How To Make JVSIP

Version 0.91

March 13, 2012

Introduction

About

This document is included with a VSIPL distribution I have called Judd VSIP (JVSIP) to differentiate this VSIPL implementation from other implementations. The VSIPL C library is governed by the Version 1p3 VSIPL Specification. I also include some python tools. These have no governing specification at this time; but I may try to write one once I have a proof of concept developed.

I provide source code. The purpose of this document is to help users in turning the source code into products.

Basics

These instructions assume you have downloaded the jvsip distribution from the github repository using git. This looks (something) like:

```
> git clone git://github.com/rrjudd/jvsip.git jvsip
```

Note the ' > ' is supposed to be the system prompt and not something you type in. Also note I am still learning git; so I won't spend much time trying to teach it here (the blind leading the blind); but when you do this download you get a full fledged git repository where you can do your own development work if you so desire. For additional information on git I suggest searching the internet.

The directory **jvsip** will reside somewhere in your path. Here we assume **$HOME** is the spot in the directory where **jvsip** resides; however you can put it anywhere. If you don't want to work directly in **jvsip** you may want to clone a working directory. For instance

```
> git clone $HOME/jvsip $HOME/local/src/jvsipWorking
```

Once you build the JVSIP products doing git status (for instance) will show a lot of object files, build directories, etc. which you probably don't want as part of your git repository. I find that by working in **jvsipWorking** and then updating **jvsip** when a product becomes mature makes it easier to keep cruft out of my git repository. As I become more familiar with git perhaps I will find this redundant.

Making and installing the C vsipl library.

There is a makefile in the top level `jvsip` directory. This top level makefile is for creating the C VSIPL library. You should use GNU's make. I am not expert on makefiles; and I don't use the configure option popular with more complicated distributions. To change something you need to edit the Makefile.

Typing

```
> make
```

in the **$HOME/jvsip** directory should create a **libvsip.a** in the **c_VSIP_src** directory; create a **$HOME/jvsip/lib** directory and a **$HOME/jvsip/include** directory; and then copy **libvsip.a** into **$HOME/jvsip/lib** and copy **vsip.h** into **$HOME/jvsip/include** .

Note that makefiles included with testing and example codes for traditional C VSIPL will look for these items in the **$HOME/jvsip/lib** and **$HOME/jvsip/include** directory.

You can also go directly in **$HOME/jvsip/c_VSIP_src** and make there. This will just create the **libvsip.a** file in **$HOME/jvsip/c_VSIP_src**.

If you want to install into **/usr/local** (or some other standard location) then all you need to do is copy **libvsip.a** and **vsip.h** to the locations you want them.

Making the baseline test

Inside **$HOME/jvsip/c_VSIP_testing** is a lot of (fairly ugly) code designed to exercise the C VSIPL routines enough to give one some confidence that most things work. Most of these codes are available in TVCPP but this distribution is not the same as TVCPP. You should not put files with a '.h' extension in this directory unless it is a test. See documentation in **$HOME/jvsip/doc** or the testing READMEs for more information.

Note the default mechanism for building the tests assumes you have already created the library in **./jvsip/lib** and that **./jvsip/include/vsip.h** exists. This should already be the case if the steps above have been done.

I make this with a shell script (called **gen_all.sh**). To create the tests do

```
> cd $HOME/local/jvsip/c_VSIP_testing
> sh gen_all.sh
```

This should create a file called **test_all.c** and compile it into an executable called **test_all**. To run do

```
>./test_all | grep error
```

or

```
>./test_all >output
```

or just

```
>./test_all  .
```

This last is not recommended. There is a lot of output. The first option should give you nothing (a good thing). The second option lets you search **output** to see the results.

See C VSIP testing document for additional info on writing your own tests in the testing directory.

Python

The directory **$HOME/jvsip/python** contains sub-directories for various python modules. Python is not part of the VSIPL specification; however, it is easy to make a VSIPL python module using the SWIG tool and the C VSIPL code already developed. This allows interactive VSIPL development using the python environment. This simple encapsulation is done in the **$HOME/jvsip/python/vsip** directory.

I also include a vsiputils module in the **$HOME/jvsip/python/vsiputils** directory. This module adds some utility functionality but its main purpose is to overload python functions to flatten the name space. I eventually plan to do a pyvsip which will be a proof of concept for a python VSIPL. The vsiputils module is the first step toward that goal.

Finally I have a module I call vsipUser contained in the **$HOME/jvsip/python/vsipUser** directory. This is just codes for whatever functions a user might find handy but are not really part of the main VSIPL specification.

Creating and installing the vsip python module

The vsip python module uses the same source code as the C VSIP Library; however, there is no requriement to first build the C VSIP Library. The steps described below will build a library that python will use for the module and there is no need to build the C library described above if you are only interested in python.

To make the vsip python module `cd` to **$HOME/jvsip/python/vsip**. Use swig to create the wrapper files and vsip.py. Then use **setup.py** to create and install the vsip module. This looks like

```
>swig -python vsip.i
>python setup.py install
```

Note the swig I was using was SWIG Version 2.0.4. I believe this version should make a wrapper suitable for python 2.6, 2.7 or 3.2

There are several ways to do the install. You can do a build first using

```
>python setup.py build
>python setup.py install
```

and then do the install as above.

If you don't have permissions to install you can also install into a user space using

```
>python setup.py install –user
```

The python vsip module is a direct copy of the C VSIPL library with a minor addition to support VSIPL index types in the selection operations. There is a *VSIP python module* document available in **$HOME/jvsip/doc** with more information on this module and how to use it.

Creating and installing the vsiputils python module

The directory **$HOME/jvsip/python/vsiputils** contains the file **visputils.py**. Currently the best documentation for the VSIP utilities module is just reading the file. The module vsiputils this is not well tested and is very much alpha code. The module requires the vsip module. It overloads VSIPL functionality and adds some support. To install I generally do

```
> python setup.py install
```

Creating and installing the vsipUser python module

The file **$HOME/jvsip/python/vsipUser/vsipUser.py** contains code for the VSIP User module. This module requires vsiputils and adds support for VSIPL user code. (for instance printing). To create the module I do (for instance)

```
>python setup.py install –user
```

Currently this module is small but I expect additional functionality will be added as time goes by.

Objective-C

The C VSIPL library works well with objective-c, and I have started to include some examples. Currently I only support objective-c on an Apple (OS X) platform using the Xcode IDE. I have found creating and installing a VSIPL framework for use with this environment is helpful. There is a separate document which discusses some of my experience in an OS X environment and I have include an Xcode project (in the **AppleXcode/jvsipF** directory) to aid in creation of a VSIPL framework. Any examples dealing with objective-c code may expect this framework to be available in order to build without some extra work. Note if you don't work on an OS X platform then you can ignore this information.