EMTG Windows Installation

The first part of this guide describes how to install a distributed EMTG executable on Windows for analysis tasks. Developers who also wish to setup the EMTG build environment should complete all of the steps in this guide (both sections) beginning with the first section.

Support

Contact Donald Ellison (donald.ellison@nasa.gov) or Jacob Englander (jacob.a.englander@nasa.gov) if you find any problems with this guide or the build system itself.

EMTG Installation Required Dependencies

NOTE: These dependencies are for installing EMTG only. If you would like to be able to build EMTG and attach a debugger the second part of this guide lists additional required dependencies.

- MS C++ redistributable C++ libraries
- MinGW-64
- Python 3.0
  - wxPython 4.0.1
  - SpiceyPy
  - JPLEphem
  - de423
- SNOPT 7.6
- Mission-specific CSPICE Kernels

EMTG version 9 **requires** C++17 support, therefore the 2017 MS C++ runtime libraries must be used.

MS C++ Redistributable Libraries

The redistributable libraries may be downloaded from this page.

MinGW-64

MinGW-64 is required in order to provide runtime Fortran libraries to SNOPT. These can also be provided using Intel Visual Fortran, however, MinGW is a free alternative.

Important: Make sure you download the 64 bit version of minGW, not the 32 bit version.

1. download the MinGW online installer from the project Sourceforge page.
2. run the installer
3. On the configuration screen select the x86_64 architecture, other settings can be left alone
4. Press Next to proceed to the installation location screen.

5. Specify an installation location (e.g. `C:\mingw-64`) **IMPORTANT:** Ensure that the MinGW installation path does *not* contain any spaces. If you want to build SNOPT from source, Visual Studio will not be able to find the MinGW runtime libraries if there is a space in the installation path.
6. Once MinGW has been installed, you must add its `bin` directory to your user and system environment variables. Go to **system->advanced settings->environment variables**.
    - For Windows 8+ use 'winkey+x' and select 'system'
    - For Windows 10 Creator's Update (i.e. May 2017 and onward) you'll ALSO have to then go to 'System Info'
    - For Windows 10, you can also just search for `environment` in the search bar and the utility will pop up
7. In the top half of the screen, under **User Variables** select **Path** and edit it.
8. Set the path to your minGW bin folder. For example: `C:\mingw-w64\x86_64-7.1.0-posix-seh-rt_v5-rev0\mingw64\bin` or `C:\mingw-w64\bin` depending on what installation path you chose.
9. Edit the system level Path variable with the same minGW path

Python 3.0

We recommend using the Anaconda Python distribution as it is well-supported and had comes fully configured with the conda package manager, which makes acquiring the astrodynamics sub-packages straightforward.

1. The latest distribution may be obtained from the Anaconda project download page.
2. Run the installer.
3. There is no need to add Anaconda to the path via the installer (it recommends that you don't). Instead, you can add it manually later on if you would like commands such as `conda`, `ipython` and `python` to be accessible from the Windows command line.
4. Assuming you install Anaconda to `C:\Anaconda3`, adding that directory and `C:\Anaconda3\Scripts` to your **User Variables Path** will give the command line access to those commands.
5. Open a Windows command prompt as an Administrator
6. Install wxPython: `conda install -c anaconda wxpython`
7. Install SpiceyPy: `pip install spiceypy`
8. Install JPLEphem: `pip install jplephem`
9. Install de423: `pip install de423`

SNOPT

SNOPT is commercial nonlinear programming software licensed by Stanford Business Solutions. The GSFC Flight Dynamics and Mission Design branch has licenses for SNOPT 7.2, 7.5 and 7.6 and a.i. solutions has a single-seat license for SNOPT 7.6 via the FDSS-II contract. If you are a student, the SNOPT developers offer a six month limited student license if you contact them with your .edu email address.

SNOPT Licensing page: https://ccom.ucsd.edu/~optimizers/licensing/ Stanford Business Software SNOPT product page: http://www.sbsi-sol-optimize.com/asp/sol_product_snopt.htm Stanford Business Software SNOPT purchase page: http://www.sbsi-sol-optimize.com/asp/sol_snopt.htm

Install EMTG

Contact donald.ellison@nasa.gov or jacob.a.englander@nasa.gov to obtain an EMTGv9.exe executable file.

1. Place the executable in the EMTG bin directory: `EMTG\bin\EMTGv9.exe
2. Place the SNOPT shared library in the EMTG bin directory: `.dll` in `EMTG\bin\libsnopt.dll`
3. Configure `EMTG\PyEMTG\PyEMTG.options` appropriately.

   i) Make a copy of `EMTG\PyEMTG\PyEMTG-template.options` and name it `EMTG\PyEMTG\PyEMTG.options` ii) `EMTG_path` must be set to the location of the `EMTGv9.exe` file iii) `default_universe_path` is the EMTG Universe parent directory (i.e. not `EMTG\Universe\ephemeris_files`)
4. SPICE ephemeris kernels must be placed in `EMTG\Universe\ephemeris_files`.
5. A desktop shortcut can be created for the PyEMTG GUI as follows:
   1. Create a generic Windows shortcut
   2. Set the **Target** field to:

      ```
      "C:\Program Files\Anaconda3\pythonw.exe"
      C:\Users\donald.ellison\NASA\EMTG\PyEMTG\PyEMTG.py
      ```

      Note the double quotes. The line break in the above command is a space between the two paths.
   3. Set the **Start in** to:

      ```
      C:\Users\donald.ellison\NASA\EMTG\PyEMTG
      ```

EMTG Windows Build System

The core EMTG code base is written in C++. The GUI is Python. If you have read access to the EMTG Git repository, the source may be obtained as from a git-enabled command prompt:

`git clone nasaid@128.183.251.48:/archive/emtg.git`

There are two configuration files that customize EMTG for operation on your personal machine, one for EMTG proper, and one for PyEMTG.

EMTG Build System Additional Required Dependencies

NOTE: The dependencies listed below are those required in addition to the dependencies listed in the first part of this guide.

- MS Visual Studio 2017
- CMake 3.12.0
- Boost 1.66.0
- GNU Scientific Library 2.4.0
- CSPICE N0066
- SNOPT 7.6

EMTG version 9 **requires** C++17 support, therefore, MSVS 2015 and older are not sufficient.

Before proceeding with the rest of this setup, please ensure that you have completed the MinGW and Python installation instructions above.

CMake

The latest stable CMake Windows installer can be obtained here.

IMPORTANT: If CMake was installed *before* MinGW, then the CMake installation repair will need to be run in order for CMake to recognize the MinGW installation.

Boost

The Boost C++ libraries come with their own build system called bjam.

1. Open a MS Visual Studio developer command prompt and navigate to the Boost root directory.
2. Run **bootstrap.bat** in the Boost root directory.
3. Edit your **project-config.jam** (typically located in your root boost directory). This file will only appear after bootstrapping Boost. Specifying the location of MSVC is not necessary, but sometimes the different Boost and MS Visual studio development cycles cause problems, so including it is more robust.

   ```
   import option ;
   using msvc : 14.0 : "C:\Program Files (x86)\Microsoft Visual
   Studio\2017\Community\VC\Tools\MSVC\14.10.25017\bin\HostX64\x64\cl.exe";
   option.set keep-going : false ;
   ```

4. Build Boost, exploiting multiple cores if you have them (`-j` option):

   ```
   shell> .\b2 -j4 toolset=msvc-14.0 address-model=64
   ```

5. The Boost libraries will be built to `Boost\stage\lib`

GNU GSL

The AMPL development team has created a CMake build system for the GNU Scientific Library. Their distribution can be found on Github here.

1. Download GSL, and create a build directory (e.g. `GSL\build`)
2. Use CMake to generate an MS Visual Studio Solution for GSL
3. Build GSL using MSVS

CSPICE

The download page for the C implementation of the CSPICE tookit is here. The CSPICE toolkit download typically comes pre-built, but building it again is performed by navigating to the CSPICE root directory and running the `makeall` script:

```
shell> .\makeall.bat
```

The generic CSPICE kernels can be found here. Any downloaded CSPICE kernels should be placed in `EMTG\Universe\ephemeris_files`.

EMTG requires that a leap second kernel file (`.tls`) and a frame kernel file (`.pck`) be present in the `ephemeris_files` directory as well as SPICE objects for any planetary bodies included in the optimization problem.

SNOPT

SNOPT is distributed with a pre-built MS Visual Studio Solution that assumes the use of Intel Visual Fortran. In order to use MinGW, a CMake build system has been developed for GTOL use.

1. Create a build directory for SNOPT (e.g. `SNOPT\build`) and use CMake to generate an MSVS Solution for SNOPT
2. The CMake system will generate `libsnopt.dll` in the build directory (e.g. `SNOPT\build\Release`) that must be re-located to the same directory as the `EMTGv9.exe` executable. The EMTG CMake system performs this action automatically.

EMTG

The core EMTG codebase is built using CMake. Before doing this, however, a local copy of the file `EMTG-Config-template.cmake` must be created and named `EMTG-Config.cmake`. This new copy must be customized based on your own directory structure.

The following variables must be specified in `EMTG-Config.cmake`

```
CSPICE_DIR
SNOPT_ROOT_DIR
BOOST_ROOT
GSL_PATH
```

The variable `EMTG_MPI` should be set to **OFF**, unless you wish to configure Microsoft MPI on your workstation.

1. In order to generate MS Visual Studio Project and Solution files, open CMake and set the source directory to the root EMTG directory. Set the build directory to `EMTG\build`.
2. Press **Configure** and select the appropriate compiler settings.
3. After the configuration has completed, press **Generate** to create the Visual Studio files.
4. Open the MSVS Solution, select either **Debug** or **Release** mode and build the Solution. The EMTG CMake system will perform a multicore build automatically, so expect high CPU usage during a build.

SSH Keys

In order to streamline interfacing with the EMTG repo., it is recommended that a developer install a public ssh key on NavLab 101.

1. Generate a private/public key pair using your favorite keygen tool. Git bash has access to **ssh-keygen** or many Git clients like GitKraken have a built-in method for doing so.

2. Once the key pair has been generated, then one can export their ssh identity to NavLab 101. For example, if you are Donald:

   ```
   shell> ssh-copy-id delliso2@128.183.251.48
   ```