Using JVSIP in an OS X Environment

Version 0.5

March 10, 2012

Introduction

This document is to present how I use JVSIP in an Apple environment using Xcode as a development tool in the hope that it will be helpful for others who develop on this platform. Note I do not consider myself to be expert on OS X or X-Code or on any library other than VSIPL. Instructions in this document work for me but may not be the best way to do things. In other words people who are expert may have other advice.

Some Background

I normally develop JVSIP and associated products using Apple computers running OS X and Xcode; however, JVSIP is not specific to Apples operating system and my development environment is not specific to Xcode. I also run and test in Linux and other unix style environments if they are available. I also use older tools such of vi, make, etc. I expect, perhaps with some work on compiling the library, for JVSIP to work in any environment which supports an ANSI C 89 compiler. I don't have a Microsoft environment but the baseline JVSIP library should work there as well.

Parts of the JVSIP distribution which are not limited to ANSI C 89 may be more difficult to use. In particular my objective C products I currently only test on Apples.

On an unrelated (to OS X) note python work should be more portable; but getting a python environment installed on my computer, at least one that supported matplotlib[1], numpy[2], and scipy[3] as well as ipython[4], I found to be mildly challenging. Unless others have a compatible python environment they may have trouble. Python environments do not have a single standard or version. Although all these packages have stable distribution downloads I found my best luck with downloading the newest thing from git and installing directly.

The VSIPL modules included with jvsip should work fine on python 2 or 3, but some of my Python examples may not work everywhere without additional work.

Working with Xcode

I am not going to try to produce an Xcode manual. With every release there seem to be major change to the Xcode user interface. Consequently there are many books available, all of which spend significant effort on the user interface, and unless you have the latest and greatest book they are probably out of date. Even Apples documentation, which is voluminous, is frequently wrong or out of date. Consequently I will just state that I am using for this manual Version 4.2.1 of Xcode and my recipes are based upon what my experience is with this particular Xcode. Your usage may vary.

---

1 http://matplotlib.sourceforge.net/ https://github.com/matplotlib/matplotlib
2 https://github.com/numpy/numpy
3 https://github.com/scipy/scipy
4 https://github.com/ipython/ipython

Notation

I will attempt to maintain a particular style for some items.

For text corresponding to a command line I will use a courier font with a blue tinge and a ">" at the front. For instance:

```
> cd $HOME/local/jvsip
```

For text corresponding to a file name, directory, or group in Xcode I will use helvetica 12 point bold. For instance
"The framework will show up in the navigator window under the group called **Products."**

Menu items will start with the top level with an arrow (->) between items in the order of selection. For text corresponding to an Xcode menu Item I will use helvetica neue 12 point bold. For instance:
"create a new file using **File->New File"**

For other items inside Xcode, such as pop-up menus and labels I will use helvetica neue 12 point bold italic. For Instance:

"pick ***Coca Framework***."

JVSIP Framework

This section covers building, installing and working with an Xcode framework for VSIPL. Note you can also just work with a standard VSIPL library and header using historical methods. I find the Framework method to be nice when working with Xcode.

Building The Framework

The easiest way to work with VSIPL on an apple platform is to use xcode to create a framework; and then install the framework into /Library/Framework. The method I use to do this follows.

Note I am using Xcode Version 4.2.1 as I write this example. I have decided to call the framework **jvsipF**. I will assume the user has already used Xcode at least a bit. This is not an example for beginners.

1) Start Xcode and start a new project.
   File->New Project
   In the window that pops up there will be a list of templates for your project. Select ***Framework & Library.***
2) There will be a selection of choices on the right. I pick ***Cocoa Framework*** because it is the only one that is called a framework. However we don't have any dependencies other than a c89 compiler and the included jvsip source code.
3) You then go through the normal process of naming the project and saving it. You can call this anything you want. For the purpose here I will call it **jvsipF**. Note what you call it is the name of your framework.

4) You should see two file, **jvsipF.h** and **jvsipF.m** inside the **jvsipF** group of the new project. Select those two files and push delete. When asked select delete. We don't need these files anymore.
5) Select the group called **Frameworks**. Push delete on your keyboard. You don't want to actually delete anything here (and the system shouldn't let you) so select the button that says ***Remove References Only***
    a) Don't do anything with the group that says **Supporting Files** (not sure what these do so we better keep them).
6) Open finder and find the directory inside the jvsip distribution that says **c_VSIP_src**. Make sure that directory is fairly clean (no object files) and drag it underneath the **jvsipF** group.
    a) If you have not removed the makefile inside c_VSIP_src then Xcode will ask if you want to create an external build system project. That box should **not** be checked. We don't want to do this. Press next
    b) You have the option to copy the source to a new directory inside your project or use the source where it is. Either option works so decide for yourself which one you want.
    c) Wait a bit for Xcode to finish it's work.
7) Open the new **c_VSIP_src** group. There is some cruft in there like **Makefile**. Select those and delete them. Use the ***Delete References Only*** button unless you made a copy. You may still want these files; just not represented in your Xcode project.
8) Find the vsip.h file. You can do this by typing vsip.h in the search box at the bottom of the navigator area of Xcode.
    a) Select **vsip.h**
    b) Make sure the utilities area is open(V*iew*->U*tilities*->S*how Utilities*). You should see a tool box icon and the checkbox beside it should be checked. This is under the **Target Membership** section. To the right is a pulldown menu and it should say **Project**. Open the menu and select **Public**. Note this tells Xcode to include **vsip.h** as an include file in your framework package.
9) We should be ready to make. Select
Product->Build For->Build For Running

If everything is OK then Xcode should compile the library and create an framework. For this example the framework would be called **jvsipF.framework**. If not successful review the steps above and any error messages and try to fix. If you find something wrong with my instructions feel free to email me.

Assuming success control click on the f **jvsipF.framework** in the products group and select **Show in Finder.** The directory should say **Release**. If it says **Debug** you probably want to adjust your build for the release build to improve performance. Open the framework directory and look at the header directory. The **vsip.h** file should be in there. If not go back to step 8 again and make sure the target membership of vsip.h is public.

Installing the Framework

I have not found any way to have Xcode build and install the framework in one step. The framework should go into **/Library/Frameworks/.** I generally just copy the **jvsipF.framework** (directory) over manually.

Using the Framework

To demonstrate the use of the framework I will do a simple example. Write a VSIPL Program. The vector add example below is what I will use. Note I have assumed the framework is called **jvsipF**. If you built using a different name use that.

Compile using the command line

Using the command line, assuming the file name is **vadd.c** and the executable target is **vadd** then use.

```
> cc -o vadd vadd.c -framework jvsipF
```

There are many options and compiler incantations. This is the simplest.

For very simple code the command line is probably easiest. For more complicated code where other frameworks from objective C and GUI code is added you probably want to work from within Xcode.

Compile using Xcode

Using Xcode can be a bit obnoxious. For simple testing the command line is short and sweet. Xcode forces enormous numbers of confusing options on the user. I understand very few of them and continually find myself unsure how to make Xcode do what I want.

In this section I will go through how to build this simple example. One should not confuse my instruction with instructions from somebody who is expert on Xcode.

Assuming you have already written the vector add example and tried the command line method this is how to import it into Xcode. I will assume the reader knows how to do some of this stuff from building the framework example above.

1. Start a new project.
   a) Select ***Command Line Tool***, and select next.
   b) From the pop-up menu beside *Type* select `C`.
   c) Insert a project name. I used **vadd**.
   d) Select a place to store it.
2. Now we have a project and there is a file in it called main.c.
   a) Drag **vadd.c** into the project.
   b) The **vadd** example has a main so delete the Xcode supplied **main.c**. We don't need it
3. Add the jvsipF framework to the project.

Simple vadd program

```
1   #include<jvsipF/vsip.h>
2   int main(int argc, char **argv){
3     int init=vsip_init((void*)0);
4     vsip_length n;
5     vsip_index i;
6     vsip_vview_d *a, *b;
7     if(argc > 1)
8       n = (vsip_length) atoi(argv[1]);
9     else
10      n = 10;
11    a = vsip_vcreate_d(n,VSIP_MEM_NONE);
12    b = vsip_vcreate_d(n,VSIP_MEM_NONE);
13    vsip_vfill_d(.1,b);
14    vsip_vramp_d(1,.1,a);
15    vsip_vadd_d(a,b,a);
16    for(i=0; i<n; i++){
17      printf("%f\n",(float)vsip_vget_d(a,i));
18    }
19    vsip_valldestroy_d(a);
20    vsip_valldestroy_d(b);
21    vsip_finalize((void*)0);
22    return init;
23  }
```

a) Select the **vadd** project icon at the top of the Project Navigator list.

b) To the right of the Project Navigator list should be a **Project** called vadd and a **Target** called **vadd**. Select the **Target** vadd.

c) You should see a horizontal list of **Build Settings**, **Build Phases**, **Build Rules**. Select **Build Phases**.

d) The list under ***Build Phases*** should contain an item called ***Link Binaries With Libraries***. Open that list.

e) Select the **+** at the bottom of the opened item. You will see many frameworks. At the top type in **jvsipF**. This is a search. You don't really need to do a search but it speeds up the process.

f) If you see something (as a result of the search) like ***Mac OS X 10.7*** open that. You should see the ***jvsipF*** framework. Select it and click add.

4. You now should be able to select the run button (or **Product->Build For Running**) and the code should run.
5. Not the vadd code has a default length of 10 or you can pass a value in via the command line. To provide a (command line) argument directly in Xcode
   a) Select **Product->Edit Scheme**.
   b) Select ***Run vadd*** and open the ***Arguments Passed On Launch***
   c) Add a value by pressing the plus button and typing in the value.

Finding Xcode products and files in a terminal

Xcode has places it puts products and other files. They are not obvious or necessarily easy to get to from a command line by typing on the keyboard.

As a workaround right click on an item in the project navigator list and select ***Show in Finder***. This will open a finder window with that file in it. If you drag the folder at the top of the finder window into a terminal window the path will be displayed. For Instance one can open a terminal window, type `cd` , drag in the path, and hit return. The terminal windows focus is now at the same as the finder window and any products or files in the directory are available to the terminal.

For instance the method of running an executable is dependent on what you are trying to do. If you want to change the argument to the program frequently, or run from a script, running from the command line is probably your best option. If you want to debug, or modify and recompile/run your code, then working from Xcode might be a better option.

For text files frequently the GUI supplied by Xcode is preferable to the historical vi, however for some editing tasks vi (assuming one is vi proficient) is the preferable method.