Building VICAR

Open Source version

Version 2.0

2016-05-22

Prepared by:

_____

Walter Bunch

_____

Robert Deen, VICAR Cognizant Engineer

Jet Propulsion Laboratory

California Institute of Technology

Pasadena, California

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. It is included in the delivery under "vicar_open_2.0/docsource/vicar" as well as on the VICAR open source web site: http://www-mipl.jpl.nasa.gov/vicar_open.html .

1.1. 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:

- Mac OS X
- Solaris 10

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 5.x for our Linux distribution, though we also do some testing on 6.x. The build "should" work with other distributions but you may find quirks that need fixing.

2. 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.

2.1. Obtaining VICAR

VICAR can be obtained from the following link:

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

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

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 available pre-built VICAR+externals tar files are:

- vicar_open_bin_sun-solr_2.0.tar.gz
- vicar_open_bin_x86-64-linx_2.0.tar.gz
- vicar_open_bin_x86-linux_2.0.tar.gz
- vicar_open_bin_x86-macosx_2.0.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_2.0.tar.gz
- vicar_open_ext_sun-solr_2.0.tar.gz
- vicar_open_ext_x86-64-linx_2.0.tar.gz
- vicar_open_ext_x86-linux_2.0.tar.gz
- vicar_open_ext_x86-macosx_2.0.tar.gz

The first includes the external libraries for all four platforms, while the rest include only the external libraries for the named platform. Note the intentional "linx" in the 64-bit Linux version.

2.2. 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.

2.2.1. Installing a Pre-Built VICAR

So for example, using the pre-built 64-bit linux tarball:

```
cd /usr/local/vicar/v2.0
gunzip vicar_open_bin_x86-64-linx_2.0.tar.gz
tar xvf vicar_open_bin_x86-64-linx_2.0.tar
```

You will then end up with these directories:

```
/usr/local/vicar/v2.0/vicar_open_2.0
/usr/local/vicar/v2.0/vicar_open_ext_x86-64-linx_2.0
```

where the "vicar_open_2.0" directory contains "vicset1.source", "p2", etc. while "vicar_open_ext_x86-64-linx_2.0" contains "JAI", "tae", etc. Each platform-specific tarball, or cloning from git, will result in different directory names. Note that a softlink called "external" will be created in the vicar_open_2.0 directory pointing to the external tree during the build. This is the reason for placing vicar_open_2.0 in the versioned directory v2.0.

The V2TOP environment variable should be set to the "vicar_open_2.0" tree: the one that contains VICAR itself.

```
setenv V2TOP /usr/local/vicar/v2.0/vicar_open_2.0
```

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). Having unpacked the pre-built VICAR and externals, and having set V2TOP, you would want to jump into the "Starting up VICAR" section of the companion document *VICAR_guide_2.0*.

3. 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.

3.1. 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

3.2. geotrans

Library libdtcc.a is built from GEOTRANS 2.2.3- Geographic Translator, a geographic projection code library. It has been enhanced to allow compilation on Sun's cc: Forte Developer 7 C 5.4 2002/03/09 and gnu's gcc version 2.95.3 20010315 (release). Also, two of the projections, polarst.c and utm.c were slightly corrected/enhanced.

Version: 2.2.3

Source: http://www.nima.mil/GandG/geotrans/

3.3. 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

3.4. jai-ext

JAI-EXT is an open-source project which aims to replace in the long term the JAI project. JAI is provides a set of high level objects for the 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.

3.5. 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

3.6. 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

3.7. 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

3.8. 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: 1.1.1 release candidate 8

Source: http://jogl.dev.java.net

3.9. 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

3.10. 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: .97

Source: http://heasarc.gsfc.nasa.gov/docs/heasarc/fits/java/v0.9/v0.97

3.11. 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

3.12. 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

3.13. SPICE

The SPICE toolkit computes spacecraft and solar system geometry.

Version: 6.4

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

3.14. 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

3.15. 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/

3.16. 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: 2.1.0

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

3.17. 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. 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 four primary $VICCPU values are:

```
x86-linux
x86-64-linx
sun-solr
x86-macosx
```

where x86-linux is 32-bit and 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. Also note that for Mac, we build for 32-bits explicitly, but that works on either a 32 or 64 bit platform. Porting to a new platform is possible but way outside the scope of this document; contact us if you really need to try this.

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.

4.1.1. 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/v2.0, 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/v2.0
gunzip vicar_open_ext_x86-64-linx_2.0.tar.gz
tar xf vicar_open_ext_x86-64-linx_2.0.tar
git clone https://github.com/nasa/VICAR.git
mv VICAR/vos ./vicar_open_2.0
```

```
cd vicar_open_2.0
ln -s ../vicar_open_ext_x86-64-linx_2.0 external
setenv V2TOP `pwd`
source build_open_vicar.csh |& tee build_open_vicar.log
```

Note that the build_open_vicar.csh works for the 32&64-bit Linux and 32-bit MacOS distributions, but the alternate build_open_vicar_sol.csh should be used for Solaris.

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

```
gunzip nasa-VICAR-479f5fb.tar.gz
tar xf nasa-VICAR-479f5fb.tar
mv nasa-VICAR-479f5fb/vos  ./vicar_open_2.0
```

4.2. Prerequisites for Linux-32

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. Two locations are:

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

http://xorg.freedesktop.org/archive/X11R6.9.0/

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.

It is assumed that you have X-windows and Motif installed; if not, do so. Also, gcc, g++, and gfortran are required. While gcc and g++ are ubiquitous, gfortran may not be. Follow standard installation procedures for it.

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.

4.3. Prerequisites for Linux-64

The "imake" program is also needed for linux-64; see the linux-32 section.

Building on Linux-64 requires version 4.4 or later of the gcc compiler suite (due to bugs in earlier versions). Because the MIPL systems still default to gcc 4.1.2, we use "gcc44", "g++44", and "gfortran44" to force using the newer compilers.

If your system does not have executables named with the "44" but the gcc suite is in fact 4.4 or later (gcc –version will tell you the version), then you have two options. Of these, the second is recommended:

1. Create a softlink in /usr/bin (or anywhere in your $path) for gcc44, gfortran44, and g++44, which point at gcc, gfortran, and g++ respectively. This requires system admin privilege if you use /usr/bin, but you can put the links anywhere and add an entry to $path.

2. Edit the build template files to remove the "44"s. The following files (relative to $V2TOP, the top of the VICAR tree) need to be edited:

```
util/imake.config
util/vicsys.tmpl
MotifApp/Makefile.x86-64-linx
tae53_changes/config/x86_64_linx.cf
```

4.4. Prerequisites for Solaris

Solaris also requires imake. However, currently it is still included in the Solaris distribution. If this changes in future releases, you will have to find a version. Solaris 11 may be problematic in this regard, but we only "officially" support Solaris 10. It appears that the packages "developer/build/imake" and "developer/build/makedepend" may provide imake support under Solaris 11, but this has not been tried.

We use the Sun compilers (C, C++, and Fortran) rather than the gcc suite.

4.5. 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 (10.10.5) 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.

4.5.1. imake

A version of imake for Mac OS X is included in the delivery, so you do not need to do anything about imake. However, there are a few other uncommon packages you will need.

4.5.2. xquartz

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

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.

4.5.3. OpenMotif

After installing X-windows, install the OpenMotif library:
external/OpenMotif/v2.1.32/x86-macosx/openmotif-compat-2.1.32_IST.macosx10.5.dmg

This package was obtained from
http://www.ist-inc.com/motif/download/motif_files/ IST no longer
supports OpenMotif on the Mac, so we are including it with this
distribution. The license information also is provided in x86-macosx/license.

4.5.4. Gnu Compilers

We use gcc, g++, and gfortran. 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:

http://cran.r-project.org/bin/macosx/tools/gfortran-4.2.3.pkg

This is for OS X 10.5+, signed, 64-bit driver. Again, test with "gfortran --version".

VICAR is currently built in 32-bit mode (-m32 option to all the compilers). This is to provide compatibility across all Intel-based Macs. 32-bit-only processors have not been sold in some time, so we may change this to 64 bits in the future. You can try doing so on your own (look for –m32 in the files described above in the Linux-64 "44" change in Section 4.3). But, you will also most likely need to recompile all the externals as well. Thus it is not a small job, and probably not worth it for most users.

5. 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_2.0.tar.gz). **For Linux and Mac OS X** use build_open_vicar.csh:

```
setenv V2TOP /usr/local/vicar/v2.0/vicar_open_2.0
cd $V2TOP
./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.

For Solaris, use build_open_vicar_sol.csh instead.

When the build is finished, review the logs. While some 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.

A simple VICAR aliveness test is included in the VICAR Quick-Start Guide. As mentioned in Section 1, this is included in the delivery under "vicar_open_2.0/docsource/vicar" as well as 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!