Building VICAR

Open Source version

Version 5.0

2023-06-26

Update by:

_____

Shari C. Mayer

_____

Robert Deen, VICAR Cognizant Engineer

Jet Propulsion Laboratory

California Institute of Technology

Pasadena, California

Table of Contents

1. Introduction

This document describes how to build and/or install the Open Source version of VICAR. It assumes some familiarity with Unix command lines and build processes.

The VICAR build process has been developed over the years to meet the needs of MIPL internal users. We are providing most of VICAR in open source form as a service to the community. However, we do not have the resources to invest in making it "pretty".

Furthermore, we have tested this only on a limited number of platforms, and only a handful of machines. Therefore, the Open Source build process has some rough edges, and may not work properly out of the box.

Hopefully most of the errors you encounter are easily resolved. But we would appreciate knowing what they are, so we can update the procedures or the documentation for the next release.

If you have problems you can't resolve, ask us. We may not have time or resources to answer every question or solve every problem, but we will do our best to try.

The VICAR Quick-Start Guide provides an introduction to VICAR, including how to use it and some aliveness tests you should run after the build. This document assumes some familiarity with that one.

1.1. Supported Platforms

VICAR is officially supported on the following platforms:

- 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:

- Mac OS X (64-bits) – we have built on MacOS Mojave v10.14.6
- Windows 10 (run, but not built, via a Centos 7 Docker image)

Note that VICAR will run on arm-based Macs (the new M1, etc) but they use the x86-64 code base at this time.

VOS also should run on Windows Server, Amazon Web Services, Microsoft Azure, and IBM Cloud, via Docker, though we haven't tried ourselves. Let us know if any of these work for you.

We simply don't have the resources to fully test those 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 four platforms can be considered "supported".

Note that we build on Red Hat Enterprise 8.x for our Linux distribution. The build "should" work with other distributions but you may find quirks that need fixing.

VOS 3.0 added a new platform option: Docker. The 64-bit Linux build configuration has been built and run in Docker (Centos 7 image) on a 64-bit Mac (running Mac OS X 10.11.6 El Capitan). It also has been run, though not built, on 64-bit Windows 10.

Note that building on Mac OS v10.14.6 and later likely will require rebuilding the external libraries with the latest gcc/gfortran. We built on MacOS Mojave 10.14.6 using gcc 8.2, gfortran 5.

2. System Prerequisites

A few packages need to be installed on your system in order for VICAR to work (whether you're using precompiled binaries or building it yourself).

2.1. Prerequisites for Linux-64

2.1.1. GNU compilers, including Fortran

The gcc, g++, and gfortran compilers are needed for VICAR. The gfortran compiler is needed even for the pre-built binaries in order to get access to the Fortran library (.so). Thus, we recommend installing them even if you're using pre-built binaries.

Release 5 was built using version 8.5.0 of all three compilers.

See your system documentation for installing these.

2.1.2. X-windows and Motif

These come standard with most Linux installations, but are required for xvd and some other packages. Motif, or more likely Lesstif, are only needed to build xvd, but an X-windows server is needed for runtime.

2.1.3. imake

If you are not building VICAR programs you can skip imake.

VICAR builds are based on "imake". This program was included with the X-windows system until X11R7. If you do not have imake you will need to get it. One location is:

https://www.archlinux.org/packages/extra/i686/imake/

Note that Mac users don't need to do this; a special Mac version of imake is included with the VICAR delivery. Given the difficulty in getting imake, it is likely that we will include linux versions too in the future, but for now you must get it independently.

A Debian Linux user reported the need to include the following Debian packages:

- xorg-dev and xserver-xorg-dev for the X11 development environment
- xutils-dev for imake
- xorg-libxp-dev and xorg-motif-dev for libXp and libXm (Motif)
- libncurses5-dev for /usr/include/curses.h

  A user reported a problem when using gcc 4.8 (or higher). Starting with this version, the compiler inserts extra comments, which messes up the processing of vicset1.csh during the build. If you are using this compiler, edit the file $V2TOP/vos/util/process_project_file.csh and add the "-nostdinc" option to the call to "cpp". We expect to add this in future versions but haven't had the time to test it on all platforms yet.

2.2. Prerequisites for Mac OS X

The Mac development environment seems to change a lot between releases of Mac OS X. While we've tried it on both fairly old (10.7.3) and new (10.14.6) versions, your mileage may vary. We have not tested all Mac OS X releases, but we have not had problems with the ones we have tested.

We have also had success running Intel x86-64 code on the newer arm-based Macs.

2.2.1. Mac System Integrity Protection

The Mac SIP system helps secure your mac, but it really gets in the way of running programs like VICAR. The $DYLD_LIBRARY_PATH environment variable, which we use to indicate where to find shared libraries, gets *reset* in every subshell! To avoid this, in vicset1 we set an alternate env var, $SIP_DYLD_LIBRARY_PATH, which contains what the value *should* be. We then recommend you put this at the top of your ~/.cshrc file so it gets executed in every subshell:

```
# Work around SIP problems with VICAR
if ($#SIP_DYLD_LIBRARY_PATH != 0) setenv DYLD_LIBRARY_PATH $SIP_DYLD_LIBRARY_PATH
```

2.2.2. Mac Gatekeeper

In a related vein, MacOS has a system called Gatekeeper which prevents you from running unsigned apps because they're not "trusted". We are looking at signing the VICAR apps in the future to avoid these problems but in the meantime you may have to work around this.

One option is to keep overriding this for each app as it comes up with a message. Okay if it's just a couple, but a problem otherwise.

On Ventura (MacOS 13) you can disable it from all applications run from Terminal:

https://stackoverflow.com/questions/76205425/xyz-app-cannot-be-opened-because-the-developer-cannot-be-verified-on-mac-vent

On other OS's you may need to turn off the Gatekeeper entirely:

https://wiki.jpl.nasa.gov/pages/viewpage.action?spaceKey=opencae&title=Cameo+Systems+Modeler+-+OSX+Will+Not+Start

at least temporarily while using VICAR. This may or may not be needed depending on your security settings.

2.2.3. Gnu Compilers

The gcc, g++, and gfortran compilers are needed for VICAR. The gfortran compiler is needed even for the pre-built binaries in order to get access to the Fortran library (.so). Thus, we recommend installing them even if you're using pre-built binaries.

The gcc and g++ compilers are pretty standard, you should be able to get them with the Mac Developer Tools or with Xcode. Some users have reported using compilers for MacPorts or Fink (which includes gfortran). Open up a terminal window and type "gcc --version" and "g++ --version" to see if you have these already installed (and installed properly, if you had to do it yourself).

If you're using the Apple-supplied compilers, you'll need gfortran. One convenient place to obtain a precompiled version is:

```
https://github.com/fxcoudert/gfortran-for-macOS/releases
```

In a recent clean installation, we did this:

From the Mac App Store, download and install Xcode

Install the command-line tools:

```
xcode-select –install
```

Then we installed `gfortran-ARM-12.2-Ventura.dmg` from the above link.

Important note: For ARM-based machines, we are still using x86-64 code. That means we need Fortran shared libraries for x86-64. So then you want to *also* install the Intel version from that site (as of this writing, `gfortran-Intel-12.1-Monterey.dmg`). You do not need the entire set of compiler tools for the Intel processor, just install that dmg.

2.2.4. Java

Java is required for some tools such as the transcoder and Labelocity. If you don't already have a version installed, get one e.g. from here:

https://www.java.com/en/download/

Any version 8 or beyond should work. Note, you may need to edit $V2TOP/vicset1.csh to adjust where $V2JDK points on your system.

2.2.5. imake

A version of imake for Mac OS X is included in the delivery, so you do not need to do anything about imake.

2.2.6. xquartz

Download and install X-windows, if you don't already have it. We recommend xquartz:

https://www.xquartz.org

or (perhaps an older link):

http://xquartz.macosforge.org

which is derived from the version that Apple used to ship with their OS until very recently. Note that we've had some reports of difficulty on recent OS X versions using X-windows remotely from Mac to Mac (i.e. from the Mac, ssh'ing to another Mac machine), but it works using "ssh –X" when logging in to a Linux machine. Local use on the Mac (as described here) should not be a problem.

2.2.7. OpenMotif

OpenMotif is only needed if you are rebuilding GUI code such as xvd.

After installing X-windows, install the OpenMotif library.

Get OpenMotif from here and install:

```
https://sourceforge.net/projects/openmotif-mac/
```

Then edit the following three files:

- $V2TOP/vicset1.csh
- $V2TOP/util/imake.config
- $V2TOP/MotifApp/Makefile.x86-macosx

In all three, search for OpenMotif and change the pathname as needed (it appears to occur, coincidentally, in two places in each file). The new pathname should be:

/Applications/OpenMotif21

If you upgrade to Mac OS X 10.10.5 Yosemite after installing X-windows, the symbolic link from /usr/X11R6 to /opt/X11 may be removed. You will need to recreate it with:

```
% ln -s /opt/X11 /usr/X11R6
```

because the OpenMotif binaries reference /usr/X11R6.

2.2.8. tbb library

VISOR uses an external called Embree, which needs the tbb library (for runtime, not just compile time) which is not included by default on Macs. In principle, you could install it using the directions here:

https://macappstore.org/tbb/

However, the currently available versions are too new for the VOS 5 build. We hope to fix this in the future by upgrading the library and rebuilding. However, for now, we have put an appropriate Intel Mac version of the library in the Embree external, so it will automatically get picked up. So no action is required, but we needed to document this for the future or if you want to upgrade Embree yourself.

3. Obtaining/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.

3.1. Obtaining VICAR

VICAR can be obtained from the following link:

https://github.com/NASA-AMMOS/VICAR/releases

There you will find links to:

- A git repository of the VICAR source, without its external libraries
- A tar file of the VICAR source, without its external libraries, auto-generated by git
- Tar files of pre-built external libraries, one for each of the supported platforms
- Tar files of pre-built VICAR and external libraries, one for each of the supported platforms
- A 64-bit Centos 7 Docker image pre-loaded with libraries needed for VOS, though it does not include VOS itself – you still need to grab and install the 64-bit Linux VICAR tar(s)
- A 64-bit Centos 7 Docker image pre-loaded with libraries needed for VOS, and with VICAR and the externals libraries all pre-built and installed. Launch this image and you can immediately start running VICAR (and modifying/building if you like).
- Calibration files and sample data for VISOR (see separate VISOR user's guide).

The pre-built external libraries include their source. You should not have to rebuild them. If so, you are on your own. For the externals, generally instructions should be in a README file of some sort in each library.

The VOS source tree (excluding external libraries) is:

- vicar_open_5.0.tar.gz

The pre-built VICAR+externals tar files are:

- vicar_open_bin_x86-64-linx_5.0.tar.gz
- vicar_open_bin_mac64-osx_5.0.tar.gz

If you need to build VICAR from source (obtained from the git repository), you can avoid building the externals by using one of these pre-built externals tar files:

- vicar_open_ext_x86-64-linx_5.0.tar.gz
- vicar_open_ext_mac64-osx_5.0.tar.gz

Note the intentional "linx" in the 64-bit Linux version.

The Centos 7 Docker image is:

- centos7_for_vos5.0.img.gz

3.2. Directory Layout

The location for VICAR is arbitrary; you can put it wherever you want. However, the pathname should be all lowercase (TAE does not like uppercase in the path for some operations). If needed you can create a softlink alias for VICAR. For example, if you wanted VICAR to be in /Users/myaccount/vicar (note the capital U), you could do this:

```
% sudo ln -s /Users/myaccount/vicar /usr/local/vicar
```

There's nothing magic about /usr/local/vicar, it's just the pathname we use at MIPL. As long as it's all lowercase, it's fine. If you absolutely must have uppercase characters in the path, running programs from the shell will still work; it's only TAE that really has issues with the upper case.

Underneath this directory you will probably want a version number directory, so you can have multiple versions of VICAR. This is easily accomplished by cd'ing to that version-numbered directory and simply untarring the tarball there.

3.3. Installing a Pre-Built VICAR for Linux or Mac

In this section, "x86-64-linx" is for the 64-bit Linux build. Replace with "mac64-osx" for the Mac build, the commands are otherwise the same. Any other differences are noted.

```
% cd /usr/local/vicar/
% tar xzf vicar_open_bin_x86-64-linx_5.0.tar.gz
```

You will then end up with these directories within /usr/local/vicar/:

```
vicar_open_bin_x86-64-linx_5.0/
vicar_open_bin_x86-64-linx_5.0/vicar_open_5.0
vicar_open_bin_x86-64-linx_5.0/vicar_open_ext_x86-64-linx_5.0
```

Rename the "vicar" directory:

```
% cd vicar_open_bin_x86-64-linx_5.0/
% mv vicar_open_5.0 vos
```

Rename the "external" directory:

```
% cd vicar_open_bin_x86-64-linx_5.0/

% mv vicar_open_ext_x86-64-linx_5.0 external
```

where the "vicar_open_5.0" directory contains "vicset1.source", "p2", etc. while "vicar_open_ext_x86-64-linx_5.0" contains "JAI", "tae", etc. Each platform-specific tarball, or cloning from git, will result in different directory names.

The V2TOP environment variable should be set to the "vos" tree: the one that contains VICAR itself, e.g.

```
% setenv V2TOP /usr/local/vicar/vicar_open_bin_x86-64-linx_5.0/vos
```

Everything is keyed off of this V2TOP environment variable; it is what allows VICAR to be moved at will (and for multiple VICAR installations to coexist on the same machine).

If you are not attempting to build VICAR, you are done! Jump into the "Starting up VICAR" section of the companion document *VICAR_guide_5.0*.

3.4. Docker

Docker is a system that includes all the prerequisites in a container that requires no installation on your local machine. It can be more convenient to use.

Note: Building and/or running VOS in Docker requires a VOS Docker image, which includes necessary libraries, such as gfortran and X11. However, the Docker host (Linux, MacOS, or Windows) needs a running X11 server in order for X11-based applications to run. So on MacOS, you will still need XQuartz. On Windows, you can choose from a number of available X11 servers. See Build Preparation below.

3.4.1. Installing in Docker on a MacOS Host

Two Docker images are available, one with just Centos7 and some required yum packages, and one that also has pre-built VICAR and its external libraries already installed.

- At the Mac OS X command line, enter (do this once to load the image into your Docker environment):

  ```
  % docker load -i centos7_for_vos5.0.img
  ```

- In XQuartz, check the box "Allow connections from network clients" in XQuartz -> Preferences -> Security (you should need to do this only once)
- At the Mac OS X command line, enter (every time you start XQuartz):

  ```
  % xhost + localhost
  ```

- At the Mac OS X command line, start the Docker container by entering (on one line):

  ```
  % docker run -ti -e DISPLAY=docker.for.mac.localhost:0 -v
  /host/dir/path:/docker/container/dir/path centos7_for_vos tcsh
  ```

The -ti options keep the command line working. The -e option defines the DISPLAY environment variable for the benefit of X11 clients. Note that “docker.for.mac.localhost” is predefined by Docker. The -v option mounts a host directory into the Docker container. You can install VICAR in this directory. Replace both paths as appropriate. For example, if your username is myaccount, you might use /Users/myaccount:/home. The centos7_for_vos identifies the image to load. The tcsh tells Docker to launch the tcsh shell for you after starting the container. This is the shell you would want to use to run VICAR. Here is an example startup:

```
bash-3.2$ uname -a
Darwin 15.6.0 Darwin Kernel Version 15.6.0: Thu Jun 21 20:07:40 PDT 2018; root:xnu-3248.73.11~1/RELEASE_X86_64 x86_64
bash-3.2$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos7_for_vos     latest              9fc6dcdf4a39        3 days ago          945MB
bash-3.2$ xhost + localhost
localhost being added to access control list
bash-3.2$ docker run -ti -e DISPLAY=docker.for.mac.localhost:0 -v /Users/wbunch:/home centos7_for_vos tcsh
[vos /vos]$ ls
Dockerfile
[vos /vos]$ uname -a
Linux 97df6539178b 4.9.87-linuxkit-aufs #1 SMP Wed Mar 14 15:12:16 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
 [vos /vos]$ cat /etc/centos-release
CentOS Linux release 7.5.1804 (Core)
```

Continue as if installing on a 64-bit Linux platform. For example, assuming that VICAR and the externals were untarred in the Mac OS X host user's home directory (/Users/myaccount) as

```
/Users/myaccount/vicar/vicar_open_bin_x86-64-
linx_5.0/vicar_open_5.0
/Users/myaccount/vicar/vicar_open_bin_x86-64-
linx_5.0/vicar_open_ext_x86-64-linx_5.0
```

then in the Docker Centos 7 container, they will appear as

```
/home/vicar/vicar_open_bin_x86-64-linx_5.0/vicar_open_5.0
/home/vicar/vicar_open_bin_x86-64-linx_5.0/vicar_open_ext_x86-64-
linx_5.0
```

Rename the vicar directory:

```
[vos v5.0]$ cd /home/vicar/vicar_open_bin_x86-64-linx_5.0
[vos v5.0]$ mv vicar_open_5.0 vos
```

Rename the external directory:

```
[vos v5.0]$ cd /home/vicar/vicar_open_bin_x86-64-linx_5.0
[vos v5.0]$ mv vicar_open_ext_x86-64-linx_5.0 external
```

If you use the Docker image that has VICAR preloaded, then the VICAR and externals tars will already be unpacked in place, and the "external" link above will already be created.

Define V2TOP, source the vicsets, and test start the VICAR TAE shell:

```
[vos v5.0]$ setenv V2TOP /home/vicar/vicar_open_bin_x86-64-
linx_5.0/vos
[vos v5.0]$ source $V2TOP/vicset1.csh
[vos v5.0]$ source $V2TOP/vicset2.csh
[vos v5.0]$ vicar
```

3.4.2. Installing in Docker on a Windows Host

This section assumes that you have read the previous section for installing on a Mac OS X host.

The Docker image is loaded from a Windows command shell (or Powershell, but not Powershell ISE) in the same way that it is in Mac OS X:

```
> docker load centos7_for_vos.img
```

You may have to xhost + localhost to allow Docker container X11 clients to connect to your Windows X11 server, depending on that server. Using Cygwin's X11 server, it was not necessary.

Right-click on the Docker whale icon in the quick launch area and select

```
Settings->Shared Drives->C
```

to allow the mounting of a directory in C into the Docker container. Start docker with a command like the following in a Windows command shell (on one line):

```
> docker run -ti -e DISPLAY=10.0.75.1:0 -v
C:\Users\myaccount:/home --user vos centos7_for_vos tcsh
```

Note that the DISPLAY address is the default host IP address on the network interface Docker creates for Windows. You can use the Windows command shell command "ipconfig" to see this interface. From this point, installing and running VICAR is the same as for Mac OS X above. Note that you must untar the VICAR and external tar.gz archives using the Centos 7 environment. It seems that Cygwin's representation of Linux soft links in Window's filesystem is not compatible with that used by Centos 7.

3.4.3. Build Your Own Docker Image

You can download a pre-built Docker image containing the libraries needed by VICAR. You also can build your own Docker image, if you like. The centos7_for_vos.img was built with this command:

```
% docker build -t centos7_for_vos .
```

where a text file named "Dockerfile" was located in the current directory. That file contained these Docker directives:

```
# Use an official centos O/S as a parent image
FROM centos:latest


# Install general needed packages
RUN yum -y install gcc-c++.x86_64 \
  gcc-gfortran \
  imake.x86_64 \
  java-1.7.0-openjdk-devel.x86_64 \
  libXp.x86_64 \
  libXpm-devel.x86_64 \
```

```
  make.x86_64 \
  motif-devel.x86_64 \
  ncurses-devel.x86_64 \
  tcsh.x86_64 \
  xterm-295-3.el7.x86_64

RUN ln -s /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.181-
2.6.14.8.el7_5.x86_64 /usr/java

# Vicar GDAL plugin-specific needed packages
RUN yum -y install boost-devel.x86_64 \
  libcurl-devel.x86_64 \
  libtiff-devel.x86_64 \
  libxml2-devel.x86_64 \
  patch.x86_64 \
  pcre-devel.x86_64 \
  postgresql-devel.x86_64 \
  sqlite-devel.x86_64 \
  unixODBC.x86_64
# Set up VOS User
RUN mkdir /vos
RUN useradd -ms /usr/bin/tcsh vos
WORKDIR /vos
RUN chown vos:vos /vos

# Copy the current directory contents into the container at /vos
ADD --chown=vos:vos . /vos

# Switch to user vos
USER vos
```

4. External Libraries

Before describing the process for building VICAR source, the externals deserve mention.

VICAR makes use of several third-party software packages, which we collectively call "external" libraries. These external libraries are provided as a convenience, as some of them are unusual or hard to find. However, the original developers maintain all copyrights to the externals, and if you use them you are bound by the license terms of each individual package.

The external libraries are listed here. Note that the version numbers are current as of this writing, but subsequent VICAR releases may have different version numbers. URL's were current when we obtained the packages, but that was years or in some cases decades ago and they may or may not still be valid. In general the version VICAR uses is far behind "current" for the still-maintained packages, and may not still be available. If you wish to upgrade you can try, but you are on your own to resolve problems.

A very few packages have been modified slightly for MIPL use. A README file at the top of each such external tree defines what changes (if any) have been made. Generally the changes are limited to resolving build issues.

If a given external library does not work for you, it's likely that only a handful of application programs actually use it. Depending on what you want to do with VICAR, you may be able to get along without it.

Note: there is one additional package not listed here, because it is not included in the external directory. That package is "gnuplot". Unlike the other packages, it is not needed at compile/link time, which is why it's not included in the external directory. A few programs, such as ccdnoise, ccdrecip, ccdslope, mosplot, otf1, plot3d, plotint, pltgraf, power, qplot, qplot2, and tieplot, output plot files that must be sent through gnuplot to be seen. We may include gnuplot in external in future releases.

Bottom line, you should use the supplied external directory. In which case you can skip the rest of this section, it's just for reference.

4.1. boost-c++

Version: 1.68.0

Source: https://www.boost.org/

4.2. ceres-solver

Ceres Solver is an open source C++ library for modeling and solving large, complicated optimization problems. It can be used to solve Non-linear Least Squares problems with bounds constraints and general unconstrained optimization problems. It is a mature, feature rich, and performant library that has been used in production at Google since 2010.

Version: 1.12.0

4.3. coin

The location for the COIN3D, a clone of
Open Inventor Library: The root source URL is https://coin3d.github.io/.

Version: v4.0.0

4.4. commons-vfs

The Commons VFS provides a single API for accessing various different file systems.

Version: 2.0

Source: http://commons.apache.org/proper/commons-vfs/download_vfs.cgi

4.5. eigen

Source: http://eigen.tuxfamily.org/
Eigen is a C++ template library for linear algebra: matrices, vectors,
numerical solvers, and related algorithms. Needed by Ceres.

Version: v3.2.5

4.6. embree

Source: http://embree.github.io/downloads.html

Version: v3.7.0

4.7. fftw

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST).

Version: 3.3.4

Source: http://www.fftw.org/

4.8. gdal

GDAL is a translator library for raster and vector geospatial data formats.

Version: 2.2.3

Source: https://gdal.org/

4.9. glog

Version: v0.3.4

4.10. gnuplot

Gnuplot is a portable command-line driven graphing utility.

Version: 4.6.1

Source: http://www.gnuplot.info/

4.11. gsl

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers.

Version: 2.2.1

Source: https://www.gnu.org/software/gsl/

4.12. JAI – Java Advanced Imaging

The Java Advanced Imaging API (JAI) provides a set of object-oriented interfaces that supports a simple, high-level programming model which allows images to be manipulated easily in Java applications and applets. JAI goes beyond the functionality of traditional imaging APIs to provide a high-performance, platform-independent, extensible image processing framework.

Version: 1.1.3

Source: http://java.sun.com/products/java-media/jai/downloads/download-1_1_2.html

4.13. jai-ext

JAI-EXT is an open-source project which aims to replace in the long term the JAI project. JAI provides a set of high level objects for image processing. JAI-EXT improves this API in three different ways:

- adding more features to the existing operations, like the support for nodata;
- improving the performances of the existing operations;
- developing new operations.

Version: 1.0.3

Source: https://github.com/geosolutions-it/jai-ext

4.14. JAI_ImageIO – JAI Image I/O package

The Java Image I/O API provides a pluggable architecture for working with images stored in files and accessed across the network. The JAI Image I/O Tools classes provide additional plugins for other stream types and for advanced formats such as JPEG-LS, JPEG2000, and TIFF.

Version: 1.1

Source: http://java.sun.com/products/java-media/jai/downloads/download-1_1_2.html

4.15. jakarta-commons-logging

Jakarta Common Logging is an abstract interface for different logging toolkits such as JDK1.4 util.logging and log4j.

Version: 1.0.4

Source: http://jakarta.apache.org/commons/logging

4.16. jakarta-oro

Regular Expression parsing tools for java. Includes a set of perl utilities so that perl style parsing can be done in java.

Version: 2.0.4

Source: http://jakarta.apache.org/builds/jakarta-oro/release/v2.0.4/jakarta-oro-2.0.4.zip

4.17. jama

JAMA is a basic linear algebra package for Java.

Version: 1.0.3

Source: https://math.nist.gov/javanumerics/jama/

4.18. javamail

The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications.

Version: 1.3.1

Source: https://www.oracle.com/java/technologies/javamail.html

4.19. jogl

JOGL provides Java bindings to the native 3D graphics library, OpenGL. It provides full access to the APIs in the OpenGL 2.0 specification as well as nearly all vendor extensions, and integrated with the AWT and Swing widget sets.

Version: 1.1.1a

Source: https://jogamp.org/jogl/www/

4.20. jpeg

Libjpeg is a widely used C library for reading and writing JPEG image files.

Source: http://libjpeg.sourceforge.net/

Version: 9.2.0

4.21. log4j

Apache Log4j is a Java-based logging utility.

Source: https://logging.apache.org/log4j/2.x/

Version: 1.2.14

4.22. math77

The Math77 library was developed by the JPL Computational Mathematics Subgroup. For more information, see <http://math.jpl.nasa.gov> or mail to vsnyder@math.jpl.nasa.gov. This library has been included under the VICAR tree in the past; it was moved to the external category in Feb. 96.

Version: 5.0

Source: http://math.jpl.nasa.gov

4.23. mpich

The following defines the location of the MPI (Message Passing Interface) library. MPI is a library for Beowulf-style parallel processing. The implementation being used is MPICH from Argonne National Laboratory. See <http://www-unix.mcs.anl.gov/mpi/mpich/>.

Version: v3.3.2

4.24. nom_tam_fits

A FITS image access library created by Thomas McGlynn of Goddard. Copyright: Thomas McGlynn 1997-1999. This code may be used for any purpose, non-commercial or commercial so long as this copyright notice is retained in the source code or included in or referred to in any derived software.

Version: 1.13.1

Source: https://heasarc.gsfc.nasa.gov/docs/heasarc/fits/java/v1.0/v1.13.1/

4.25. pds

This is the PDS 3 library, which is unfortunately a different library (it's newer) than the pds_label_lib below. It is needed by xvd. This includes the lablib3 and OAL (Object Access Library) parts of PDS.

Version: 4.8

Source: http://pds.nasa.gov/tools/pds-tools-package.shtml

4.26. pds4_tools

Software for accessing PDS4 data objects including selected transformations to common formats.

Version: 0.5.0

Source: https://pds.nasa.gov/tools/about/pds4-tools/

4.27. pds_label_lib

The PDS label library is used to parse, read, and write PDS 3 (Planetary Data System) format labels.

Version: 4.0

Source: https://pds.nasa.gov/tools/pds-tools-package.shtml

4.28. pds_mi_label

Metadata Injector for PDS Labels (MILabel), formerly known as Generate Tool.

Version: 1.1.2

Source: https://github.com/NASA-PDS/mi-label

4.29. product-tools

Library supporting the design/generation, validation and submission of PDS3 archival products.

Version: 3.2.0

Source: https://github.com/NASA-PDS/pds3-product-tools

4.30. sift

Version: v1.0

4.31. SPICE

The SPICE toolkit computes spacecraft and solar system geometry.

Version: 6.5

Source: http://naif.jpl.nasa.gov

4.32. TAE

TAE is the Transportable Applications Executive, which is used as a command-line interface for VICAR. Although optional at this point (programs can be run from the shell), it is required for building. TAE was developed by Goddard Space Flight Center, but is no longer available; the version in the VICAR external library package must be used.

Version: 5.3

Source: n/a

4.33. tiff

The TIFF library is a set of routines used to read/write TIFF files. Also included is the GEOTIFF library.

Versions: 4.0.6 (libtiff), 1.4.1 (libgeotiff)

Source: http://www.remotesensing.org/libtiff/

4.34. tinygltf

TINYGLTF library is a header only C++11 version of glTF, a royalty-free specification for the efficient transmission and loading of 3D scenes and models by applications.

Version: v2.5.0

4.35. tinyobjloader

Source: https://github.com/syoyo/tinyobjloader/blob/master/tiny_obj_loader.h
This TPS is a header only, single file, C++ code that allows to load in containers mesh file in OBJ format.

Version: v1.2.3

4.36. xalan

An XSL and XPATH processor for Java that transforms XML into HTML, text, or other XML documents under direction of an XSL stylesheet.

Version: 2.7.2

Source: http://xml.apache.org/xalan-j/dist/xalan-j_2_1_0.tar.gz

4.37. xerces

XML tools for Java that include DOM, DOM 2, SAX, and SAX 2 parsers, JAXP 1.2, as well as support for XML Schema 1.0.

Version: 2.4.0

Source: http://xml.apache.org/dist/xerces-j/Xerces-J-bin.2.4.0.zip

4.38. xerces-c++

Version: 3.0.0

5. Build Preparation

This section describes the initial steps you need before building the VICAR source.

VICAR makes extensive use of the $VICCPU environment variable (which is automatically set by vicset1.csh, described later). This environment variable contains the platform name – the type of machine you are building on. This is used throughout the VICAR and external trees in directory and file names to differentiate files that could be different on different platforms. VICAR supports multiple platforms under one physical tree; this is how we do it at MIPL. However, these instructions assume you are building for only one platform. If you have a shared filesystem, however, you could try a multi-platform build. Just merge the external trees, and the rest of the process should work.

The two primary $VICCPU values are:

```
x86-64-linx
mac64-osx
```

where x86-64-linx is 64-bit linux. Note the intentional misspelling of "linx" in that name; that is done so the name fits in 11 characters. Porting to a new platform is possible but way outside the scope of this document; contact us if you really need to try this. Note that for the time being, "arm"-based Macs still use mac64-osx. Finally, we have not actually built VICAR on an ARM machine creating x86-64 executables; we build on an Intel machine. If you want to do this, you might need to add options to the compile statements to set the target as Intel. We hope to support ARM fully in the near future.

The pre-built VICAR binaries were built on the following OS platforms:

- 64-bit Red Hat Enterprise Linux 7.3; gcc/gfortran 4.8.5
- 64-bit Mac OS X 10.9.5 Mohave; gcc 4.2.1 (llvm 6.0)/gfortran

Important! VICAR is built on csh/tcsh. All of the build instructions below, as well as the VICAR setup scripts, assume you are using csh or tcsh as your default shell. If it is not your default, you should start up tcsh in the windows you use to work with VICAR (just type "tcsh"). Using other shells is possible; VICAR programs don't actually care. But you'll have to set up the necessary environment variables on your own. This is very much not worthwhile for building, but it might be useful for running programs, as only a few if any environment variables are needed for any given program. Experiment.

5.1. Building VICAR Using Pre-built External Libraries

To build VICAR with the pre-built external libraries, git clone or unpack the git tar of the VICAR source code into /usr/local/vicar/v5.0, and unpack the externals tar into the same. Then create a link from the VICAR source to the externals directory. So for example, using the pre-built 64-bit linux externals libraries:

```
% cd /usr/local/vicar/v5.0
```

```
% gunzip vicar_open_ext_x86-64-linx_5.0.tar.gz
% tar xf vicar_open_ext_x86-64-linx_5.0.tar
% mv vicar_open_ext_x86-64-linx_5.0 external
% git clone https://github.com/NASA-AMMOS/VICAR
% mv VICAR/vos vos
% cd vos
% setenv V2TOP `pwd`
% source build_open_vicar.csh >& build_open_vicar.log &
% tail -f build_open_vicar.log
```

The examples here are for x86-64-linx, but the commands are the same for mac64-osx, just replace those strings.

Note that if you pull the VICAR source via the auto-generated git tarball, replace the git and ln commands above with *something like*:

```
% gunzip nasa-VICAR-479f5fb.tar.gz
% tar xf nasa-VICAR-479f5fb.tar
% ln -s nasa-VICAR-479f5fb/vos  vos
```

6. Build Instructions

Now you're ready to build! Although the build is composed of many parts, we have created an overall build script to run them all. The following assumes you are using the multi-platform distribution (vicar_open_5.0.tar.gz), and that you have downloaded the pre-built externals tree (e.g. vicar_open_ext_x86-64-linx_5.0.tar.gz). Creating the externals link below is necessary only if you start from a Git clone.

For Linux and Mac OS X use build_open_vicar.csh:

```
% setenv V2TOP /usr/local/vicar/v5.0/vos
% cd $V2TOP/..
% mv vicar_open_ext_x86-64-linx_5.0 external
% cd $V2TOP
% source build_open_vicar.csh >& build_open_vicar.log &
% tail -f build_open_vicar.log
```

(obviously, adjust V2TOP as needed). Running in this way puts the build in the background, so it will continue if you log out. You can ctrl-C the tail at any point without affecting the build.

When the build is finished, review the logs. While several warnings are likely, there should be few if any outright errors, if all goes well. If there are, dive in and start fixing! Note that you can directly call any of the sub-part builds from the build_open_vicar.csh script if you want. For example, if the Java build fails, you can re-run just the Java build without rerunning everything else. As an example, to search the log for errors, enter

```
% grep -i error build_open_vicar.log
```

This should show about 27 lines of compiler output related to things that have "error" in their names, though there should be no actual compiler errors.

A simple VICAR aliveness test is included in the VICAR Quick-Start Guide, available on the VICAR open source web site: http://www-mipl.jpl.nasa.gov/vicar_open.html .

As stated earlier, please contact us with problems and we will try to help, but cannot guarantee it. Especially let us know of any errors or omissions you find in these instructions.

Good luck!