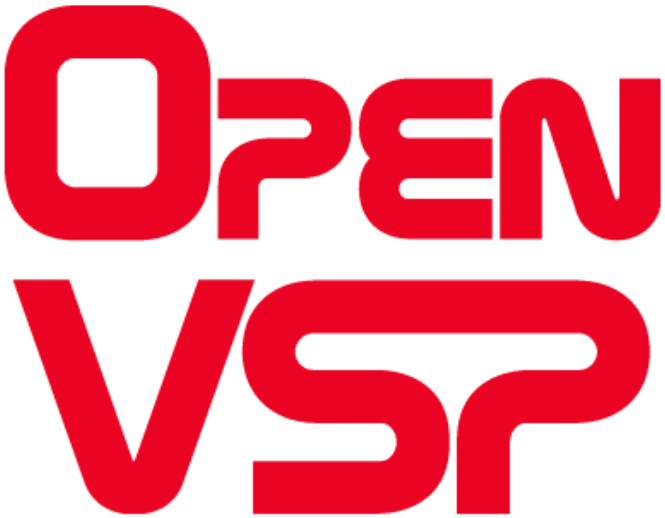
VSP Workshop 2019

OpenVSP – MATLAB API



Presented by:
Nick Brake
Empirical Systems Aerospace, Inc.

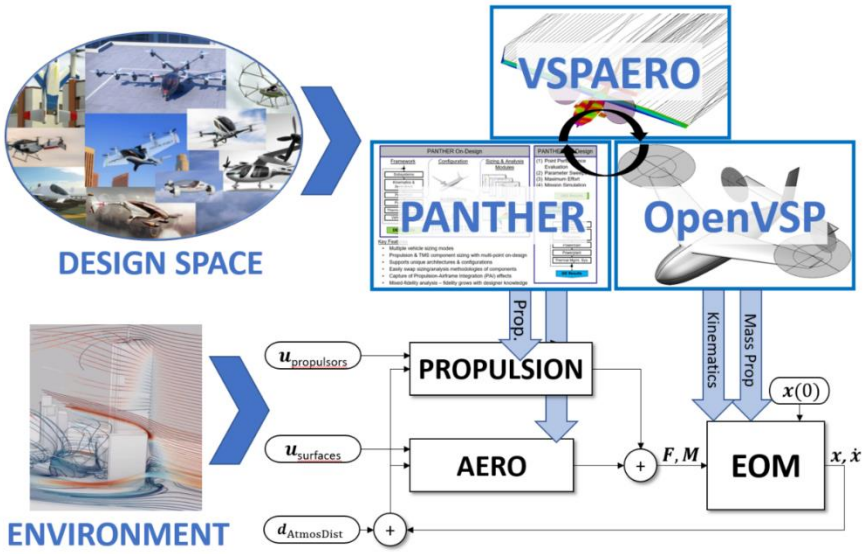
Objectives, Approach

<u>Goal</u>

- Provide API access to OpenVSP & VSPAERO through native Matlab

<u>Approach</u>

- Utilize SWIG to ‘wrap’ C-API for Matlab (similar to Python API)
- Duplicate test suite to verify functionality
- Keep the build process simple as simple as possible
- OpenVSP 3.16.1 & MATLAB 2018



NASA SBIR PHASE I - UAM Disturbance Rejection in Conceptual Design

SWIG & SWIG-Matlab

- SWIG – Simplified Wrapper and Interface Generator
  - Official distribution
  - Used to provide python interface to C-API
  - http://www.swig.org/
- SWIG-Matlab
  - Unofficial SWIG build with matlab bindings
  - https://github.com/jaeandersson/swig.git

Build Process – Overview

1. Clone the swig-matlab repo
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

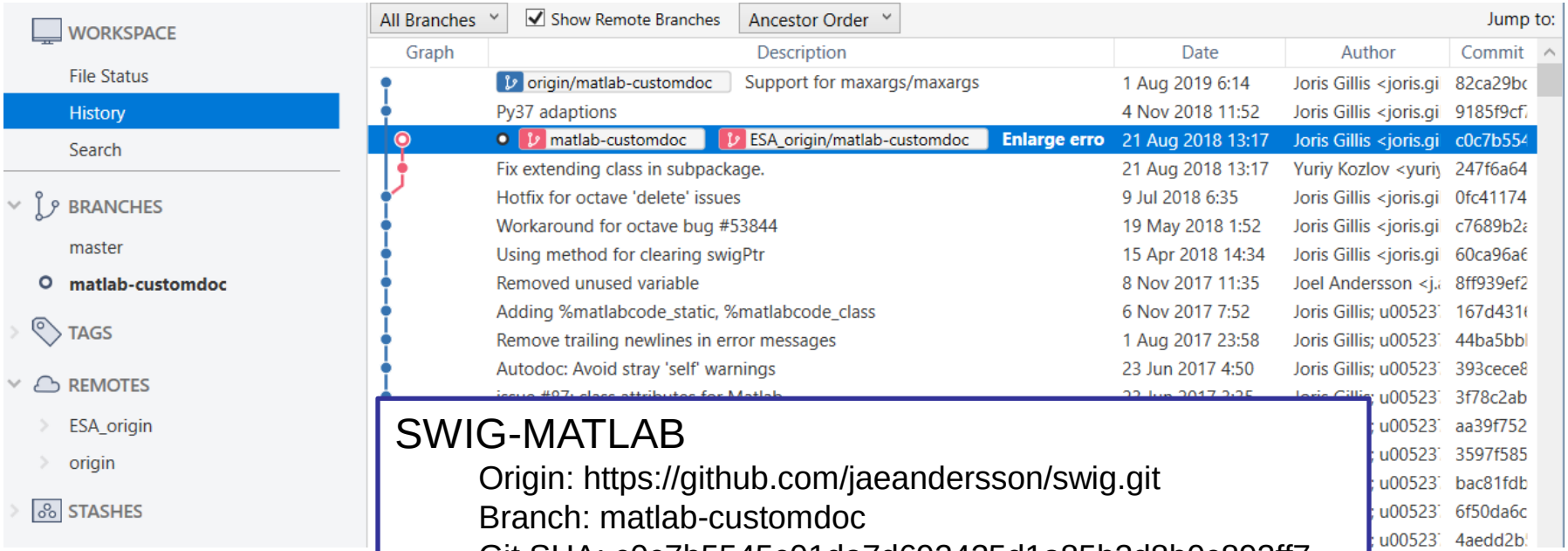
Build Process – Overview

1. Clone the swig-matlab repo
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 was the hardest step. We are looking into getting these changes into the open source release

Build Process – 1. Clone the swig-matlab

1. Clone the swig-matlab repo
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

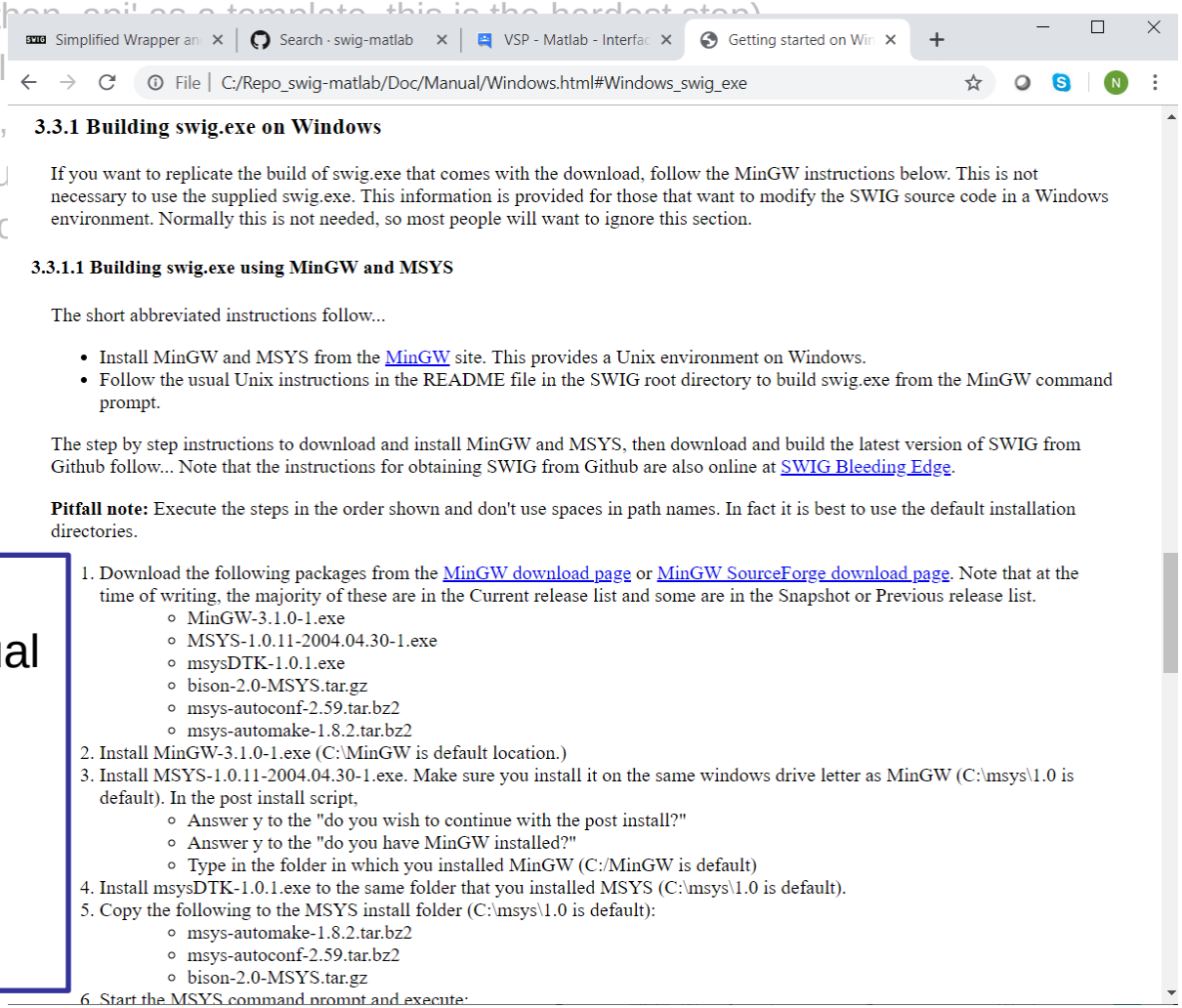


SWIG-MATLAB

Origin: https://github.com/jaeandersson/swig.git
Branch: matlab-customdoc
Git SHA: c0c7b5545c01da7d692425d1a85b3d8b0c893ff7

Build Process – 2. Compile swig-matlab

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



Simplified Wrapper an | Search · swig-matlab | VSP - Matlab - Interfac | Getting started on Wir

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.

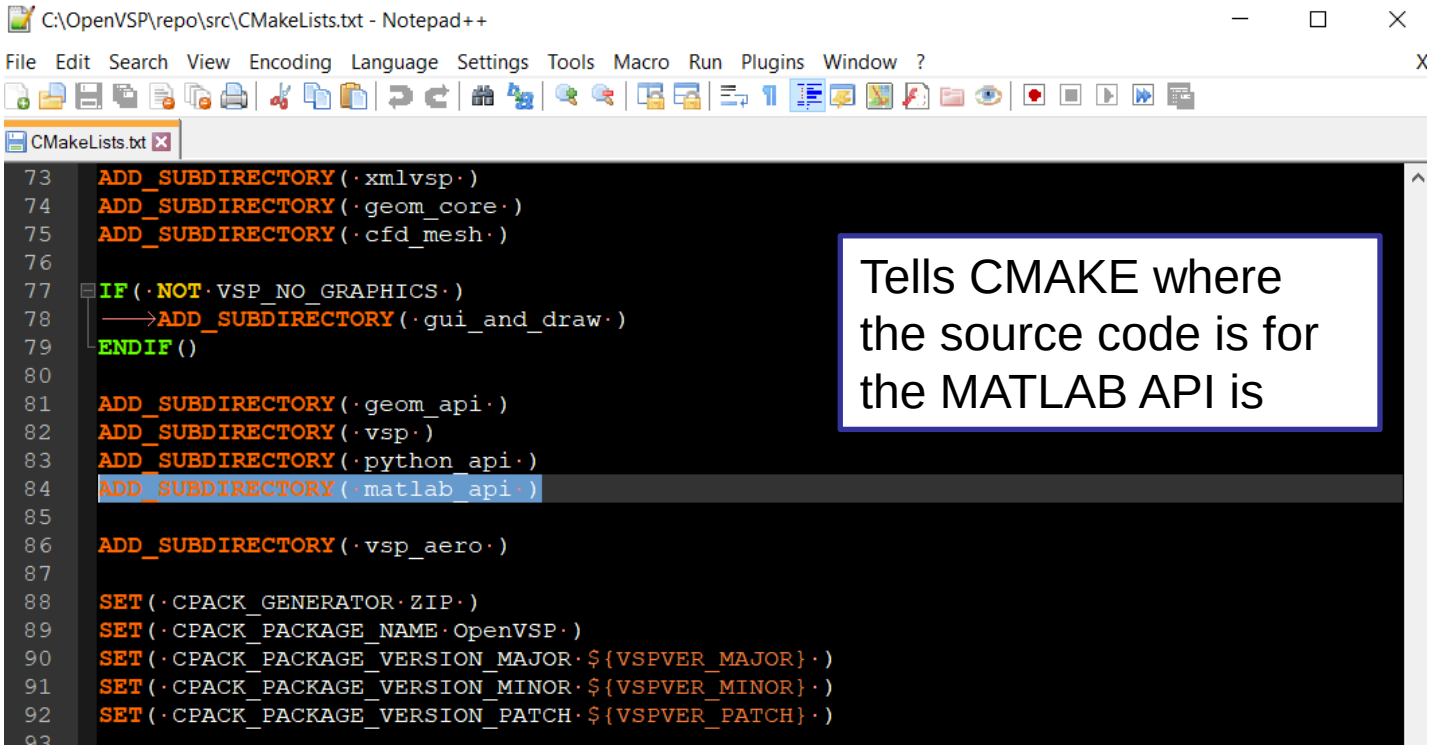
1. Download the following packages from the MinGW download page or MinGW SourceForge download page. Note that at the time of writing, the majority of these are in the Current release list and some are in the Snapshot or Previous release list.
   - MinGW-3.1.0-1.exe
   - MSYS-1.0.11-2004.04.30-1.exe
   - 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:
.\Repo_swig-matlab\Doc\Manual

Instructions exist to build using
- MinGW & MSYS
- Cygwin
- VisualStudio IDE (a bit manual)

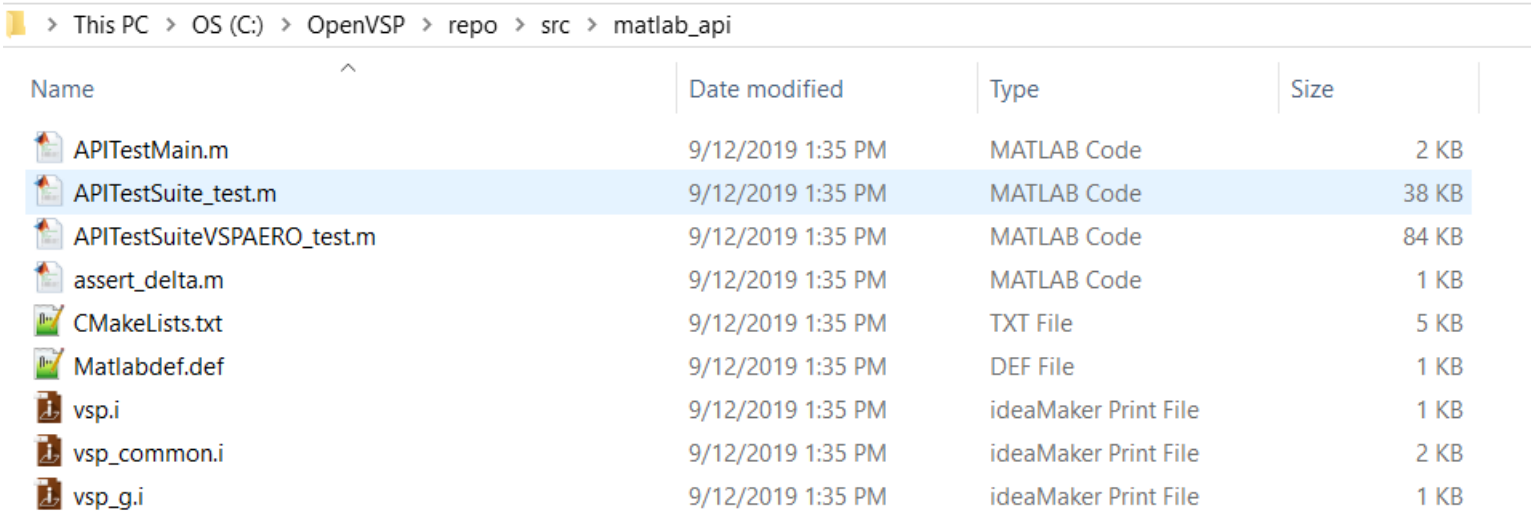
Build Process – 4. Create a 'matlab_api'

1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Point the VSP CMAKE options to look for the swig.exe that you just created
4. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
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. Create a 'matlab_api'

1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Point the VSP CMAKE options to look for the swig.exe that you just created
4. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
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



Copy *.i files from python API example

Build Process – 4. Create a 'matlab_api'

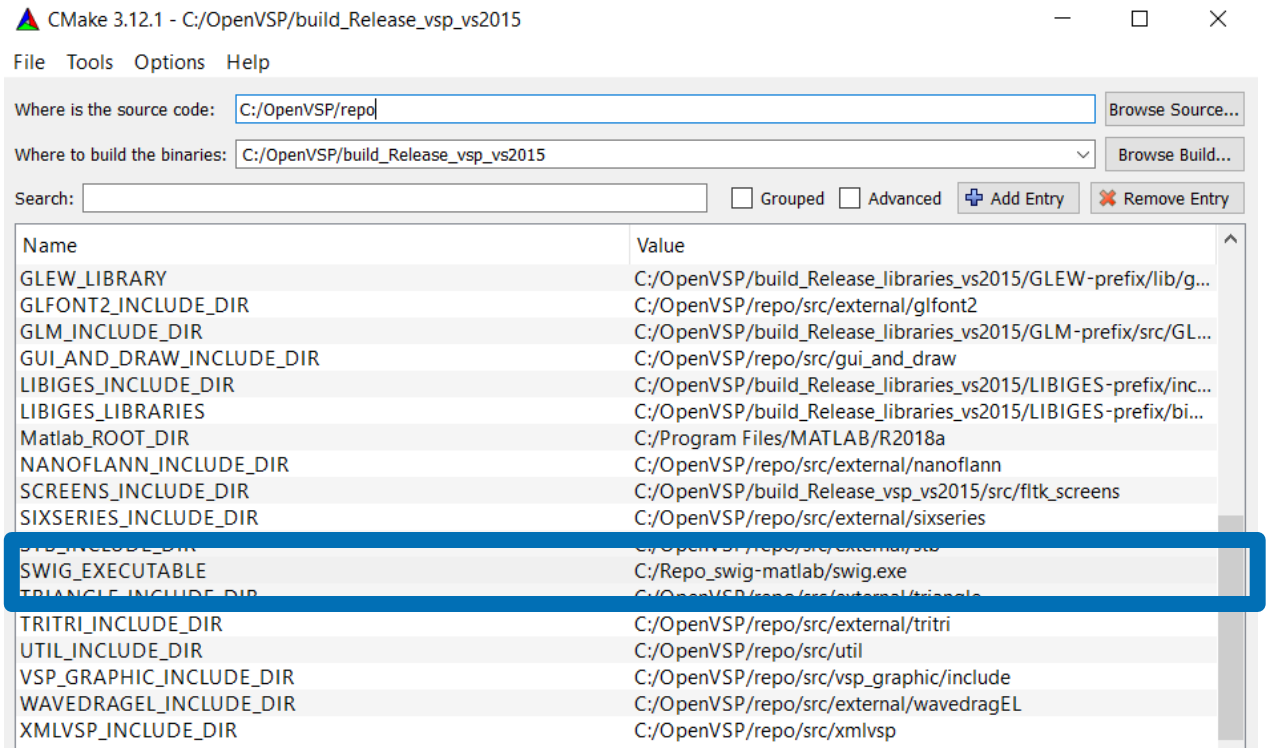
1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Point the VSP CMAKE options to look for the swig.exe that you just created
4. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
5. Compile OpenVSP (Build solution, Build INSTALL)
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. Point the VSP CMAKE options to look for the swig.exe

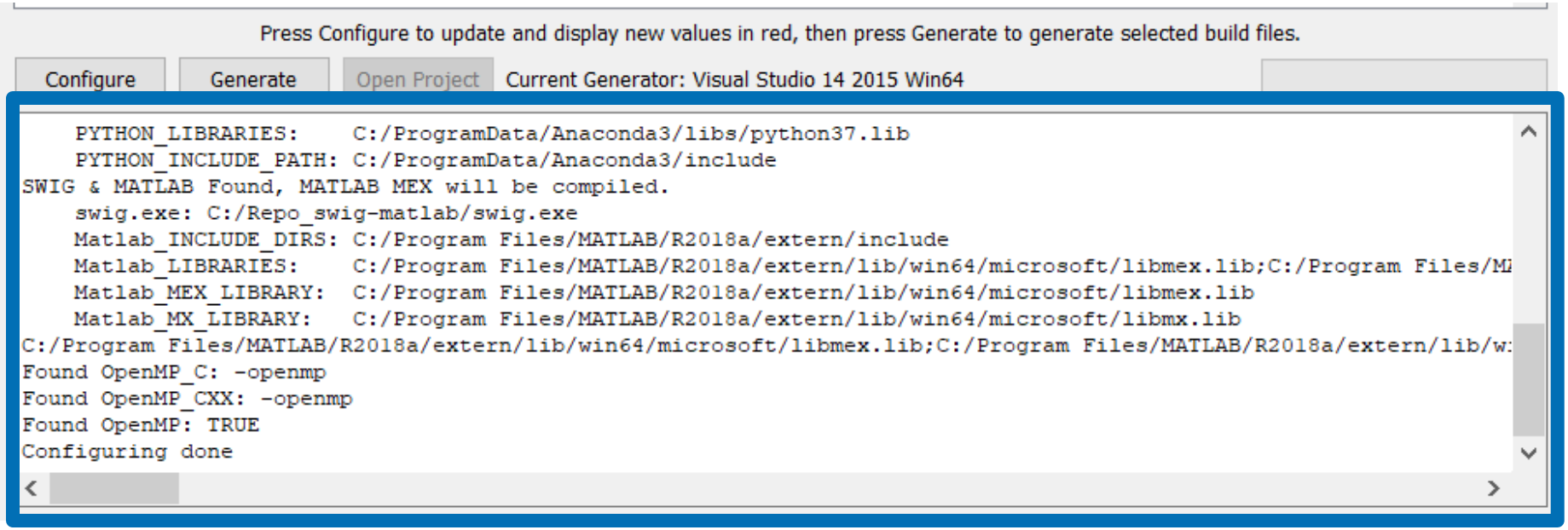
1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)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



Build Process – 4. Point the VSP CMAKE options to look for the swig.exe

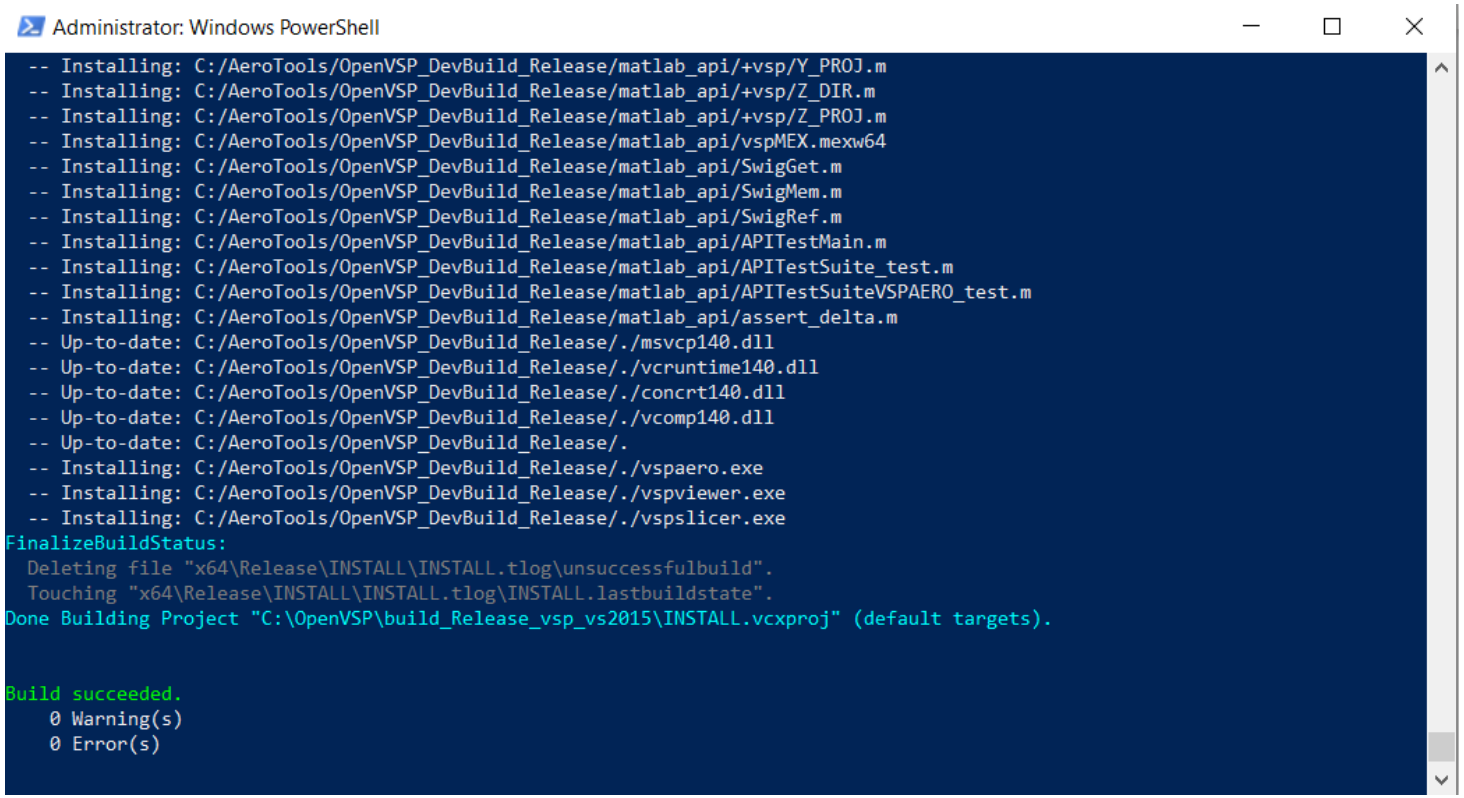
1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
4. Point the VSP CMAKE options to look for the swig.exe that you just created
5. Compile OpenVSP (Build solution, Build INSTALL)
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….



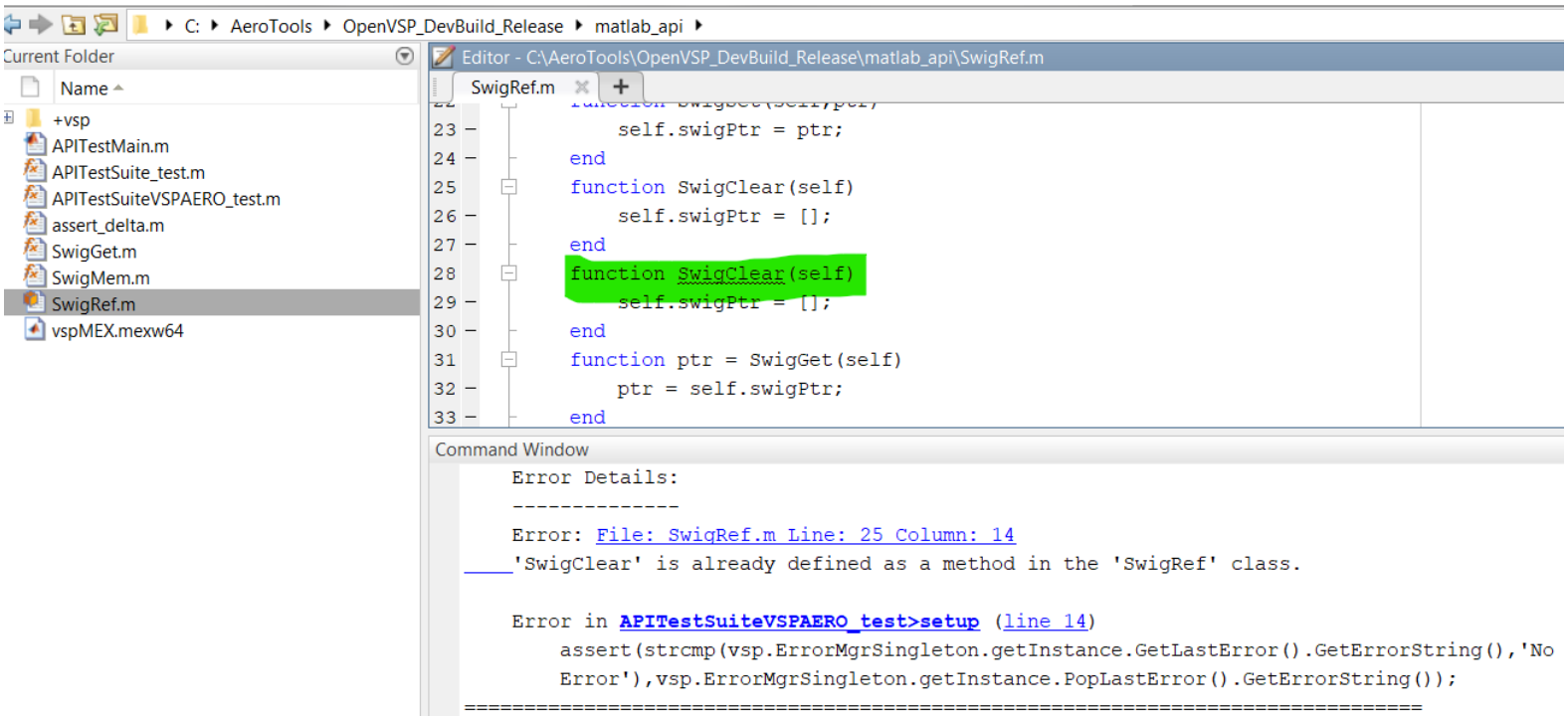
Build Process – 5. Compile OpenVSP

1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Point the VSP CMAKE options to look for the swig.exe that you just created
4. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
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 – 6. Delete duplicate SwigClear()

1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
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. Build some test scripts

1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
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

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

```
%APITestSuite
function tests
    tests = fu
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?

Contact Information

Nick Brake

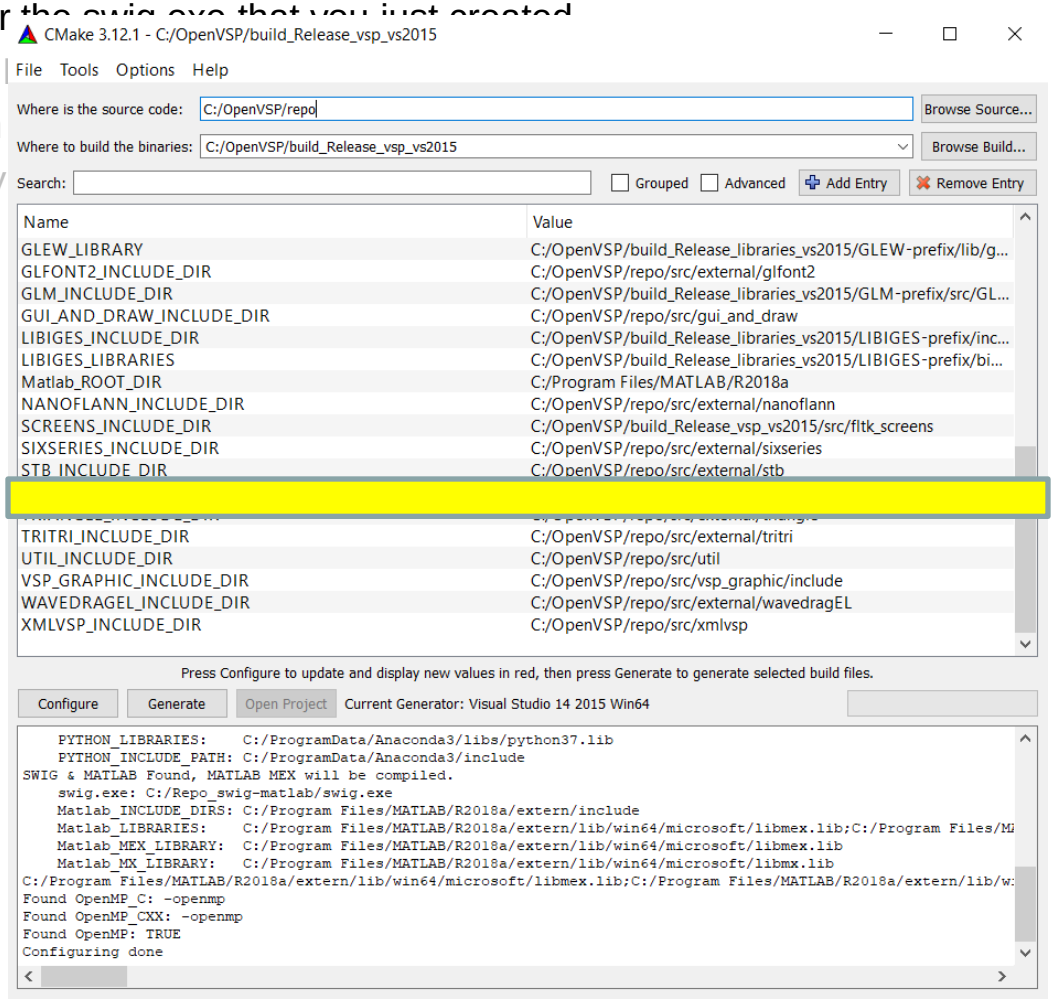
nick.brake@esaero.com

ESAero

openvsp@esaero.com

Build Process – 4. Point the VSP CMAKE options to look for the swig.exe

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



Documentation:
.\Repo_swig-matlab\Doc\Manual

Instructions exist to build using
- MinGW & MSYS
- Cygwin
- VisualStudio IDE (a bit manual)

Build Process – Creation of Interface files

1. Clone the swig-matlab repo
2. Compile swig-matlab from source and generate a swig.exe
3. Point the VSP CMAKE options to look for the swig.exe that you just created
4. Create a 'matlab_api' (use the 'python_api' as a template, this is the hardest step)
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

Documentation: