
  S1                 = 0.01486 <I/F per Ft-Lambert>
  RSUN               = 0.96103116591445 <(Planet-Sun range)/5.2 A.U.>
  Scale              = 1.0 <I/F units per DN>
  GC                 = 4.824 <Cube gain conversion>
  GG                 = 9.771 <Gain file gain conversion>
  IOF-SCALE0        = 0.006775822787906 <(S1/Scale)*(GC/GG)/RSUN**2>
End_Group
```

8. All of the special pixel values and negative pixel values will need to be removed. Use the ISIS stats command to get the range of pixel values and identify LRS and HRS pixel values.

```
stats from=5000r_cal.cub
```

stats prints the following output:

```
Group = Results
  From              = 5000r_cal.cub
  Band              = 1
  Average           = 0.212702254678
  StandardDeviation = 0.10454933103788
  Variance          = 0.010930562620467
  Median            = 0.22062550389977
  Mode              = 1.10941419996531e-04
  Skew              = -0.22735437357013
  Minimum           = 0.0
  Maximum           = 0.80783843994141
  Sum               = 79628.70497704
  TotalPixels       = 640000
  ValidPixels       = 374367
  OverValidMaximumPixels = 0
  UnderValidMinimumPixels = 0
  NullPixels        = 256095
```

VICAR Quick-Start Guide

```
LisPixels      = 0
LrsPixels      = 9538
HisPixels      = 0
HrsPixels      = 0
End_Group
```

9. Use the ISIS stretch command to remove the negative and special pixel values. Set the null pixels to 0 and hrs to the maximum value from stats above.

```
stretch from=5000r_cal.cub to=5000r_str.cub pairs="0:0 0.807838:0.807838" null=0 lrs=0 hrs=0.807838
```

stretch creates the cub file and prints the following output:

```
Group = Results
StretchPairs = "0.0:0.0 0.807838:0.807838"
End_Group
```

10. Remove ISIS label in preparation for conversion to VICAR:

```
isis2raw from=5000r_str.cub to=5000r.raw bittype=32bit
```

isis2raw creates the raw file and prints the following output:

```
Group = "DNs Used"
Null      = 0.0
LRS       = 0.0
LIS       = 0.0
HIS       = 0.39185168230716
HRS       = 0.39185168230716
ValidMin  = 0.0
ValidMax  = 0.39185168230716
End_Group
```

11. Copy the following VICAR script to a file named "fft_wiener.pdf":

```
procedure
body
```

```
! Import the image into VICAR format
```

```
label-create inp=5000r.raw out=5000r_new.img nl=800 ns=800 format=real
```

```
! Fourier transform the image.
```

```
fft22 in=5000r_new.img out=5000r_fft22.img format=comp
```

```
! Create the point spread function for each color filter used in the observation.
```

```
! Make sure the image size as the image mosaics (800x800).
```

```
gllpsf nl=800 ns=800 out=grn.psf filt=grn
```

```
! Fourier transform the point spread function.
```

```
fft22 in=grn.psf out=grn_fft22.psf format=comp
```

VICAR Quick-Start Guide

! Run the direct wiener filter on the image.

```
wiener inp=(5000r_fft22.img, grn_fft22.psf) out=5000r_direct.img option=direct
```

! Inverse fourier transform the wiener filtered image.

```
fft22 in=5000r_direct.img out=5000r_direct_fft22.img mode=inverse format=real
```

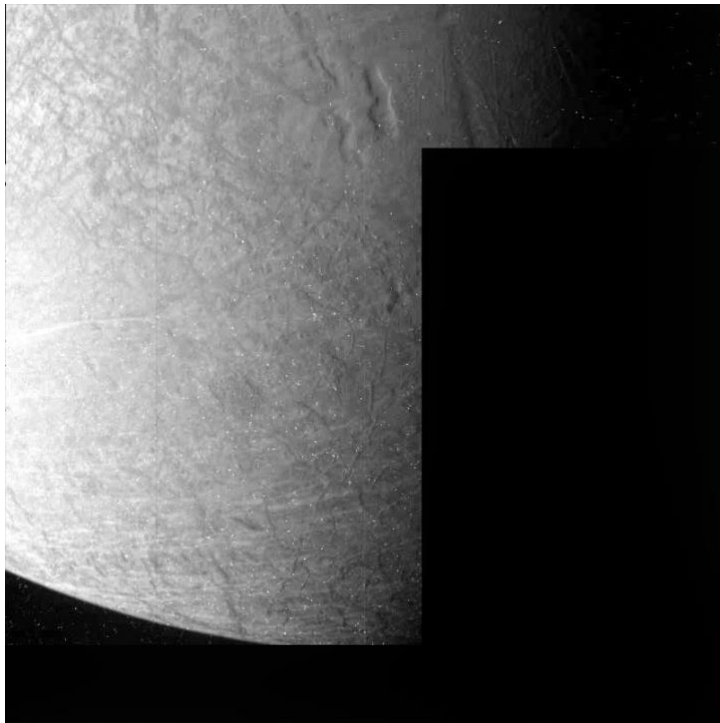
```
end-proc
```

12. Run the VICAR script to import and process the image in VICAR:

```
taetm -s fft_wiener.pdf
```

13. Use VICAR xvd to view the wiener filtered image:

```
xvd 5000r_direct_fft22.img
```



14. Use ISIS vicar2isis to import the image back into ISIS.

```
vicar2isis from=5000r_direct_fft22.img to=5000r_direct_fft22.cub bitttype=real
```

15. Use ISIS handmos to restore the pointing information to the new filtered isis3 image by mosaicking the wiener filtered image on top of the old image to preserve the image label.

```
handmos from=5000r_str.cub mosaic=5000r_labeled.cub nlines=800 nsamples=800 nbands=1  
create=y
```

```
handmos from=5000r_direct_fft22.cub mosaic=5000r_labeled.cub
```

4.6. *Neptune's Satellite Proteus (1989N1)*

We are enhancing the faint signal (and noise) above the background, then smoothing out that noise at the expense of spatial resolution followed by suppression of some smoothing artifacts.

This processing requires three input images:

- C1138920_RAW.IMG available at
http://pds-rings.seti.org/volumes/VGISS_8xxx/VGISS_8207/DATA/C11389XX/C1138920_RAW.IMG
- C1138920_RESLOC.DAT available at
http://pds-rings.seti.org/volumes/VGISS_8xxx/VGISS_8207/DATA/C11389XX/C1138920_RESLOC.DAT
- C1241506_CLEANED.IMG available at
http://pds-rings.seti.org/volumes/VGISS_8xxx/VGISS_8210/DATA/C12415XX/C1241506_CLEANED.IMG

The script performs this in five steps.

```
procedure
body
```

```
! Smooth over reseau marks (fiducial marks)
RESSAR77 (C1138920_RAW.IMG,C1138920_RESLOC.DAT) r

! Subtract camera dark image and crop
F2 (r,C1241506_CLEANED.IMG) nrsub (351,151,400,400) FUNC=IN1-IN2+8

! Stretch the remaining 4 DN
STRETCH nrsub nrstr LINE=(10,13)


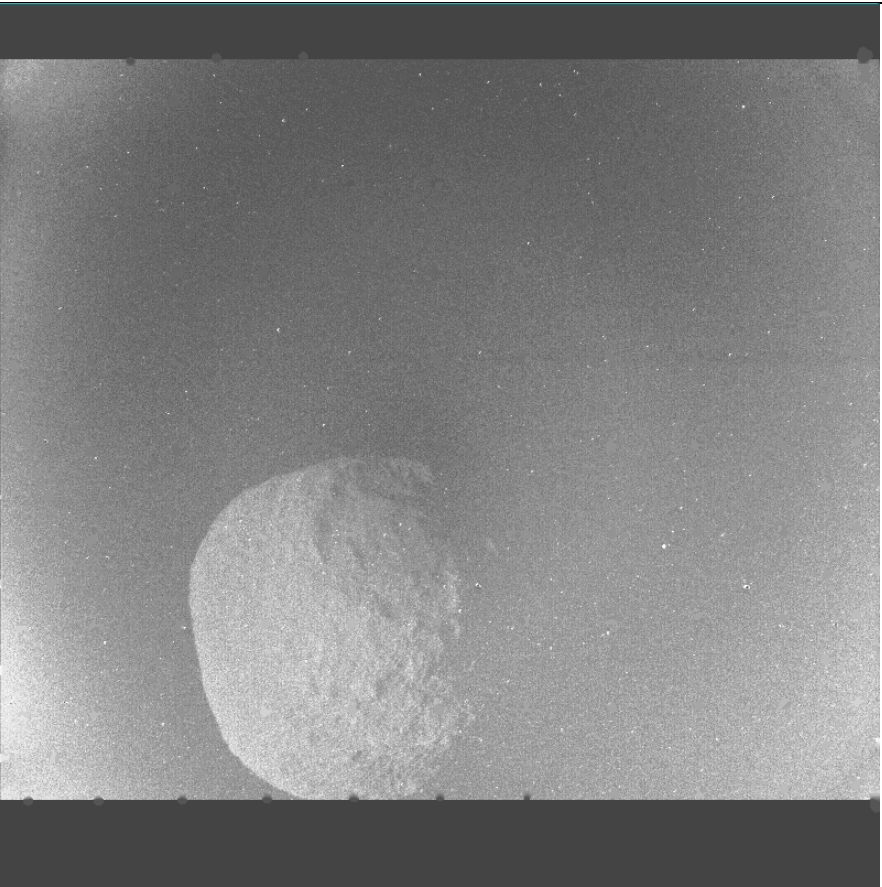
! Low-pass filter to reduce noise
BOXFLT2 nrstr nrflt 'LOW NLW=7 NSW=7

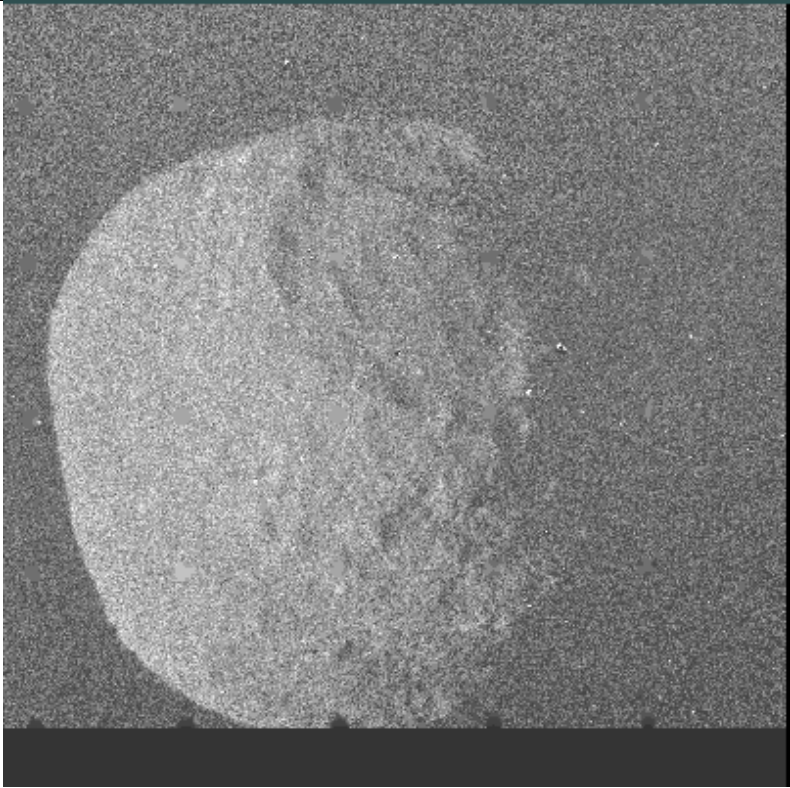
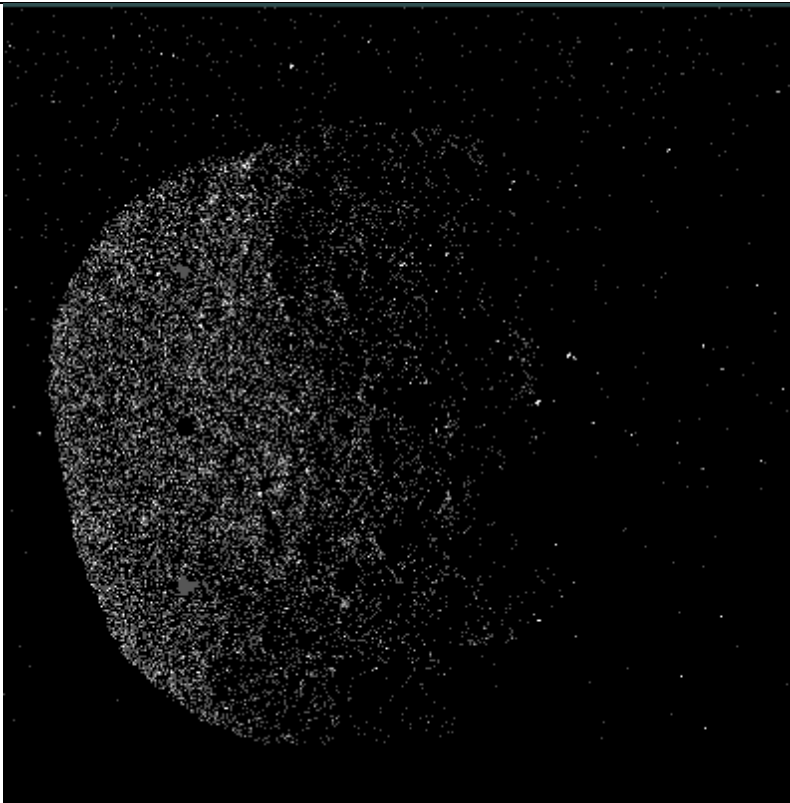
! Filter to suppress some artifacts
FILTER nrflt nrfinal NLW=3 NSW=3 WEIGHTS=(1,1,1,4)


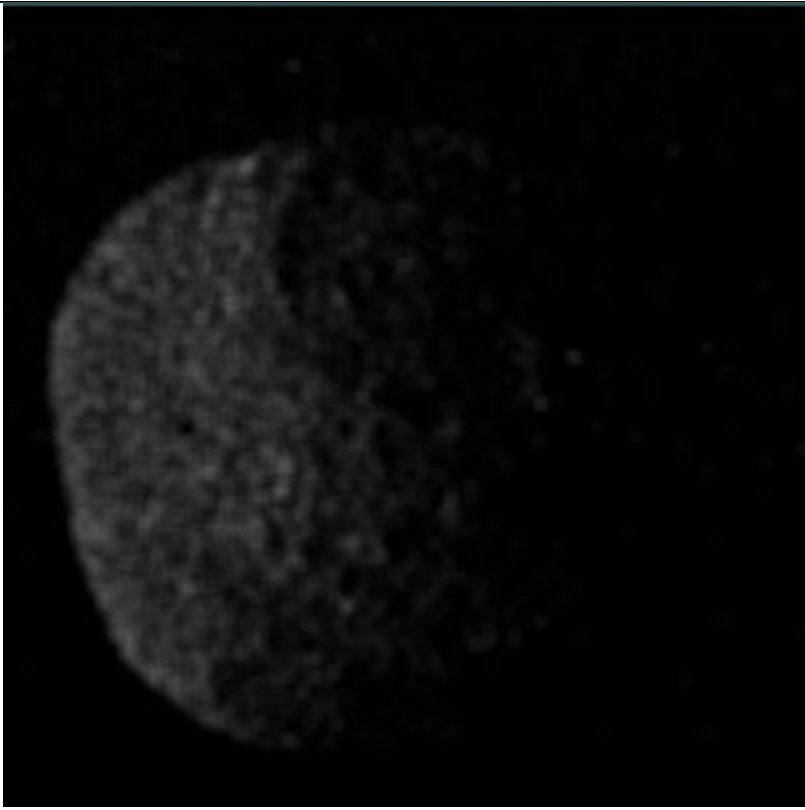
end-proc
```

Following are the initial, intermediate, and final products:

VICAR Quick-Start Guide

	<p>C1138920_RAW.IMG</p> <p>The original image viewed with a Gaussian stretch in xvd to make the moon visible.</p>
	<p>r</p> <p>After using RESSAR77 to remove the fiducial marks, viewed with a Gaussian stretch in xvd.</p>

	<p><code>nrsb</code></p> <p>After subtracting the dark image and cropping, viewed with a Gaussian stretch in xvd.</p>
	<p><code>Nrstr</code></p> <p>After stretching, viewed as-is in xvd.</p>

	<pre>nrflt</pre> <p>After low-pass filtering, viewed as-is in xvd.</p>
	<pre>nrfinal</pre> <p>After a final artifact filter, viewed as-is in xvd.</p>

5. List of Programs

This is a list of the general application programs contained in the P2 library for version 3.0 of the VICAR Open Source release.

General application programs operate on any VICAR image, subject to various restrictions. Most of these programs are restricted to 8-bit and/or 16-bit data while a few handle the full range of data types (32-bit integer, single and double precision floating point, complex). Most of the programs are restricted to monochrome (single band) images while a few operate on multispectral data.

Each program is listed only once under one of the functional areas below. Functions which deal primarily with monochrome images appear first, followed by functions for multispectral images and functions for graphical and tabular data.

5.1. *Categories*

5.1.1. Utilities

- VICAR help
- VICAR utilities
- VICAR procedure generation
- Manipulating ASCII files
- Data conversion

5.1.2. Displaying Images, Text, and Graphics

- Displaying images
- Pixel listings and plots
- Label processing & display
- Text and graphics overlays

5.1.3. Generic Tools

- Generating synthetic images
- Image statistics
- Mathematical and logical operations
- Contrast enhancement
- Color reconstruction
- Digital filters
- Fast Fourier Transforms
- Image restoration
- Image blemish removal
- Image noise reduction/simulation
- Image concatenation
- Image orientation
- Image magnification and reduction
- Geometric transformations

5.1.4. Image Registration and Mosaicking

- Image navigation
- Image registration
- Map projections
- Map projections of Irregularly Shaped Objects (ISOs)
- Mosaic generation (IBIS)
- Mosaic generation (multimission)

5.1.5. Calibrating the Camera and Target

- Geometric calibration
- Radiometric calibration
- Photometric function

5.1.6. Miscellaneous

- Atmospheric feature tracking
- Astronomy
- Super-resolution
- Focus analysis
- Elevation maps
- Stereo images

5.1.7. Multispectral Data

- Multispectral data utilities
- Principal component transformation
- Multispectral classification

5.1.8. Graphics and Tabular Data

- IBIS interface file operators
- IBIS graphics file operators
- IBIS file conversion routines
- Displaying IBIS graphics or tabular data

5.1.9. Project-Specific Programs

- Cassini
- Galileo
- Viking Orbiter
- Voyager

5.2. Program Listing

5.2.1. Utilities

VICAR help:

NUT	On-line VICAR tutorial
NUTINP	Called by NUT
NUTPROMPT	Called by NUT

VICAR utilities:

CHKSPACE	Return amount of available space on specified disk
COMMON_SUBPDF	Various sub-PDFs for use by menu-driven PDFs
COPY	Copy all or part of a labeled or unlabeled image
DATETIME	Print current date and time: dd-mmm-yy hh:mm:ss
FILE2TCL	Determine various file parameters
RUN_ISQL	Enter or delete data in Sybase catalog
TCLMATH	Evaluate math expressions on tcl real variables
TEMPNAME	Append ZZZ extension to filename to make it a temporary file

VICAR procedure generation:

CNT	Return number of files in a list created by SRCH
COMMENT	Display comments during execution of a procedure
FORM	Return image format and size as TAE variables
GETLAB	Copy a VICAR label item to a TAE variable
LAB2TCL	Copy VICAR label items to TAE variables
MAKESRCHLIST	Output a list of all files in a directory in SRCH format
MAXMIN	Compute min and max DN and output as TAE variables
NXT	Return data for next file in a SRCH list
RESET	Reset the next file pointer of a SRCH list
TRANSLOG	Translate a logical name
USERNAME	Return current userID
WILDCARD	Find all files matching a wildcarded string

Manipulating ASCII files:

ADDTOFILE	Append a string to an ASCII file
CREATEFILE	Create an empty file
COLUMNAR	Concatenate two ascii files left-to-right
HEADERGEN	Output multiple records of an ASCII file as a single record
TABULATE	Concatenate ASCII files into tab-delimited file
TYPETEXT	Output ASCII text file to terminal and session log

Data conversion:

CCOMP	Convert image from complex to real format or vice-versa
-------	---

VICAR Quick-Start Guide

CFORM	Convert image between data types with optional scaling
DDD2VIC	Convert Mars Global Surveyor "ddd" format data to VICAR
FITSIN	Convert FITS data to VICAR format (P3)
GTGEN	Create a GeoTIFF label from parameter input
GTLIST	List image mapping info from a GeoTIFF label
IMG2ASCII	Convert image data to ASCII text file
ISISLAB	Prints PDS label and history objects of an ISIS cube
PIC2VIC	Convert PIC format images to VICAR
PSCRIPT	Prepare a VICAR image for output to a Postscript printer
VIC2PIC	Convert VICAR images to PIC format
VTIFF	Convert images between VICAR and TIFF format
VTIFF30	Convert images between VICAR and GeoTIFF format

5.2.2. Displaying images, text, and graphics

Image displays:

EDIMAGE	Interactive image annotation and editing
HICCUP	Create histogram file for halfword image
HISTGEN	Create histogram file for byte or halfword image
MASKV	Create an image display for film recording
PRINTPIX	Print a grey level display of an image
QB	Sequential display of a list of files (Quick Browse)
XVD	Interactive image display

Pixel listings and plots:

LIST	Print the DN values of an image area
EZLIST	Similar to LIST, but output may be an ASCII text file
LISTBITS	Print the DN values of an image area in binary
QPLOT	TAE procedure which calls QPLOT2
QPLOT2	Line or spectral plots to VRDI, Tektronix, Regis, Printronix

Label processing and display:

CLEANLABEL	Remove duplicate label items from an image's history label
GTIGEOLO	Parse a MIL-STD-2500 image header IGEOLO field
GTLABFIX	Convert property label values from exponent to decimal format
LABEL	Print or edit the VICAR label
LABLIST	Print VGR or GLL SSI flight label
LABSWTCH	Switch the history labels of two VICAR images
LABVFY	Verify that an image label contains a specified string

Text and graphics overlays:

ADL	Draw line between two points in image
CLABEL	Copy label from a "CONTOUR" file to a "POLYSCRIB" file

CONLAB	Image contouring procedure (calls CONTOUR)
CONTOUR	Create a graphics file of contours or "isolines"
FONT	Superimpose text on images in various font styles and sizes
GRID	Superimpose a user defined reference grid on a byte image
MAPGRID	Overlay a uniform grid on an image
MSSVIEW	Draw scatterplot in center of MSS image
OVERLAY	Overlay a latitude-longitude grid on an image
ZCIRCLE	Zero out a circular or elliptical area of an image
See also: EDIMAGE	

5.2.3. Generic tools

Generating synthetic images:

ELLIPSE	Create synthetic images of oblate spheroids
FRACGEN	Simulate elevation data via fractional brownian motion
GEN	Create synthetic (ramp) image
GENTHIS	Create image from input DN list
RADAGEN	Synthesize a radar image from an elevation map
RANDPIXEL	Fill a blank image with random data points
SHP2RAST	Rasterize shape data to 1x1 degree cell files
SPOT	Synthesize images of spots of various sizes and profiles
TARGET	Create test targets for optical systems of known MTFs
WEDGE	Create a pie wedge image

Image statistics:

ASCHIST	Create a tab-delimited ASCII histogram file
ENTROPY	Compute image entropy
HIST	Print histogram of byte, integer, or floating point image
LAVE	Compute mean or sigma for each line or column of an image
MOORESC	Count overlap of Moore regions for clustering
PIXGRAD	Compute the magnitude and gradient of an image
PIXSTAT	Compute statistical data in a local area about a pixel
IMGSTAT	Output image representing local min, max, mean, or sigma

Mathematical and logical operations:

AVERAGE	Average up to 48 images into one image
DIFPIC	Compute difference between two images
F2	Perform mathematical and logical operations on images
RATIO	Compute ratio between two images
SC2RPC	Compute RPC from spacecraft ephemeris, camera model
SCINTERP	Interpolate two ephemerides/attitudes to sc2rpc vectors

Contrast enhancement:

VICAR Quick-Start Guide

ASTRTCHR	Convert floating point images by byte via histogram scaling
FIT	Convert halfword images to byte via histogram scaling
HSTRETCH	Modify specific DN values of an image
STRETCH	Image contrast enhancement
STRETVAR	Linear contrast enhancement as a function of line number
VLOOKUP	Modify DNs of B/W or multispectral images via table lookup

Color reconstruction:

COLORFIT	Replace missing image of color triplet via numerical fit
COLORME	Color balancing of uncalibrated RGB images
COLORRGB	Convert n multispectral images into RGB or XYZ tristimulus
COLORT	Transform color triplets between RGB and other color domains
COLORT2	Transform color, like COLORT but for half/full/real data
DNTOXY	Convert multispectral images to xyY color space
GIACONDA	Color transformation to reproduce specified spectra
RGB2ISH	Convert between RGB and ISH color spaces.
RGB2PSEUDO	Create pseudo-color rendering of an RGB color triplet
RGBTOXY	RGB to xyY color transformation
SPECTOXY	Create xyY color triplet from registered color n-tuplet
TRISTIM	Compute tristimulus values and chromaticity coordinates
TRUCOLOR	Color reconstruction of designated spectra
XYZ2HDTV	Convert xyY color triplet to RGB triplet for HDTV
XYTOSPEC	Convert an xyY color triplet to an RGB triplet
YFIT	Autostretch of the tristimulus Y element of a xyY triplet

Digital filters:

APODIZE	Reduce ringing on the edge of image during filtering
BOXFLT2	High-pass or low-pass filter
CONCOMP1	Removes high frequency noise components from an image
FILTER	General purpose digital filter
MEDIAN	Median filter
SBOXFLT	Highpass filter (TAE procedure which calls BOXFLT2)
SHADOW	Brighten shadows preserving details of the image
SHADY	Add contour lines and/or shading to an image
SHADY2	Simulate shadows from illumination at given azimuth-elevation
TFILT	High-pass filter with thresholding to prevent ringing of limb

Fast Fourier Transforms:

FFT11	1-D FFT
FFT1PIX	Convert a 1-D FFT to an amplitude and/or phase image
FFT2	2-D FFT procedure (calls FFT22)
FFT22	2-D FFT
FFTADD	Add 2 FFTs

VICAR Quick-Start Guide

FFTFIT	Modify 2-D FFT to force images to have identical power spectra
FFTFLIP	Translate 2-D FFT axes so DC term is in center of output
FFTMAGIC	Compute amplitude of an FFT from the phase or vice-versa
FFTPIC	Convert a 2-D FFT to an amplitude and/or phase image
IFFT	Interactive modification of FFT
POWER	Compute 1-D power spectrum of an image area
SWAP	Swap the quadrants of an image or complex FFT

Image Restoration:

CLEAN	Restore image by iteratively deconvolving a pt spread function
FIL2	Compute filter weights to deconvolve an image
FILTER2	Image restoration procedure (calls FIL2 and FILTER)
MEM	Non-linear deconvolution using Maximum Entropy Method
OTF1	Compute optical transfer function
PSF	Extract the point spread function from an image
RESTORW	TAE image restoration procedure (calls OTF1 and WIENER)
SPARSE	Simulate effect of a sparse aperture
WIENER	Restore an FFT image by using the Wiener noise additive model
WNR2005	Restore an FFT image by using the Wiener noise additive model

Image blemish removal:

BLEMPIC	Create image display of CCD camera blemishes
DESTRIPE	Remove striping in images caused by variations in detector response.
DS4	Remove 6-line striping from Landsat images
QSA	Add or subtracts constants to image areas
REPAIR	Locate and interpolates over bad lines
SARGON	Interpolate over polygonal regions of an image (interactive)
SARGONB	Interpolate over polygonal regions of an image (batch)
ZFILL	Interpolate over zero regions of an image

See also: EDIMAGE

Image noise reduction/simulation:

ADDNOISE	Add gaussian noise, shot noise, or bit errors to image
ADESPIKE	Remove single-pixel spikes from an image
DENOISERV	Perform impulse noise removal using total variation minimization.
DESPIKE	Remove single-pixel spikes from an image
GAMMA	Perform gamma correction.
GAUSNOIS	Create Gaussian noise image
JPEGFIX	Reduce blockiness introduced by severe JPEG compression
MINFILT	Radiation noise suppression
POLYNOIS	Generate a noise image of specified noise spectra
REMNOISE	Remove single-pixel spikes from an image
REMRAY	Remove cosmic ray and radiation noise from an image

TVREG Reduce noise by Total Variation minimization

Image concatenation:

APPEND Concatenate up to 30 images vertically
 MSS Concatenate up to 30 images horizontally
 CONCAT Concatenate images of the same size
 VICCUB Combines multiple images into one multi-band image

Image orientation:

FLOT Rotate or reflects image by 90 or 180 degrees
 ROTATE Rotate an image 90 degrees
 ROTATE2 Rotate an image by an arbitrary angle (calls GEOMA)

Image magnification and reduction:

BICUBIC Integral image enlargement via cubic convolutional filter
 FFTMAG Enlarge images by 2**N using Sampling Theorem
 INSERT Enlarge image in line direction
 SIZE Enlarge or reduce an image via bilinear interpolation

Geometric transformations (rubber sheeting):

GEOM Geometric transformation (calls LGEOM or MGEOM)
 GEOMA Geometry transform of an image, randomly spaced points
 GEOMV High-resolution geometric transformations on images
 LGEOM Geometric transformation of an image, uniform grid
 MGEOM Geometric transformation of an image, uniform grid
 POLYGEOM Geometric transformation of tiepoints
 TIECONV Prepare a gridded dataset for GEOM programs

5.2.4. Image registration and mosaicking

Image navigation:

EPHEMERIS Returns ephemeris for a planet as seen from another planet
 FARENC Correct camera pointing by fitting limb
 GETLL Convert line-sample to lat-lon and output to TAE variable
 GETPC Output planet center line-sample coordinates as TAE variable
 GSPICE Print SPICE data for an image
 HORIZON Detect strong and weak horizon borders for martian images
 MAKECK Create an empty SPICE C-kernel
 NAV Correct camera pointing by fitting limb, ring, or stars
 NAV2 Correct camera pointing by tiepoint registration
 OMC Coordinate transformation of C-matrices and position vectors
 PERSLAB Store navigation data for a flight image into VICAR label
 RINGORBS Generate the Ring Orbital Elements file (for NAV)

VICAR Quick-Start Guide

SPICE Print SPICE data for an image

Image registration:

AUTOMATCH	Find matching tiepoints in a sequence of images
CORNER	Locate candidate tiepoints by scanning an image for corners
LINEMTCH	1-d line matching of an image pair (correlation)
MANMATCH	Find matching tiepoints in a sequence of images (interactive)
PICMATCH	Find matching tiepoints in an image pair
PICREG	Find matching tiepoints in an image pair (interactive)
POLYREG	Perform affine transformation on a set of tiepoints
TIECONM	Compute geometric distortion from randomly spaced ties
TIEPARM	Compute geometric distortion parameters from tiepoints
TIEPLOT	Plot tiepoints stored in an IBIS file as vector displacements
TP	Find matching tiepoints in a sequence of images (interactive)

Map projections:

GEOMREC	Transform slant range radar data to ground range
MAP3	Standard cartographic projections
MAPCOORD	Convert from lat-lon to line-samp or vice-versa
MAPLABPROG	Store projection data into label
MAPTRAN	Convert images from one projection to another
POLARECT	Rectangular to polar projection and vice-versa
POLARECT2	Convert images to polar coordinates and back
POLYMAP	Convert tiepoints from one projection to another
POLYPMAP	Convert tiepoints from lat-lon to line-sample
PTP	Project an image from one perspective to another
SINPROJ	Sinusoidal projection
TRICOEF	Compute coefficients for conformal and authalic projections

Map projections of Irregularly Shaped Objects (ISOs):

AREAIISO	Compute AUXiliary lat-lons for Irregularly Shaped Objects
AUXILIARY	Compute conformal-to-planetocentric auxiliary ISO coords
EFGISO	Compute E, F, and G components of projected ISOs.
MAPAUX	Map projection of irregularly shaped objects (ISOs).
SNYDER	Compute centric coordinates for ISOs.

Mosaic generation (IBIS):

FEATHERV	Mosaic images using Moore distance feathering
GEOMZ	Brightness transformation (rubber-sheeting of DN axis)
MASKMOS	Create an image mask to aid in mosaicking
RAPIDMOS	Assemble registered images into a mosaic

Mosaic generation (multimission):

FASTMOS	Assemble registered images into a mosaic
GTAPPEND	Concatenate images in a top to bottom fashion
GTMISS	Concatenate images in a left to right fashion
IBISGCP	Specify ground control points
IBISNAV	Copy SPICE data to an IBIS file
IBISUPDATE	Store corrected camera pointing into a C-kernel
INSECT	Mosaic two images
MOSPLOT	Plot footprints, overlap files, or error vectors for mosaics
NEWMOS	Assemble registered images into a mosaic

5.2.5. Calibrating the camera and target

Geometric calibration:

FIXLOC	Edit tiepoints
GETLOC	Extract tiepoints for a subarea of a grid target
GRIDGEN	Synthesize image of a grid target
GRIDLOCB	Locate intersections on a grid-target image
INTERLOC	Locate intersections on a grid-target image (interactive)
LOCUS2	Perform a least squares fit between two tiepoint files
MARK	Scribe rectangles about specified pixel locations
RADDIST	Project uniform grid of tiepoints to simulate optical distortions
SKEW	Linear transformation of tiepoints
XLOCUS	Apply transform (computed by LOCUS2) on grid locations

Radiometric calibration:

BLEMGEN	Create blemish file for GLL SSI and Cassini ISS cameras
DC	Compute dark current frame from light transfer sequence
CCDNOISE	Measure noise and system gain (CCD camera)
CCDRECIP	Measure shutter offset (CCD camera)
CCDSLOPE	Measure light transfer slope and offset (CCD camera)
FCNPOLAR	Fit polarization data to determine polarization axis of a filter
GALGEN	Create radiometric and dark-current files for GLL & Cassini
LTGEN	Create a light-transfer or reciprocity file
MOMGEN	Compute moments for image areas of light-transfer sequence
MOMGEN2	TAE procedure to process light transfer or reciprocity data
MOMLIST	Print or output to a text file contents of Light Transfer File
PICSUM	Compute sum of multiple images and flags saturated pixels
SIGNAL	Output light transfer data for a pixel to a text file
SRCHEDGE	Get angle of image divided diagonally into light & dark areas

Photometric function:

PHODEM	Demonstrate use of menu-driven PDFs
PHOPDF	Contain sub-PDFs specific to each photometric function
PHOTTEST	Generate synthetic data for testing PHOTFIT2

PHOTFIT2	Fit photometric function to data in catalog
PHOTFUNC	Photometric function correction of flight images

5.2.6. Miscellaneous

Atmospheric feature tracking:

DVECTOR	Draw vectors representing tiepoint displacements
MORPH	Create intermediate images between two images
TPTEDT2	Identify and removes erroneous tiepoints

Astronomy:

STARCAT3	Locate and catalogs stars in an image
----------	---------------------------------------

Super-resolution:

SUPERRES	Combine many images to create super-resolution image
----------	--

Focus analysis:

BESTFOCUS	Convert focus stack to best-focus image and depth map
BESTSCALE	Rescale images to the same size for BESTFOCUS

Elevation maps:

LSTOXYZ	Converts tiepoints to xyz planet coordinates
TOPOMAP	Generate relative elevation maps from tiepoint data
TOTOPO	Converts tiepoints from xyz to line-samp of topomap

Stereo images:

CORRELATE1D	Compute 1-D correlated tiepoints between images
DISPARITY	Combines two disparity images into radial disparity
MPFTPT1	Compute line/sample disparity of each pixel of a stereo pair
STEREOCAM	Convert tiepoint locations to xyz coordinates for a stereo pair
XYZSUN	Convert stereo tiepoint data of the Sun to xyz coordinates

5.2.7. Multispectral data

Multispectral data utilities:

HIST2D	Create 2-D histogram file of multispectral data
INSERT3D	Insert a band into a 3-d multispectral file
TRAN	Convert multispectral data between BSQ, BIL, BIP, MSS fmts

Principal component transformation:

EIGEN	TAE procedure which calls EIGENVEC and XFORM
-------	--

EIGENVEC	Computes principle components transformation matrix
XFORM	TAE procedure which calls XFORMAP or XFORMEM

Multispectral classification:

CLASSIFIER	Classify multispectral image pixels based on classes generated by clusterer.
CLUSAN	Apply clustering algorithm to multispectral data
CLUSTERER	Perform K-Means clustering on data produced by sampler.
CLUSTEST	Compute statistical significance of cluster in a state file
FASTCLAS	Bayesian maximum likelihood multispectral classifier
IBISCLST2	Calculate clusters from (x,y) points based on radial distance.
IBISCLST3	Calculate clusters from (x,y) points based on line/samp distance.
SAMPLER	Produce sample of image pixels used for statistical analysis.
STATPLT	Plot a classification statistics file
STATS	Compute statistics of training areas
USTATS	Perform unsupervised clustering on multispectral data

5.2.8. Graphics and tabular data

IBIS interface file operators:

AGGRG	Form aggregates of columns in an IBIS interface file
AGGRG2	Form aggregates of columns in an IBIS interface file
EDIBIS	Interactive editing of IBIS interface and graphics files
IBIS	Create, copies, concatenates, prints, and deletes IBIS files
IBIS2TCL	Copy IBIS tabular data to TAE variables
IBISLSQ	Perform least-square fits of specified columns
IBISREGR	Perform linear regression on IBIS tabular data
IBISSTAT	Compute various statistics of IBIS tabular data
MF	Math and logical operations on columns (FORTRAN)
MF3	Math and logical operations on columns (C)
MFD	Math and logical operations on double-precision tabular data
MULTOVLY	Compute n-dimensional histogram of n input images
ROWOP	Delete or select rows, or make multiple copies of rows
SORT	Sort rows of tabular data on one or more key columns
TRANSCOL	Convert long columns of data to smaller columns
XYZPIC2	Convert IBIS table to image
ZIPCOL	Copy columns from one IBIS file to another
ZIPCOL2	Create IBIS table from existing IBIS file(s)

IBIS graphics file operators:

POLYGEN	Generate an IBIS graphics file from user parameter list
GRUTIL	2-d and 3-d IBIS graphics-1 utility (append, convert)
GF	Perform math and logical operations on an IBIS graphics-1 file

VICAR Quick-Start Guide

POLYCLIP	Clip graphics elements to fit within a window
PLTGRAF	Plot a graphics-1 file inside a labeled box

IBIS file conversion routines:

ACOPIN	Convert an ASCII file into an IBIS table file
ARC2GRAF	Convert 2-D ARC/INFO point files to IBIS Graphics-1 format
GRAF2ARC	Convert IBIS Graphics-1 files to ARC/INFO format
GRAFIMG	Convert image data to a gridded 3-D graphics-1 file
MARKIBIS	Convert tiepoints from Mark to IBIS format or vice-versa
MSSIBIS	Copy data from MSS format to interface files
OLDGEOMA2IBIS	Convert (obsolete) GEOMA parameters to IBIS format
PERSPEC	Convert 3D graphics-1 file to true 2D perspective file
PIXMAP	Convert map coordinates in an IBIS file using a GeoTIFF label
RASTOGRAF	Convert graphics from raster to IBIS Graphics 1 format
TOIBIS	Convert data from image format to IBIS format
VQUIC	Convert ASCII file into an IBIS file

Displaying IBIS graphics or tabular data:

PAINT	Paint each region of an image a different color
POLYPNT	Convert IBIS polygon file to image format
POLYSCRIB	Convert a Graphics-1 file to image format
PLOT3D	Plot a 3-d IBIS file
PLOTINT	Plot an IBIS interface file
XYZPIC	Convert a 3-D graphics-1 file into an image
ZINTERP	Interpolate over random elevation data to create an image

5.2.9. Project-specific Programs

Cassini Mission:

TABLESEARCH	TAE proc to extract point response data from a CASPRF file
-------------	--

Galileo Mission:

GALSOS	Radiometric correction of Galileo SSI images
GLLPSPF	Create an SSI point spread function file
NIMSCMM2	Create a NIMS cube from Phase 2 EDRs
RVISIS2	Simplified interface for VISIS2
VISIS2	Converts GLL NIMS cubes between VICAR and ISIS formats
VISISX	Converts VICAR 3-D image to ISIS Cube file and vice-versa

Magellan Mission:

SIZEMGN	Resize an image (see SIZE) with Magellan-specific features
---------	--

Viking Orbiter Mission:

VICAR Quick-Start Guide

BLEMVORB	VO camera blemish removal
DROPOUT	Fill in data gaps in VO images
RESLOCVO	Locate reseau on Viking Orbiter images
RESSAR75	Remove reseau from Viking Orbiter images
SOS	Radiometric correction of Viking Orbiter images

Voyager Mission:

VGRCDCOPY	Convert a VGR image archived on CDRom to a VICAR image
VGRFILLIN	Fill in data gaps in VGR (EDR) images
CAMPARAM	Copy camera params from VGR label to TAE local variables
RESLOC	Locate reseau on VGR images
RESSAR77	Remove reseau from VGR images
OSBLEMLOC	Convert VGR blemish locations from image to object space
FICOR77	Radiometric correction of VGR images
FIXVGR	Scale VGR images to correct for FICOR77 scaling error
PHOTLIST	Print phase, incidence, and emission angles for a VGR image

6. Acronym List

AFIDS- Automatic Fusion of Image Data Systems

ASTER- Advanced Spaceborne Thermal Emission and Refection Radiometer

AVIRIS- Airborne Visible/InfraRed Imaging Spectrometer

AVHRR- Advanced Very High Resolution Radiometer

COSMIC- Computer Software Management and Information Center

GeoTIFF- Georeferenced Tagged Image File Format

GOES- Geostationary Operational Environmental Satellite

GUI- Graphical User Interface

HRSC- High Resolution Stereo Camera

IBIS- Image- Based Information System

IPL-Image Processing Lab

ISIS- Integrated Software for Imagers and Spectrometers

ISS- Imaging Science Subsystem

ITAR- International Traffic in Arms Regulations

JNI- Java Native Interface

JPL- Jet Propulsion Laboratory

LROC- Lunar Reconnaissance Orbiter Camera

MDIS- Mercury Dual Imaging System

MEX- Mars EXpress

MIPL-Multimission Image Processing Lab

MODIS- MODerate resolution Imaging Spectroradiometer

NEAT-Near Earth Asteroid Tracking

NITF- National Imagery Transmission Format

OSIRIS- Optical, Spectroscopic, and Infrared Remote Imaging System

PDART- Planetary Data Archiving, Restoration, and Tools

PDF-Parameter Definition File

PDS- Planetary Data System

ROSES- Research Opportunities in Space and Earth Sciences

TAE- Transportable Applications Executive

TCL- TAE Command Line

USGS- US Geological Survey

VICAR- Video Image Communication And Retrieval

VIDS- VICAR Interactive Display Subsystem

VIMS- Visual and Infrared Mapping Spectrometer

VMC- Venus Monitoring Camera

VMS- Virtual Memory System

VRDI- Virtual Raster Display Interface

7. References

The following documents can be found in two places. First, they are included in the VICAR source distribution itself, in the directory:

`vos/docsource/vicar/`

Second, they are available on the VICAR Open Source page:

http://www-mipl.jpl.nasa.gov/vicar_open.html

[1] VICAR File Format

[2] Building VICAR

[3] RTL Reference Manual

[4] Building and Delivering VICAR Applications

[5] VICAR User's Guide

[6] VICAR Porting Guide