OpenVSP API & MATLAB/Python Integration

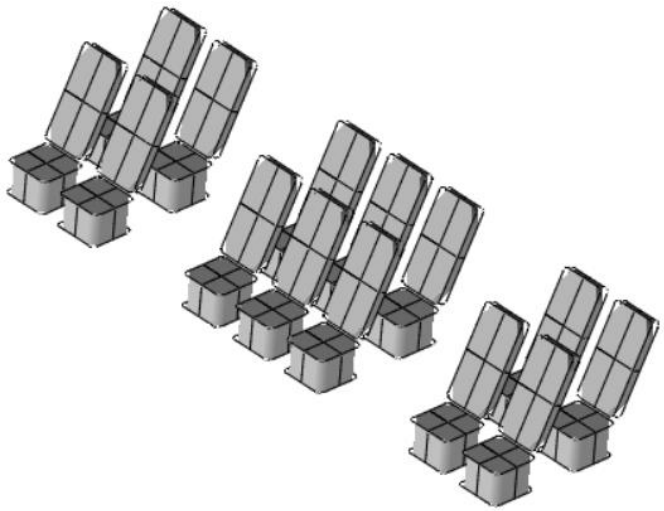
Presented by:
Justin Gravett
justin.gravett@esaero.com
Empirical Systems Aerospace, Inc. (ESAero)

Agenda

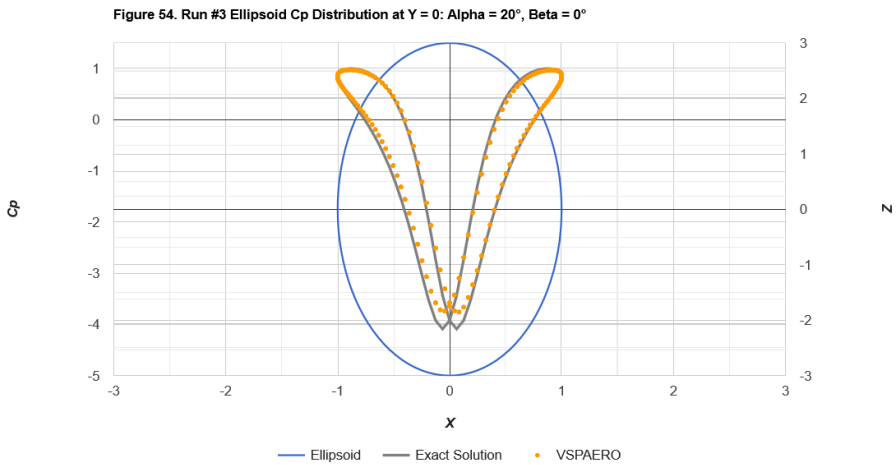
- Introduction
- C++ API
- AngelScript API
- Python API
- MATLAB API Overview
- MATLAB API Build Instructions*

*Recommended: Building OpenVSP from Source Presentation

Introduction

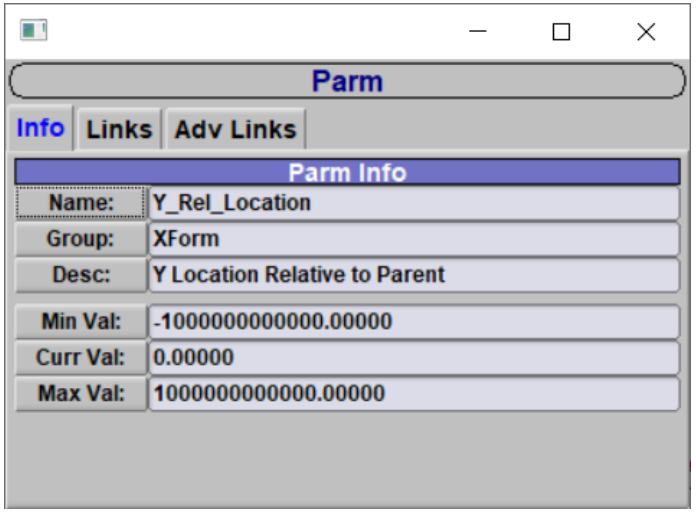


- API: Application Programming Interface
- Languages:
  - C++
  - AngelScript (close to C++)
  - Python
  - MATLAB (experimental)
- What is it used for?
  - Headless operating systems
  - Automation
  - Integration with other codes
  - Test suites
  - Custom Geoms
- How to learn:
  - Old way: study the source code
  - Examples
  - New: Doxygen documentation of functions and classes
  - Workshop presentations



API Functions

- API Documentation: http://openvsp.org/api_docs/latest/
  - Available for direct download
- Analysis and Results Manager:
  - Generic functions to setup analysis inputs, execute analyses, and collect results
  - Strings used to identify available analysis types
    - i.e. "VSPAEROComputeGeometry“
  - Preferred approach for running analyses when available
  - Use “ListAnalysis()” to identify available analysis strings
  - Used “PrintAnalysisInputs(“*analysis*”) to identify available inputs
  - Arrays used for analysis input and result data
- Parms Through the API
  - Generic functions for controlling Parms
  - Values cast to double type in API
  - Parms identified by ID or Container, Group, and Name
- Identifying Parm Info:
  - OpenVSP *.vsp3 file
  - Click and drag GUI device to text editor
  - Click on the GUI Parm button



C++ API

- Basis for all other supported languages – OpenVSP's primary language is C++
- Functions written in VSP_Geom_API (geom_api project)
- Exposes core functionality to higher level abstraction
- Used for API testing
  - Test suites for Parasite Drag, Mass Properties, VSPAERO, and more
    - src -> geom_api folder of repository (apitest project)
    - CFD Mesh test suite in progress
  - Run prior to each release to help catch bugs
  - Intended to be included with continuous integration builds, but currently too time consuming

AngelScript API

- Very similar to C++
  - Must declare variable types
  - “main” function required
- Same language used in Advanced Links and for Custom Geom scripts
- *.vspscript file extension required
- C++ API functions, classes, and enums are registered through AngelScript
- 30+ example scripts provided in scripts directory
  - Demonstrate how to use various API functionality
  - Examples of API automation
  - Most advanced: Master_VSP_VV_Script.vspscript
- 2 ways to run *vspscript files:
  1. OpenVSP GUI: File -> Run Script…
  2. VSPSCRIPT executable:
     - Run through command terminal
     - Example: vspscript -script "scripts/CpSlicer.vspscript"

Python API

- SWIG: Simplified Wrapper and Interface Generator
  - Wraps C++ API for Python
- Python version when importing OpenVSP API must match version used by SWIG
  - Python 3.6: latest version used with OpenVSP distribution
  - Must compile OpenVSP yourself if alternative Python version is desired
- *Windows 3.21.2 initially released without Python API – fixed and updated

Python API

- Installation instructions available in python folder of OpenVSP distribution – see README
  - https://github.com/OpenVSP/OpenVSP/tree/master/src/python_api/packages
  - Open a command terminal in the same directory
  - Recommended (Windows) – use the “dev” API installation version
- Test installation
  - Open command terminal and type:
  - “python”
  - “import openvsp as vsp”
  - “vsp.VSPCheckSetup()” or “vsp.VSPRenew()”

Python API – Common Errors

- DLL load error?
  - Check your python version matches version compiled with OpenVSP
- Missing VSPAERO solver, viewer, or slicer?
  - Copy executables to Python directory

```
VSPAERO solver not found.
VSPAERO viewer not found.
VSPAERO slicer not found.
```

Command Prompt - python

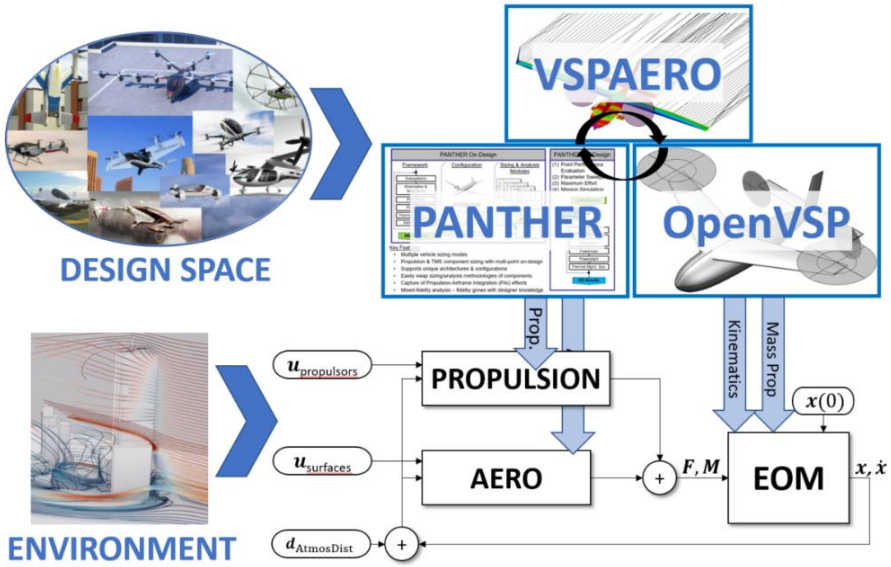
```
Microsoft Windows [Version 10.0.18363.1049]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\justin.gravett>python
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import openvsp
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "c:\users\justin.gravett\desktop\openvsp-3.21.2-win64_b\python\openvsp\openvsp\__init__.py", line 1, in <module>
    from .vsp import *
  File "c:\users\justin.gravett\desktop\openvsp-3.21.2-win64_b\python\openvsp\openvsp\vsp.py", line 13, in <module>
    from . import _vsp
ImportError: DLL load failed while importing _vsp: The specified module could not be found.
>>>
```

MATLAB API

- Advanced and experimental
  - Not recommended for new API users
  - Requires existing software development skills
- Requires unreleased version of SWIG

<u>ESAero Support</u>

- Integrated MATLAB API with PANTHER toolset in Phase I SBIR
- Minor modifications made to OpenVSP – released publicly
  - Duplicated test suite to verify functionality
  - Tried to keep the build process as simple as possible

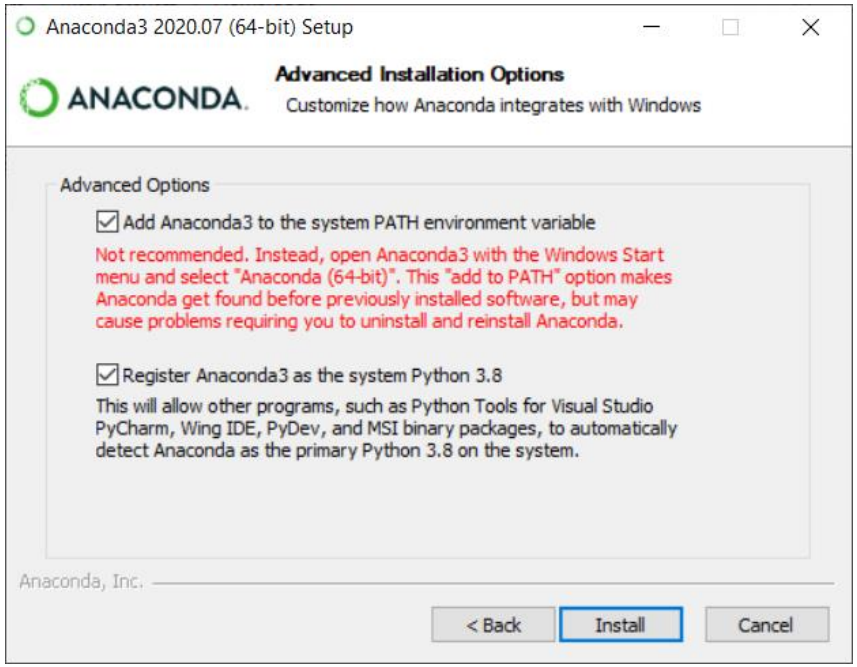


NASA SBIR PHASE I - UAM Disturbance Rejection in Conceptual Design

Software – Python API

Install Anaconda

- Anaconda Version Archive
  - Version 5.2.0 – Includes Python 3.6, compatible with OpenVSP 3.21.2
- Latest Version
  - Only use this version if compiling OpenVSP yourself
- Add to PATH
  - Recommend through installation option
  - Could manually adding to system PATH



SWIG - Python

- Only needed if compiling OpenVSP
- SWIG – Simplified Wrapper and Interface Generator
  - Official distribution
  - Used to provide python interface to C-API
  - http://www.swig.org/download.html
  - Recent CMake issue with OpenVSP finding 4.0.2 – could use 4.0.1 instead

SWIG - MATLAB

- SWIG - MATLAB
  - Unofficial SWIG build with MATLAB bindings
  - https://github.com/jaeandersson/swig.git
- User must compile OpenVSP
- Git required
  - Sourcetree recommended Git GUI
- MinGW, MSYS, & PCRE used

Build Process – Overview

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in MATLAB to verify everything is working

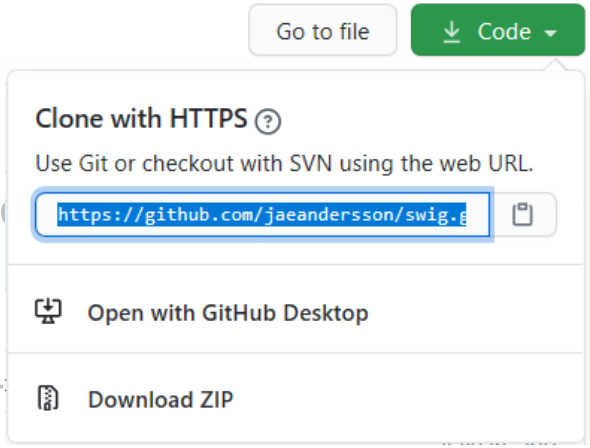
Build Process – Overview

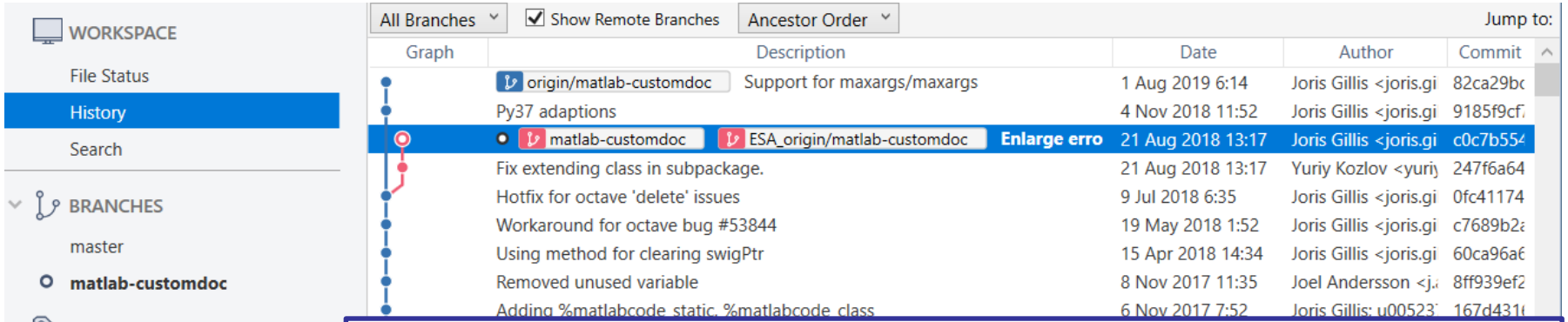
1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in MATLAB to verify everything is working

These are the most challenging steps

Build Process – 1

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that y
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in MATLAB to verify everything is wor





SWIG-MATLAB

Origin: https://github.com/jaeandersson/swig.git

Branch: matlab-customdoc

ESAero branch off of 0fc411742bdfb36c3180d5b671a1da92766af90e

Two commits cherry picked:

Use Latest Git SHA: 82ca29bc9889792856a5d98c5903fb168a5e42f7

Build Process – 2

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to l
5. Compile OpenVSP (Build solution,
6. Delete duplicate SwigClear(self) fu
7. Build some test scripts in MATLAB

Simplified Wrapper an | Search · swig-matlab | VSP - Matlab - Interfa | Getting started on Wi

File | C:/Repo_swig-matlab/Doc/Manual/Windows.html#Windows_swig_exe

3.3.1 Building swig.exe on Windows

If you want to replicate the build of swig.exe that comes with the download, follow the MinGW instructions below. This is not necessary to use the supplied swig.exe. This information is provided for those that want to modify the SWIG source code in a Windows environment. Normally this is not needed, so most people will want to ignore this section.

3.3.1.1 Building swig.exe using MinGW and MSYS

The short abbreviated instructions follow...

- Install MinGW and MSYS from the MinGW site. This provides a Unix environment on Windows.
- Follow the usual Unix instructions in the README file in the SWIG root directory to build swig.exe from the MinGW command prompt.

The step by step instructions to download and install MinGW and MSYS, then download and build the latest version of SWIG from Github follow... Note that the instructions for obtaining SWIG from Github are also online at SWIG Bleeding Edge.

Pitfall note: Execute the steps in the order shown and don't use spaces in path names. In fact it is best to use the default installation directories.

1. Download the following packages from the MinGW download page or MinGW SourceForge download page. Note that at the time of writing, the majority of these are in the Current release list and some are in the Snapshot or Previous release list.
   - MinGW-3.1.0-1.exe
   - MSYS-1.0.11-2004.04.30-1.exe
   - msysDTK-1.0.1.exe
   - bison-2.0-MSYS.tar.gz
   - msys-autoconf-2.59.tar.bz2
   - msys-automake-1.8.2.tar.bz2
2. Install MinGW-3.1.0-1.exe (C:\MinGW is default location.)
3. Install MSYS-1.0.11-2004.04.30-1.exe. Make sure you install it on the same windows drive letter as MinGW (C:\msys\1.0 is default). In the post install script,
   - Answer y to the "do you wish to continue with the post install?"
   - Answer y to the "do you have MinGW installed?"
   - Type in the folder in which you installed MinGW (C:/MinGW is default)
4. Install msysDTK-1.0.1.exe to the same folder that you installed MSYS (C:\msys\1.0 is default).
5. Copy the following to the MSYS install folder (C:\msys\1.0 is default):
   - msys-automake-1.8.2.tar.bz2
   - msys-autoconf-2.59.tar.bz2
   - bison-2.0-MSYS.tar.gz
6. Start the MSYS command prompt and execute:

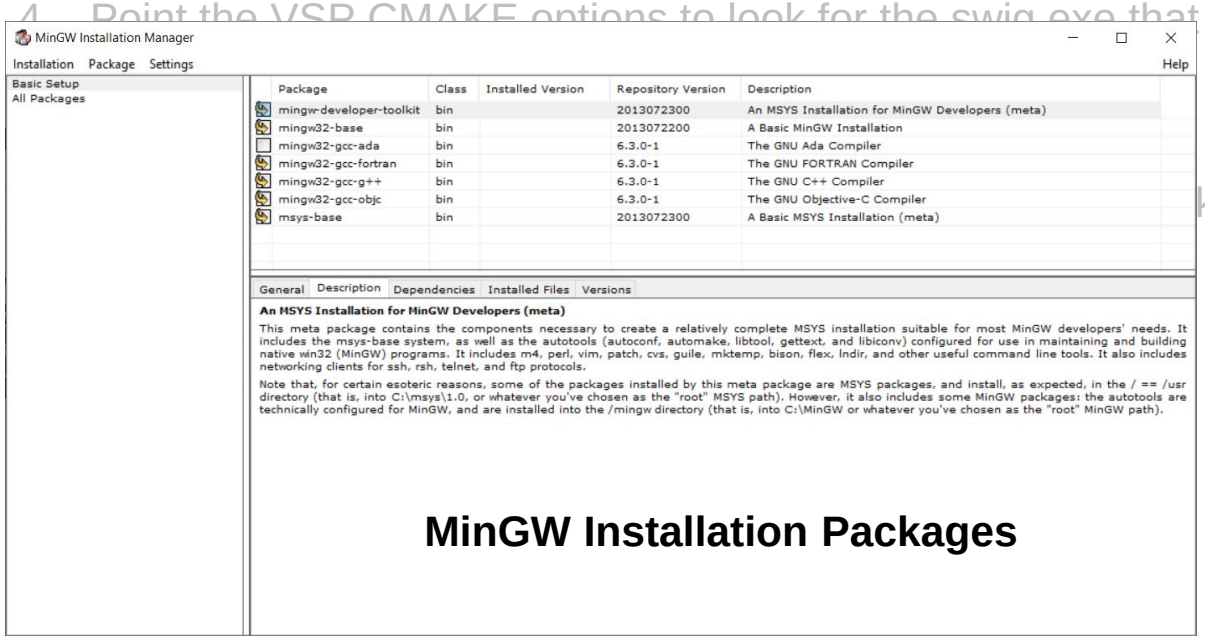
Documentation in source repository: Doc\Manual\Windows.html

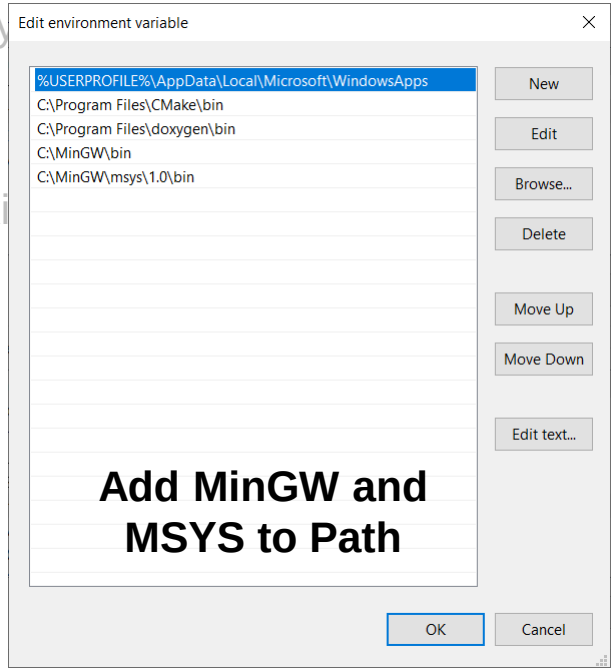
Instructions exist to build using

- MinGW & MSYS – ESAero Tested
- Cygwin
- VisualStudio IDE (challenging)

Build Process – 2

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that y





- No need to install bison, automake, and autoconf separately (already included)
- PCRE (pcre-8.44.tar.gz): ftp://ftp.pcre.org/pub/pcre/ or https://ftp.pcre.org/pub/pcre/
  - Follow instructions to extract in Swig source repository and run “Tools/pcre-build.sh”

Build Process – 2

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the V
5. Compile O
6. Delete dup
7. Build some

Command Prompt

```
checking dependency style of gcc... gcc3
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking dependency style of g++... gcc3
checking whether gcc and cc understand -c and -o together... yes
checking maximum warning verbosity option... -Wall -W -ansi -pedantic for C++ -Wall -W -ansi -pedantic for C
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for ANSI C header files... yes
checking for popen... yes
checking whether to enable PCRE support... yes
checking whether to use local PCRE... /c/Users/justin.gravett/Documents/swig/pcre/pcre-swig-install/bin/pcre-config
checking for a sed that does not truncate output... /usr/bin/sed
checking for pcre-config... /c/Users/justin.gravett/Documents/swig/pcre/pcre-swig-install/bin/pcre-config
checking whether to enable ccache-swig... yes

Checking packages required for SWIG developers.
Note : None of the following packages are required for users to compile and install SWIG from the distributed tarball

checking for bison... bison -y

Checking for installed target languages and other information in order to compile and run
the examples and test-suite invoked by 'make check'.
Note : None of the following packages are required for users to compile and install SWIG from the distributed tarball

configure: line 5873: syntax error near unexpected token `fi'
configure: line 5873: `fi'
```

Solution to error after calling “bash configure”:

1. Delete configure file
2. Rerun “bash autogen.sh”
3. Open configure – delete lines 5773 through 6027

Build Process – 3

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in MATLAB to verify everything is working

C:\OpenVSP\repo\src\CMakeLists.txt - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

CMakeLists.txt

```
ADD_SUBDIRECTORY( xmlvsp )
ADD_SUBDIRECTORY( geom_core )
ADD_SUBDIRECTORY( cfd_mesh )

IF( NOT VSP_NO_GRAPHICS )
    ADD_SUBDIRECTORY( gui_and_draw )
ENDIF()

ADD_SUBDIRECTORY( geom_api )
ADD_SUBDIRECTORY( vsp )
ADD_SUBDIRECTORY( python_api )
ADD_SUBDIRECTORY( matlab_api )

ADD_SUBDIRECTORY( vsp_aero )

SET( CPACK_GENERATOR ZIP )
SET( CPACK_PACKAGE_NAME OpenVSP )
SET( CPACK_PACKAGE_VERSION_MAJOR ${VSPVER_MAJOR} )
SET( CPACK_PACKAGE_VERSION_MINOR ${VSPVER_MINOR} )
SET( CPACK_PACKAGE_VERSION_PATCH ${VSPVER_PATCH} )
```

Tells CMAKE where the source code is for the MATLAB API is

Build Process – 3

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in MATLAB to verify everything is working

> This PC > OS (C:) > OpenVSP > repo > src > matlab_api

Name	Date modified	Type	Size
APITestMain.m	9/12/2019 1:35 PM	MATLAB Code	2 KB
APITestSuite_test.m	9/12/2019 1:35 PM	MATLAB Code	38 KB
APITestSuiteVSPAERO_test.m	9/12/2019 1:35 PM	MATLAB Code	84 KB
assert_delta.m	9/12/2019 1:35 PM	MATLAB Code	1 KB
CMakeLists.txt	9/12/2019 1:35 PM	TXT File	5 KB
Matlabdef.def	9/12/2019 1:35 PM	DEF File	1 KB
vsp.i	9/12/2019 1:35 PM	ideaMaker Print File	1 KB
vsp_common.i	9/12/2019 1:35 PM	ideaMaker Print File	2 KB
vsp_g.i	9/12/2019 1:35 PM	ideaMaker Print File	1 KB

Copy *.i files from python API example

Build Process – 3

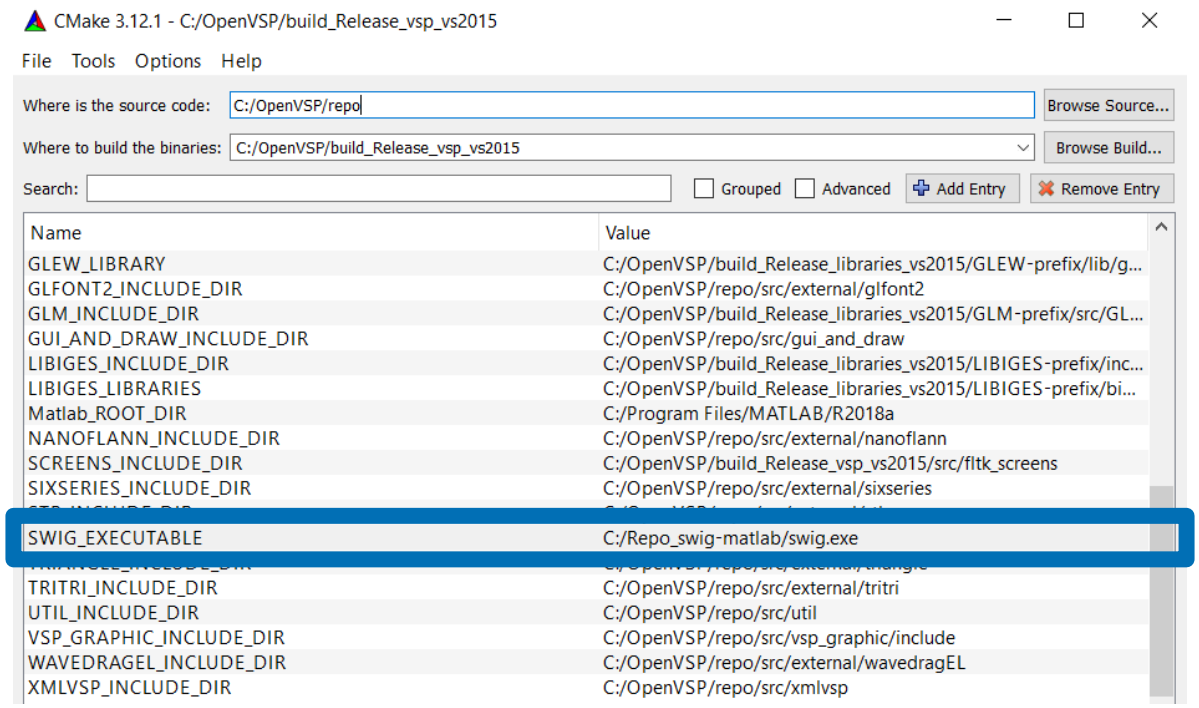
1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in MATLAB to verify everything is working

Key changes to VSP source

- Explicit control of VSPAERO path
- Reference to Matrix.h

Build Process – 4

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in MATLAB to verify everything is working



Build Process – 4

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in matlab to verify everything is working

If you are successful….

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Configure | Generate | Open Project | Current Generator: Visual Studio 14 2015 Win64

```
    PYTHON_LIBRARIES:    C:/ProgramData/Anaconda3/libs/python37.lib
    PYTHON_INCLUDE_PATH: C:/ProgramData/Anaconda3/include
SWIG & MATLAB Found, MATLAB MEX will be compiled.
    swig.exe: C:/Repo_swig-matlab/swig.exe
    Matlab_INCLUDE_DIRS: C:/Program Files/MATLAB/R2018a/extern/include
    Matlab_LIBRARIES:    C:/Program Files/MATLAB/R2018a/extern/lib/win64/microsoft/libmex.lib;C:/Program Files/Mi
    Matlab_MEX_LIBRARY:  C:/Program Files/MATLAB/R2018a/extern/lib/win64/microsoft/libmex.lib
    Matlab_MX_LIBRARY:   C:/Program Files/MATLAB/R2018a/extern/lib/win64/microsoft/libmx.lib
C:/Program Files/MATLAB/R2018a/extern/lib/win64/microsoft/libmex.lib;C:/Program Files/MATLAB/R2018a/extern/lib/wi
Found OpenMP_C: -openmp
Found OpenMP_CXX: -openmp
Found OpenMP: TRUE
Configuring done
```

Build Process – 5

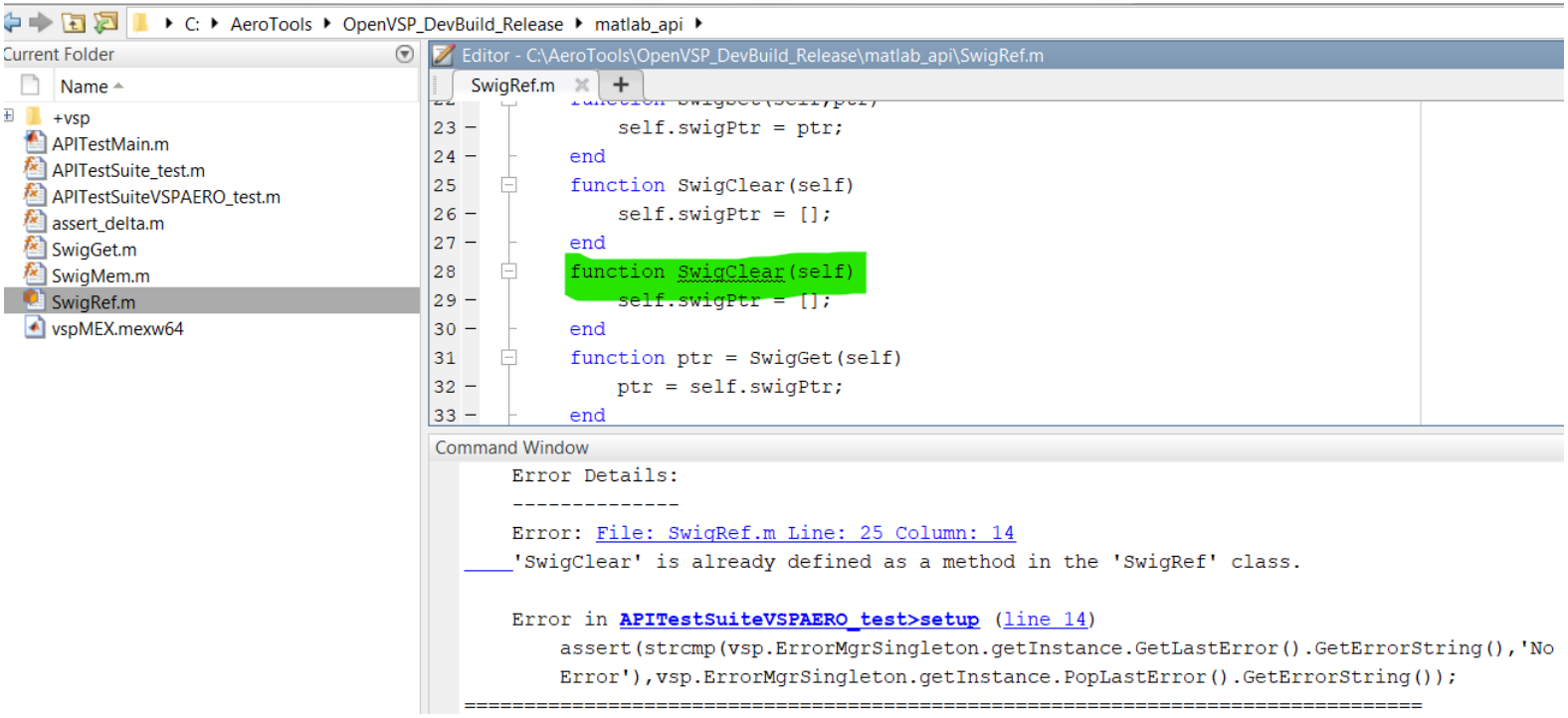
1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in matlab to verify everything is working

Administrator: Windows PowerShell

```
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/+vsp/Y_PROJ.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/+vsp/Z_DIR.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/+vsp/Z_PROJ.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/vspMEX.mexw64
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/SwigGet.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/SwigMem.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/SwigRef.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/APITestMain.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/APITestSuite_test.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/APITestSuiteVSPAERO_test.m
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/matlab_api/assert_delta.m
  -- Up-to-date: C:/AeroTools/OpenVSP_DevBuild_Release/./msvcp140.dll
  -- Up-to-date: C:/AeroTools/OpenVSP_DevBuild_Release/./vcruntime140.dll
  -- Up-to-date: C:/AeroTools/OpenVSP_DevBuild_Release/./concrt140.dll
  -- Up-to-date: C:/AeroTools/OpenVSP_DevBuild_Release/./vcomp140.dll
  -- Up-to-date: C:/AeroTools/OpenVSP_DevBuild_Release/.
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/./vspaero.exe
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/./vspviewer.exe
  -- Installing: C:/AeroTools/OpenVSP_DevBuild_Release/./vspslicer.exe
FinalizeBuildStatus:
  Deleting file "x64\Release\INSTALL\INSTALL.tlog\unsuccessfulbuild".
  Touching "x64\Release\INSTALL\INSTALL.tlog\INSTALL.lastbuildstate".
Done Building Project "C:\OpenVSP\build_Release_vsp_vs2015\INSTALL.vcxproj" (default targets).


Build succeeded.
    0 Warning(s)
    0 Error(s)
```

Build Process – 6

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m
7. Build some test scripts in matlab to verify everything is working



Build Process – 7

1. Clone the swig-matlab repo & checkout matlab-customdoc
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
6. Delete duplicate SwigClear(self) function in SwigRef.m

APITestMain.m

```
% APITestMain
clc
clear all
close all

%% setup globals
global TEST_TOL;
TEST_TOL = 1e-3;    % used in functions for delta assessment

global m_vspfname_for_vspaerotests;
m_vspfname_for_vspaerotests = 'apitest_TestVSPAero.vsp3';

global m_vspfname_for_vspaerofunctionalitytests;
m_vspfname_for_vspaerofunctionalitytests = 'apitest_TestVSPAeroFunctionality.vsp3';

global VSPAERO_PATH
VSPAERO_PATH = [pwd '\..\'];

%% Execute tests
%%% Simple run syntax
% results = run(APITestSuiteTest);
% rt = table(results)

% Complex runner with progress display
import matlab.unittest.TestRunner
import matlab.unittest.TestSuite
% import matlab.unittest.plugins.TestRunProgressPlugin
% import matlab.unittest.plugins.FailureDiagnosticsPlugin

% Create silent test runner and add plug-in to display progress
runner = TestRunner.withTextOutput;
% runner.addPlugin(TestRunProgressPlugin.withVerbosity(matlab.unittest.Verbosity.Detailed))
% runner.addPlugin(FailureDiagnosticsPlugin)

% Run general tests
suite = TestSuite.fromFile('APITestSuite_test.m');
result = run(runner,suite)

% Run VSPAERO unit tests
suiteVSPAERO = TestSuite.fromFile('APITestSuiteVSPAERO_test.m');
result = run(runner,suiteVSPAERO([7]))
```

APITestSuiteTest.m

```
% APITestSuiteT
function tests
    tests = fun
end

%% Fresh Fixture Functions
function setup(~)  % do not change function name
    vsp.VSPCheckSetup();
    vsp.VSPRenew();
end

function teardown(~)  % do not change function name
end

%% void APITestSuite::CheckSetup()
function CheckSetup_test(~)
    % fprintf( 'APITestSuite::CheckSetup()\n' );
    
    vsp.VSPCheckSetup();
    vsp.VSPRenew();

    assert(strcmp(vsp.ErrorMgrSingleton.getInstance.GetLastError().GetErrorString(),'No Error'),vsp.Erro
    % assert( !vsp.ErrorMgr.PopErrorAndPrint( stdout ) );    % //PopErrorAndPrint returns TRUE if there

end

%% void APITestSuite::CreateGeometry()
function CreateGeometry_test(~)

    % fprintf( 'APITestSuite::CreateGeometry()\n' );
    
    types = vsp.GetGeomTypes( );
    assert(numel(types)~=0)

    % fprintf( '\t[ndx]\t%20s\t[geom_id]\t%25s\t[num_geoms]\n','[geom_type]','[geom_type]' );
    n_extra_geoms = 0;
    for i_geom_type = 1:numel(types)
        %% //=== Create geometry =====//
        % fprintf( '\t%d\t%20s', i_geom_type,types{i_geom_type} )
        if ~strcmp(types{i_geom_type},'CONFORMAL')
            geom_id = vsp.AddGeom( types{i_geom_type} );
```

Questions?