GMAT CMake Build System

CMake on Windows, Linux, and MacOS X Operating Systems

Overview

Before starting, make sure that you have cloned GMAT from the GSFC-internal repository. For public developers, you can obtain the source code from SourceForge (The Git command is: "git clone ssh://YOURUSERNAME@git.code.sf.net/p/gmat/git gmat-git")

These instructions refer to <GMAT> as the top-level GMAT repository folder.

Figure 1. The <GMAT> repository layout

.

The GMAT build process can be broken down into four main steps:

1. Download and configure the dependencies (e.g. CSPICE, wxWidgets, etc.)
2. Create the build system using CMake
3. Build GMAT. Optionally, you can also install GMAT after building.
4. Run GMAT

The first two steps are generally "one-time" processes that are performed immediately after downloading the GMAT repository. They result in a build system (e.g. Visual Studio solution or makefiles) that will intelligently rebuild GMAT components as needed when source or configuration files are changed.

Step 1: Download and Configure GMAT Dependencies

Table 1 describes all software dependencies for GMAT. The <GMAT>/depends folder contains scripts to automatically download and configure the core GMAT dependencies.

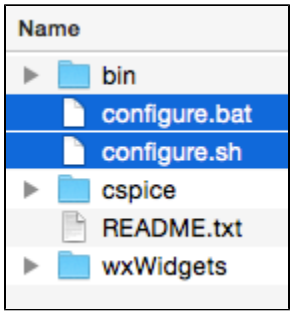
Table 1. GMAT Dependencies

Name	Version	Used in GMAT	Download
CSPICE	N0065	Core Dependency	Configure Script
wxWidgets	3.0.2	Core Dependency	Configure Script
Xerces	3.1.4	Core Dependency	Configure Script
MATLAB	R2015a+	CInterface Plugin MatlabInterface Plugin	Self-download
Python	3.4, 3.5	PythonInterface Plugin	Mac, Windows Linux (Package Manager)

Requirements:

- Windows: Visual Studio 2013 (Express or Paid versions).
- Mac: XCode tools (preferably v7+), with the command line developer tools (GCC 4.8.5 or greater)
- Linux: GCC compiler tools (GCC 4.8.5 or greater)

Figure 2. <GMAT>/depends folder layout after dependency configuration



Run the `configure.sh` (Mac/Linux) or `configure.bat` (Windows) script to set up core GMAT dependencies.

- Run configure.sh on Mac/Linux by name from the Terminal. Run configure.bat on Windows by double-clicking or running it from the Command Prompt.
- On Windows you will be prompted to select 32-vs-64 bit dependencies and a VisualStudio version. Choose according to your VisualStudio installation. On Mac/Linux these choices are currently auto-selected based on the system architecture.
- The resulting <GMAT>/depends folder structure should look like Figure 2.

Download optional GMAT dependencies.

- See Table 1 for dependency download links.
- For Python on Windows, select the option to add to system PATH. The equivalent option on Mac (Shell Profile Updater) is selected by default.

Step 2: Create GMAT Build System Using CMake

Requirements (in addition to Step 1 requirements):

- All Operating Systems: CMake (Minimum version 3.5.2)
- Linux: libgl1-mesa-dev, libglu1-mesa-dev, libgtk-3-dev (or libgtk2.0-dev), tcsh (or csh)
- Optional: MATLAB (if building MatlabInterface or CInterface plugins). On MacOS, the following must be done in Matlab once to initialize its internal compiler:
  - Launch Matlab, run the command: `mex -setup`
  - If Matlab gives a "No supported compiler or SDK was found" error, then a patch to Matlab must be applied. This is a known incompatibility between Matlab R2015a/b and XCode 7. The patch and instructions to apply it can be found here on the MathWorks website. This problem with Matlab is documented on the GMAT Bugtracker Issue 5634.
- Optional: Python

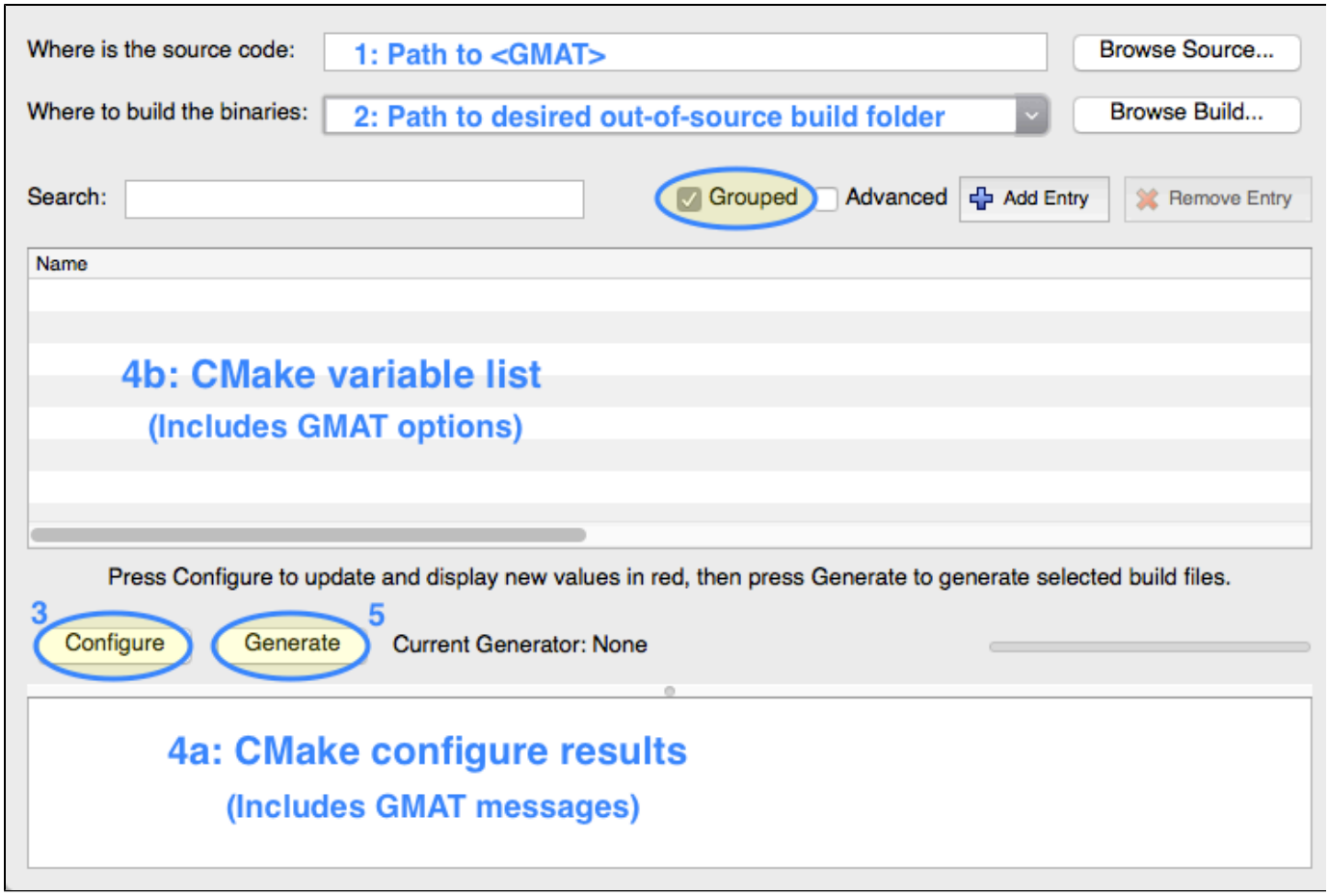
Launch the CMake GUI:

Tip: Select the "Grouped" option in the CMake GUI (Figure 3) to sort CMake variables and make them easier to find.

Tip: Select the "Advanced" option in the CMake GUI (Figure 3) to display variables, such as PYTHON_LIBRARY, that are normally hidden.

Tip: All CMake commands can also be performed on the command-line instead of using the GUI. See below for instructions.

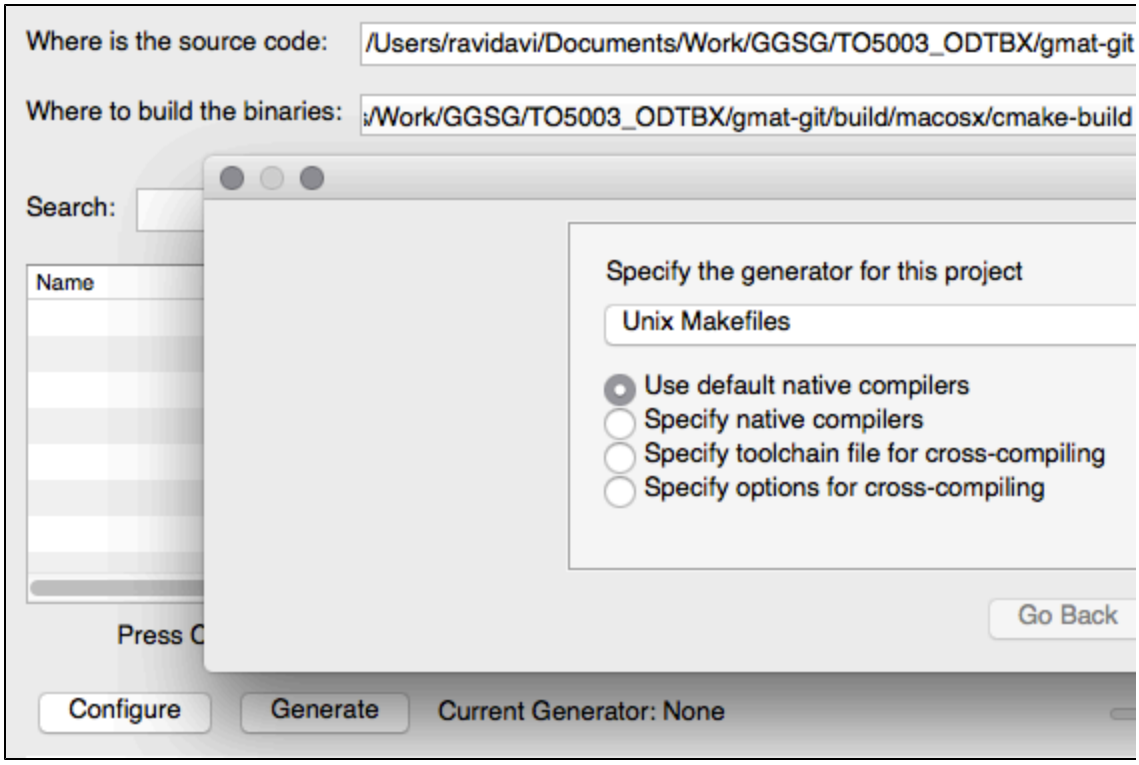
Figure 3. Components of the CMake GUI



Enter values into the CMake GUI in the following order (as annotated in Figure 3):

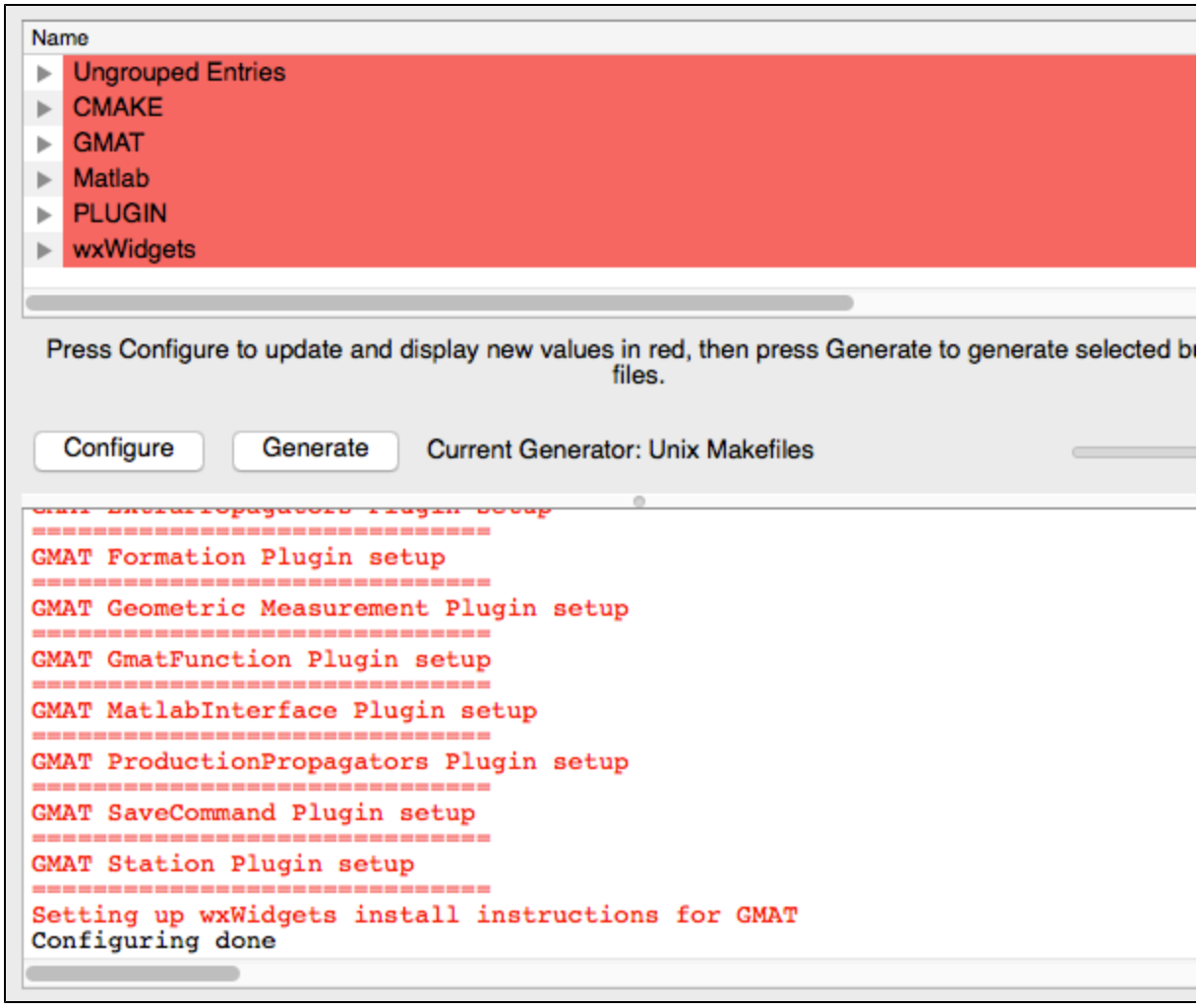
1. Enter the full path to the `<GMAT>` cloned repository on your computer
2. Enter the full path to the folder where the GMAT build system should be placed
    a. CMake produces out-of-source builds. It is recommended to use `<GMAT>/build/<OS>-cmakebuild` for this value.
3. Click "Configure"
    a. CMake may ask for permission to create the folder you specified in Step 2 ("Where to build the binaries")
    b. CMake will ask you to choose a generator (see Figure 4). See the CMake Generator webpage for an explanation of available generators.
    Recommended generators are:
        i. Mac: Unix Makefiles (although XCode has also been observed to work)
        ii. Linux: Unix Makefiles
        iii. Windows: Visual Studio (note e.g. "Visual Studio 12 2013" is 32-bit, whereas "Visual Studio 12 2013 x64" is 64-bit)

Figure 4. Choosing a generator in CMake



c. CMake will start the configuration process, after which the Variable List and Configure Results sections (Figure 3 sections 4a and 4b) will be populated:

Figure 5. Output of CMake Configure



4. Use the Configure results output box (Figure 3 section 4a) to change variables in the CMake variable list (Figure 3 section 4b) as follows:
   a. Always start at the **top** of the configure results output box and correct errors one-by-one
   b. For each error: change the appropriate CMake variable, re-configure, confirm the error was fixed (in the results output box), then repeat for next error
   c. In addition to errors, there are several CMake variables that allow you to control how the build system configures GMAT:

CMake Variable (Group)	Description	Associated CMake Error
`CSPICE_DIR` (Ungrouped)	Path to CSPICE root directory (containing `include/`, `lib/`, etc.)	CSPICE NOT FOUND (make sure to run depends script from Step 1)
`F2C_DIR` (Ungrouped)	Path to F2C root directory (containing f2c.h) Note: this should generally be `CSPICE_DIR/include`	F2C NOT FOUND (make sure to run depends script from Step 1)
`CMAKE_BUILD_TYPE` (CMAKE)	On makefile systems, this specifies the desired build type On VisualStudio/XCode systems, this specifies all possible build types Valid values: `Release`, `Debug`, `RelWithDebInfo`, `MinSizeRel` Note: On makefile systems, you should create a separate out-of-source build folder for each desired build type (Figure 3 box 2). (e.g. `cmakebuild-release` or `cmakebuild-debug`)	N/A

CMAKE_INSTALL_PREFIX (CMAKE)	Location to install GMAT when doing `make install` or building the VisualStudio->`INSTALL` project Note: This is autofilled to `<GMAT>/GMAT-<release>-<OS>-<BitType>` for convenience.	N/A
`GMAT_PROPRIETARYPLUGINS_PATH` (GMAT)	Full path to the top-level GMAT Proprietary Plugins `code/` folder. (folder that contains CMakeLists.txt) This will be automatically found if you name it `gmatinternal-git` and place it next to the main `<GMAT>` repository folder.	N/A
`Matlab_ROOT_DIR` (Matlab)	Path to MATLAB root directory (on Mac, this is the path to MATLAB_R20xxx.app)	Matlab NOT FOUND (make sure MATLAB is installed)
`PLUGIN_XXX` (PLUGIN)	Whether to build a particular GMAT Plugin Note: the proprietary plugins only show up here if `GMAT_PROPRIETARY_PLUGINS` has been correctly set	N/A
`wxWidgets_ROOT_DIR` (wxWidgets)	Mac/Linux: Path to wxWidgets `wx-config` utility (usually this is the wxWidgets `bin/` folder) Windows: Path to wxWidgets (containing `include/` and `lib/`)	wxWidgets NOT FOUND (make sure to run depends script from Step 1)
PYTHON_LIBRARY (PYTHON) (Advanced Variable)	Use a specific installation of Python for GMAT's PythonInterface plugin. If this variable is blank, the latest version of Python that can be found in default locations will be used. Set this to the FULL PATH to pythonXX.lib (e.g. C:/path/to/python35.lib) if you installed Python to a custom location.	Python NOT FOUND (make sure Python is installed, and variable is set properly)
XercesC_INCLUDE_DIR XercesC_LIBRARY (Xerces)	Xerces library and include folder locations.	Failed to find XercesC (make sure to run depends script from Step 1)

5. When all CMake errors are handled and you have specified all desired GMAT options, click "Generate".
   CMake will create the build system in the chosen out-of-source build folder (Figure 3 box 2).

Using CMake Command Line instead CMake GUI

CMake is fully scriptable and can be called from the command line instead of using the GUI. This is especially useful on operating systems (e.g. Red Hat Linux 7) where the GUI is unavailable.

1. Create the folder where the GMAT build system should be placed
   - `cd <GMAT>/build; mkdir macosx-cmake; cd macosx-cmake`
2. Tell CMake to run from the base `<GMAT>` folder and specify options
   - `cmake [options] ../..`
   - All options from above can be specified using `-DOPTION=VALUE` formatting. e.g. `-DPLUGIN_CINTERFACE=OFF`. Multiple such options can be specified.
   - CMake will perform both "Configure" and "Generate" steps at once, and place build system files in the current directory.

Step 3: Build and Install GMAT

Go to the build system folder (chosen in Figure 3 box 2), and follow the OS-specific instructions below.

Windows (Visual Studio)

1. Open the `GMAT.sln` Visual Studio solution. After loading, you should see the following projects:

Figure 5. CMake-generated VisualStudio2013 Solution

NOTE: This list may vary according to the GMAT plugins you chose in Step 2.
The common projects you will see are:
- `ALL_BUILD`: The default startup project. Ensures that all other projects are up-to-date, then builds them.
- `ZERO_CHECK`: Performs the work to ensure all other projects are up-to-date. It is automatically built along with all other projects.
- `INSTALL`: Creates a standalone GMAT folder containing all executables, plugins, data files, samples, and documentation.
- `GmatBase`, `GmatConsole`, `GmatGUI`, Plugins: The various GMAT components. You can build these individually if desired.

2. Choose a build configuration, e.g. `Release`, `Debug`, etc.
3. Build the `ALL_BUILD` project. Depending on your system speed and number of selected GMAT components, this may take a while!
4. (Optional) Build the `INSTALL` project if you want a fully standalone and relocatable version of GMAT.

MacOS and Linux (makefiles)

Makefiles are run through the command line, which on Mac and Linux can be accessed via the Terminal application.

In these instructions, `<CMake_build_path>` is the path to the build system folder that you chose in Step 2 (Figure 3 box 2).

1. Open a command prompt and type the following commands (assuming $ is your command prompt):
   ```
   $ cd <CMake_build_path>
   $ make
   ```
   - Note: if you know your computer has `N` cores, you can also do "`make -jN`" to significantly speed up the compile time
2. (Optional) Type "`make install`" if you want a fully standalone and distributable version of GMAT.

Possible Build Errors

See the following table in case there are build errors when compiling or installing GMAT.

Error	GMAT Component	Description	Fix
`Unresolved External Symbol *_Py_* referenced in function ...`	Python Interface	32/64-bit Python found by CMake is different than architecture of compiler (VisualStudio, gcc, ...)	Make sure to install the correct 32- or 64-bit version of Python and specify it via the PYTHON_LIBRARY CMake variable.
CInterface Matlab thunk files not produced during INSTALL step	C Interface	There is a known incompatibility between Matlab R2015a/b and XCode 7 that prevents the CInterface thunk files from being built.	Perform the Matlab initialization instructions.

Step 4: Run GMAT

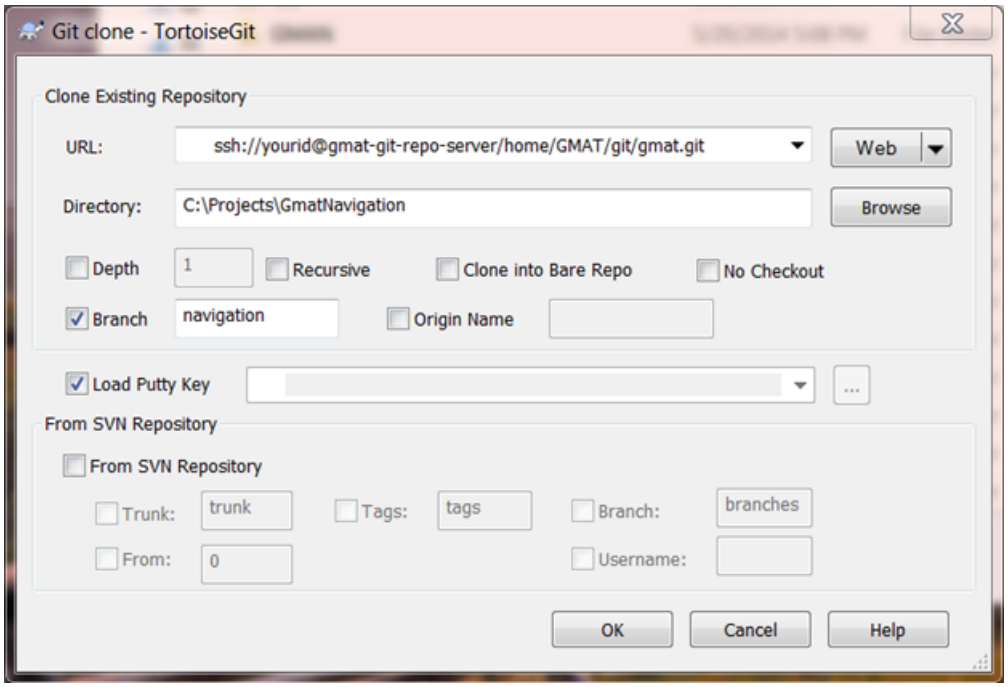
After building, you have several options for how to run GMAT:

- **GMAT Users:** After building the INSTALL project in Step 3, a standalone copy of GMAT is placed in the `CMAKE_INSTALL_PREFIX` folder that you chose in Step 2. The GMAT executables will be in the `bin/` subfolder.
- **GMAT Developers:** GMAT executables and plugins are placed in the `<GMAT>/application` directory. You can run GMAT from the `bin/` (or `debug/`) subfolders without having to perform the optional INSTALL step. This allows for a more rapid edit-build-test development cycle. On Windows, you can also run the `GmatConsole` and `GmatGUI` projects directly from within VisualStudio. This allows for in-program debugging with breakpoints.
  1. Select the configuration that you want to run (`Release`, `Debug`, etc.)
  2. Right-click on the `GmatConsole` or `GmatGUI` project, and select "Set as Startup Project"
  3. Select menu item Debug -> Start Debugging (or Start Without Debugging for Release configurations)

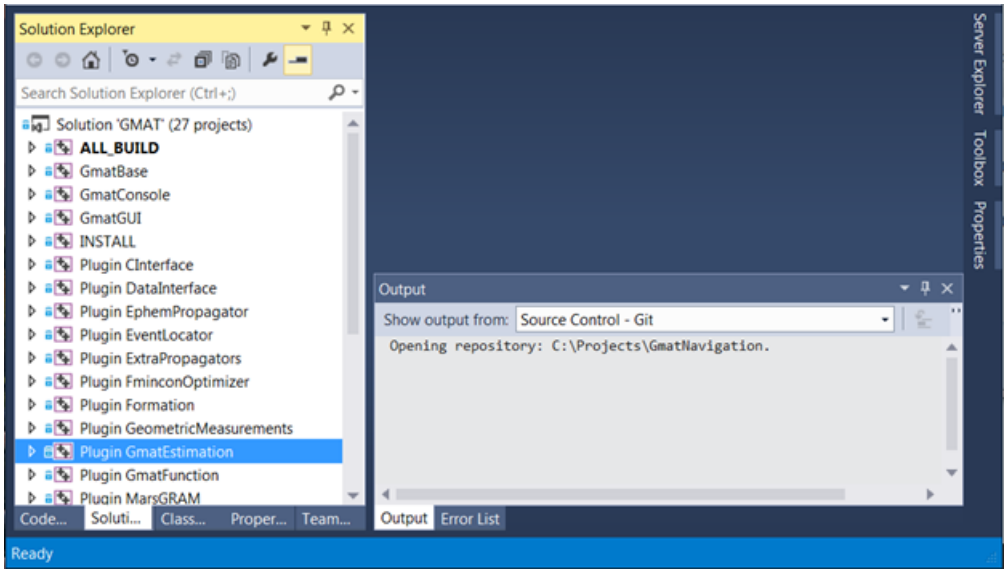
~~Building GMAT Navigation branch using CMake on Windows~~

This is no longer needed, since Xerces has been added to the configuration scripts and CMake setup.
This section can be removed.

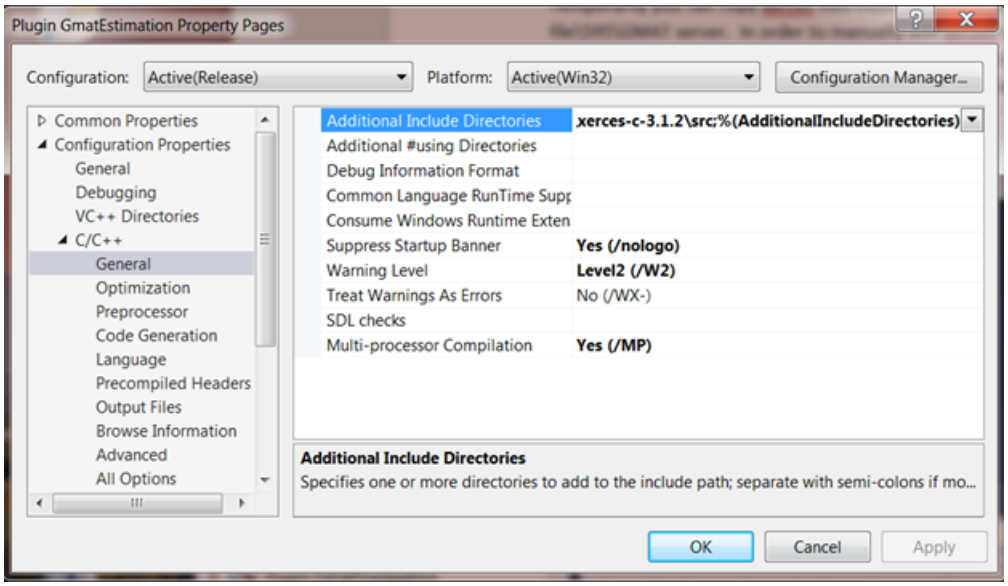
1. Git clone GMAT Navigation branch. The following example dialog shows the GMAT git repository URL and Directory where Navigation branch will be cloned. Use the correct URL and your id in the URL field. Note that Branch checkbox is checked and branch name is “navigation.” Click “OK” to start cloning the repository.



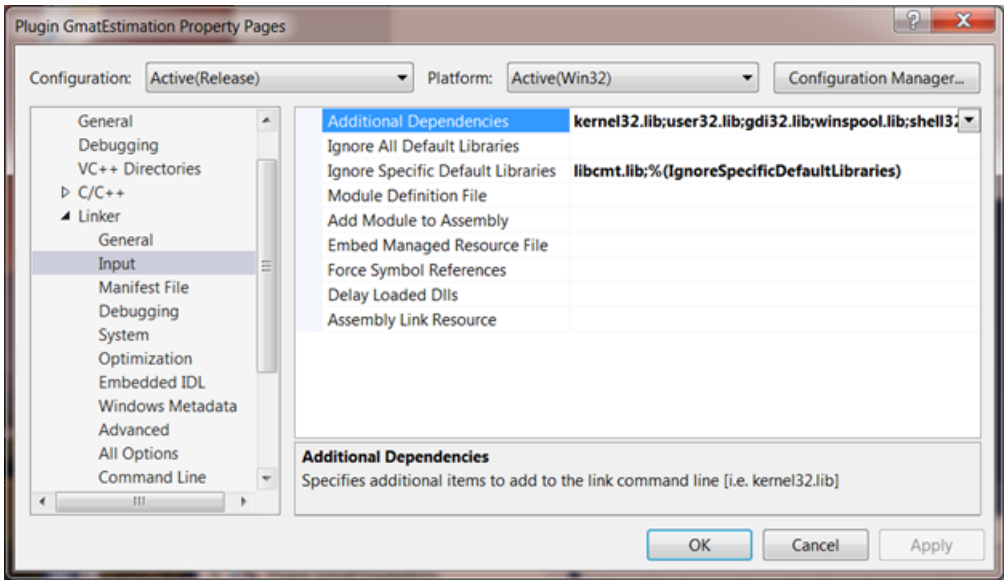
1. Follow Steps 1 and 2 for configuring and generating VS 2013 solution using CMake from the wiki link http://gmatcentral.org/display/GW/GMAT+CMake+Build+System
2. Navigation build needs additional library called “xerces” (https://xerces.apache.org/xerces-c/). At this time, the XML Parser (Xerces) is not automatically downloaded and configured in Step 2 above (it will be added in a future). Temporarily you can copy “xerces” files from the \Builds\windows\xerces-c-3.1.2 on mesa-file server. The “xerces” libraries on mesa-file were built using VS 2013. If you have other version of VS C++, you will need to build “xerces” libraries on your own and configure CMake to build GMAT using the same compiler. In order to manually add “xerces” include and library to GmatEstimation project, first open GMAT.sln in where it was configured and generated from the CMake. The example solution was generated in C:\Projects\GmatNavigation\build\windows-VS2013-CMake-32. The following shows when GMAT.sln is opened.

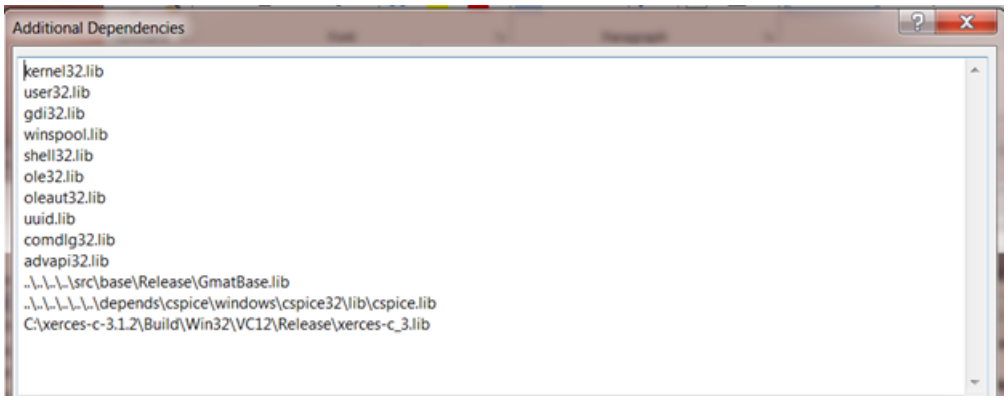


1. First, add “xerces” include file to “Plugin GmatEstimation” project. Right click on “Plugin GmatEstimation” and select “Properties.” Expand “C/C++” and select “General.” In “Additional Include Directories” field, add “xerces” source directory where you coped into. The example shows “C:\xerces-c-3.1.2\src” was added at the end. You can also add it by clicking the down error symbol and “<Edit…>.”

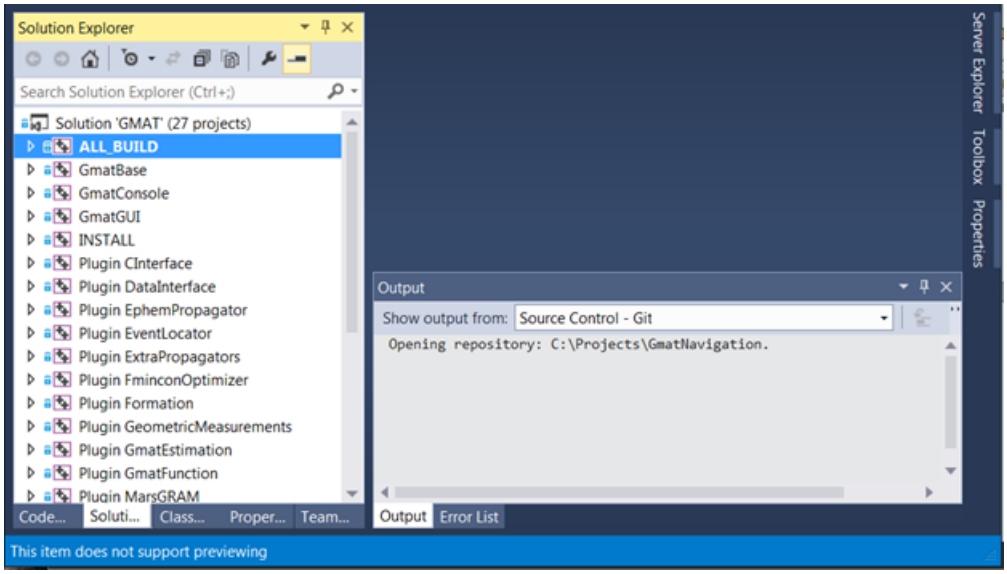


1. Next, add “xerces” library to “Plugin GmatEstimation” project. While “Plugin GmatEstimation Properties Pages” is still open, expand “Linker” and select “Input.” Add “YourXercesDir\Build\Win32\VC12\Release\xerces-c_3.lib” to “Additional Dependencies” field. The example shows C:\xerces-c-3.1.2\Build\Win32\VC12\Release\xerces-c_3.lib was added at the end.

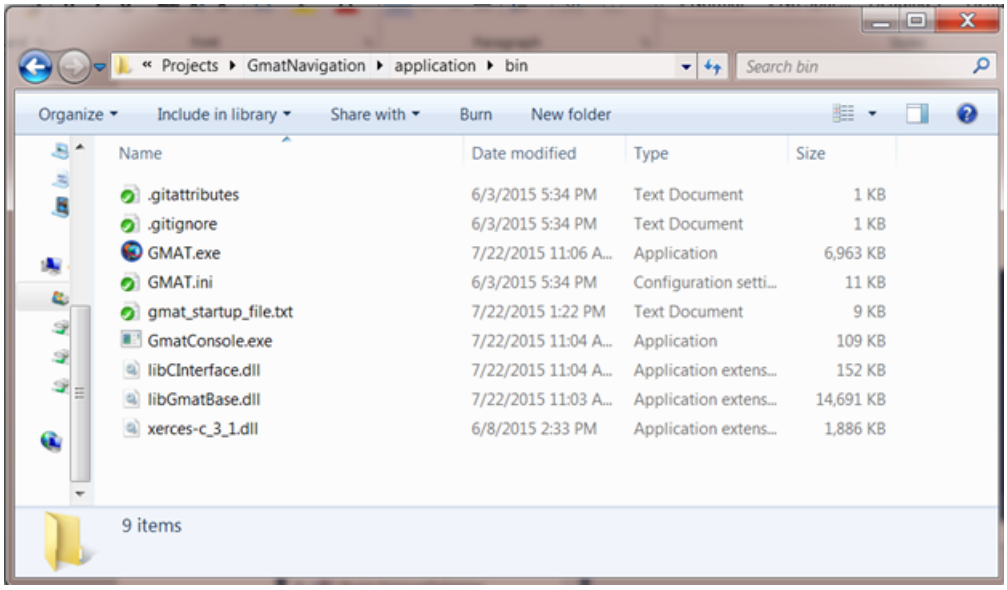


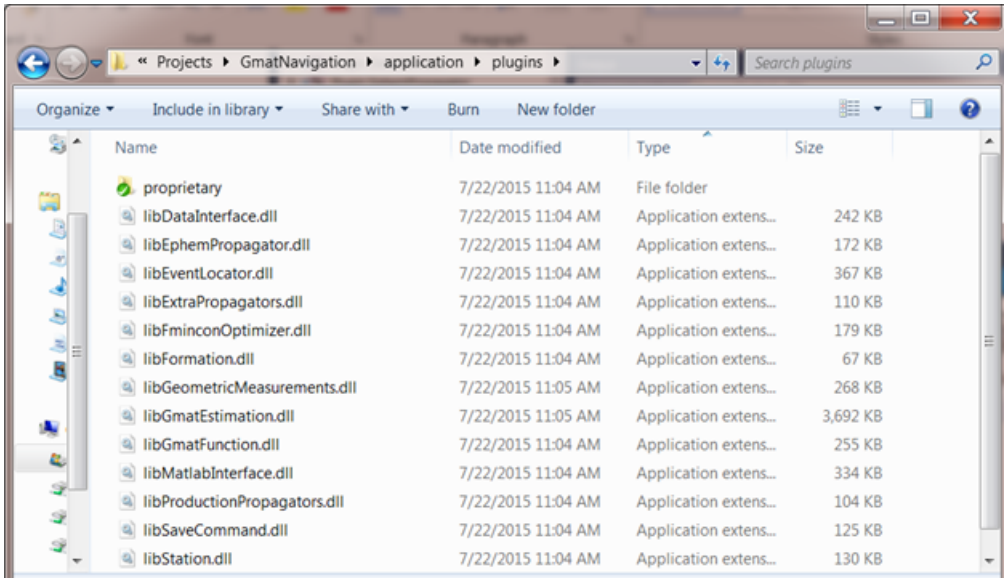


1. Next, build the solution. Right click on “ALL_BUILD” and select “Rebuild.”



1. When build has finished, the GMAT executable and base dynamic libraries (DLLS) are located in /application/bin directory. Other Plugin DLLS are located in application/plugins.





1. Before running GMAT, copy “xerces-c_3_1.dll” from “YourXercesDir\Build\Win32\VC12\Release” to /application/bin directory. Now you can double click on GMAT.exe to launch GMAT application.

Know Issues and Work-Arounds

wxWidgets on Mac

wxWidgets v3.0.2 has a known bug (documented here) on Mac OSX 10.10+ that causes a build error. As of R2016a, the GMAT dependency configuration script (`configure.sh`) implements this fix internally, so GMAT users do not need to take any additional action for wxWidgets to build on Mac.