EMsoft: General User and Development Manual

Program Manual, Release 3.1, March 25, 2017

Table of Contents

1 What is EMsoft?

EMsoft is an open source package for the simulation of electron microscopy imaging and diffraction modalities. It consists of a library with core routines for crystallography, symmetry, dynamical scattering, Monte Carlo simulations, and so on, plus a series of programs for different imaging and diffraction modalities. The package is based on the older, and now discontinued, CTEMsoft package, for which the version 2.0 source code is still available at *http://www.github.com/marcdegraef/CTEMsoft*. This code base is no longer supported and is replaced by a new source code base, version 3.1, available from *http://www.github.com/marcdegraef/EMsoftPublic*. The two versions are not compatible with each other. In the current release, we provide a series of additional support programs as well as several new dictionary-indexing programs.

This manual describes a few aspects of the EMsoft package:

- how to install the software developer toolkit;
- the different ways of communicating with the programs (input files);
- and a list of programs available in this release (3.1).

If you wish to compile EMsoft yourself, then please proceed with the next section; if you only plan on using the executables, please jump ahead to section 5.

2 System Requirements

For detailed system requirements and for instructions on how to install the Software Developer Kit (SDK), see the EMsoftSDK_Installation.md file in the top EMsoft folder. Certain EMsoft programs require a GPU (Graphical Processing Unit), which, in turn, requires proper installation of OpenCL (automatic on Mac OS X with XCode 7; needs driver installation on all other platforms). A standard graphics card in a laptop is likely insufficient to run the GPU-based programs.

3 The EMsoft Software Developer Kit

If you wish to compile the package code yourself, then you will first need to pull the code base from github; you can clone the repository from *http://www.github.com/marcdegraef/EMsoft*. Select a location on your hard drive to host the source code tree; we recommend that the foldername be EMsoft, but you are obviously free to change this.

Before you can start to compile the code, you must install the software developer kit (SDK). This is made very easy thanks to our friends at BlueQuartz.net; open a terminal window, navigate inside the EMsoft tree to the folder Support/SDK_Build_Scripts and then select the folder for your OS. In that folder you will find a file called SDK_Configuration.conf. Use a text editor to edit this file. The SDK must be built in its own folder which you can place anywhere on your system; here are a few logical locations for the SDK:

- if you are the only user on that computer, and you do not have root access, then it makes sense to place the SDK at the top level in your home folder.
- if you have root access, then perhaps an installation in the /opt folder (Linux and Mac) or in the /Users/Shared folder (Mac) would be appropriate.

Once you decide where to place the SDK, edit the SDK_PARENT field in the configuration file to set the appropriate top folder; we suggest that you do not change the SDK_FOLDER_NAME. The SDK_INSTALL variable should then combine the previous two variables. The remaining parameters in the first block of generic setup parameters should be left unchanged, with the exception of the PARALLEL_BUILD parameter which sets the number of cores to use for the SDK compilation.

The second block of parameters define the fortran-90 compiler that will be used to generate the SDK. Only gnu (gfortran) and intel (ifort) are allowed, and you should set the corresponding install parameter. You should also set the JSONFORTRAN_FC_VENDOR parameter further down in the file to the same gnu or intel value. All other parameters should be left unchanged. Save the modified configuration file. Make sure your shell init script (e.g., .cshrc for the C-shell on UNIX systems) has an entry to define the fortran compiler location, as in

```
setenv FC /opt/local/bin/gfortran
```

for the C-shell and a similar entry for other shells.

The build script should be executed with

```
(sudo) ./Build_SDK.sh
```

on unix-based systems and

```
???? ./Build_SDK.sh
```

on Windows.

This shell script will download the EMsoft_SDK archive files from one of the BlueQuartz web servers and compile them. The script will install the following program and libraries:

- CMake: CMake is a package that manages the entire compilation process. To make sure that the correct version is used, it is installed first.
- fftw3: the FFTW3 library for Fourier transforms [included as of Release 3.1]
- jsonfortran: a small library to parse and create json-formatted files (JavaScript Object Notation);
- clfortran: another small library with wrapper routines to call OpenCL from within f90 programs. Note that a previous version of EMsoft used the fortrancl library, which is now no longer needed.
- HDF5: a large library for Hierarchical Data Format support; note that this library will be compiled twice, once in debug mode and once in release mode. It is normal for there to be a large number of warnings during the compilation; just ignore them.

All five of these packages should compile and install without any issues, assuming that you have the correct compilers on your system. Note that the installation is sandboxed, i.e., it will not interfere with or overwrite any existing implementations of these libraries; you should only use the libraries inside the sandbox, though. The compilation process should only take about five minutes. Once the SDK has been compiled, you will be ready to compile EMsoft, as described in the next section.

4 Compiling EMsoft

Add the location of the cmake bin folder inside the SDK to your path variable; this will depend on your operating system and the type of shell you are using. For Mac OS X, the command is as follows (assuming the default SDK location):

```
export PATH=$PATH:/SDKfolder/cmake-#-#-#-Darwin-x86_64/CMake.app/Contents/bin/
```

For Linux, use

```
export PATH=$PATH:/SDKfolder/cmake-#-#-#-Linux-x86_64/bin/
```

Typically, you would add this line, or a similar one for other shells, to your .bashrc or .cshrc file. At the time of release 3.1, the cmake version number is 3.6.2.

In a terminal window, cd to the folder that contains your EMsoft top folder, create a EMsoftBuild folder, cd inside it and enter the following command:

```
cmake -DEMsoft_SDK=/SDKfolder -DCMAKE_BUILD_TYPE=Debug ../EMsoft
```

where /SDKfolder is the location of the SDK folder that you installed in the previous section. [note that you can also set the build type to Release; this will produce faster executing code, but error messages in case a program fails will not be as informative as for the Debug build type] This will spit out a number of variables and path names.[1] This is the first cmake run, which will set up all necessary variables for future runs. The next time you call the cmake program, you can omit the arguments (except for ../EMsoft); you can also use the interactive ccmake version. To compile the EMsoft package, simply type

```
make -j#
```

where # stands for the number of cores you want to use for the compilation and the entire library and all the executables will be compiled. From here on, you can start editing the source code and recompile things as needed. All executables will be located in the EMsoft/Bin folder (or EMsoftBuild/Bin if you have just set up the SDK), so you may want to add this folder to your path variable. Next, you will need to set up your configuration file; that process is described in the next section.

5 Installation Instructions

If you chose not to compile the code, and you only wish to execute the programs, or you did perform the steps in the last two sections, you will now need to set up your environment so that the executables will be found. The package file with executables can be downloaded from the following location:

```
http://muri.materials.cmu.edu/
```

In the older version of CTEMsoft, the programs required several environment variables to be set, so that all the resource files and such could be located. In the present release, this has been changed to a more flexible system that should work on all platforms (OS X, Linux and Windows). There are only a few variables that need to be set in order for the programs to function properly. They are defined via the EMsoftConfig.json file, located inside the hidden .config/EMsoft folder in the user's home directory.

[1]If you get any error messages or notice things that do not look right, please contact us with a screen dump as well as a description of your system, including the OS version.

1. If you are installing the executables, place the EMsoft.zip archive somewhere inside your home folder; for instance, if your username is “me,” you could create a folder named packages inside the /Users/me folder, and download the archive inside /Users/me/packages. Then unpack the archive, which will create an EMsoft folder containing several other folders. If you installed the SDK and source code, you may skip this step.

2. If you are using a standard installation as described in this document, you should be able to cd into the EMsoft/bin folder (EMsoftBuild/Bin if you installed the SDK) and execute the program EMsoftInit from the command line. This program makes some assumptions about where several folders are located and creates the configuration file for you with a number of default entries. If this program fails, it maybe because you are attempting a non-standard installation, so you can proceed manually and create the configuration file as described in the next paragraph; note that every EMsoft program will read this configuration file at the start of the execution. In addition to creating the configuration file, this program also creates an EMdata folder at the same level as your top EMsoft folder, as well as an XtalFolder inside it; this latter folder will be used to store crystal structure files and must be present.

   For a manual creation of the configuration file, first check whether or not the .config folder exists in your home folder; if it doesn’t exist, create it (don’t forget the period in front of the folder name). Then go into this folder and create the EMsoft folder; go into that folder and create a tmp folder. Then, at the same level as the tmp folder, use a text editor to create a file with the name EMsoftConfig.json and enter the following ten lines, including the curly braces:

```
{
	"EMsoftpathname" : "/full/path/to/the/top/EMsoft/installation/folder/",
	"EMdatapathname" : "/full/path/to/the/EMsoft/data/folder/",
	"EMtmppathname" : "/full/path/to/.config/EMsoft/tmp/",
	"EMsoftLibraryLocation" : "/full/path/to/the/library/folder/",
	"Release" : "No",
	"UserName" : "Your full name",
	"UserLocation" : "Your location or affiliation",
	"UserEmail" : "Your email address"
}
```

   If you installed the SDK, then you should also add the line ”Develop” : ”Yes”, to the configuration file. Note that the white space at the start of lines 2 through 8 is a tab. Obviously, you will need to change the paths above. Typically, the installation folder would be somewhere below your home folder, so on Mac OS X for instance, this could be /Users/me/packages/EMsoft/; note that the path must be terminated with a /. The second variable indicates where you will keep your data files, i.e., all the files generated by the EMsoft programs. This folder must be outside of the main EMsoft-tree, so that no data will be overwritten when future updates or releases become available. Create a top data folder, for instance /Users/me/Data/EMplay/. Then, inside this folder, create an additional folder with the name XtalFolder; this is where all the crystal structure descriptor files will be kept. Whenever the user specifies the location of a data file, this location is always a relative pathname with respect to the EMdatapathname location; so, if we have EMdatapathname set to /Users/me/Data/EMplay/, and we wish to create a data file with absolute pathname /Users/me/Data/EMplay/Silicon/EBSDmaster.h5, then the relative pathname will be Silicon/EBSDmaster.h5. This makes it possible to exchange data files between users or transfer files from one system to another. The variable Release must be set to Yes if you are using the Release version of this package (in other words, if you installed the executables). If instead you have installed the Software Developer Kit, and you are editing source code and compiling things yourself, then you should set this variable to No. Note that this variable, along with the EMsoftLibraryLocation variable, are only used by the IDL visualization interface. Finally, set the variables that start with User to the correct values; these variables will appear in all of the HDF5 files created by the EMsoft programs (added in Release 3.0.1).

3. Adjust the PATH variable so that your shell will find the executables in the EMsoft/bin (or EMsoftBuild/Bin) folder. You can either do this via your shell setup files (e.g., .cshrc or .bashrc), or via the system wide /etc/paths file (which will require sudo access). The syntax will depend a little on which shell you are using (for UNIX-based systems). Note that the UNIX flavor on Mac OS X is not case-sensitive for file and folder names; on all other UNIX-flavored platforms, filenames and folders *are* case-sensitive.

4. To view the HDF5 formatted data files, you may wish to install the free HDF5 viewer. First you need to have Java installed (if it isn't already): for Mac OS X, go to

   `https://java.com/en/download/faq/java_mac.xml`

   to install Java. Then download the HDF5 viewer app from

   `http://www.hdfgroup.org/products/java/release/download.html;`

   scroll down to Max OS X. The HDF5 files can also be read by any other program that supports this format, e.g., IDL, Matlab, etc...

6 The general structure of EMsoft programs

Each EMsoft program has the same high level source code structure, and is called in the same way. If EMprogram represents a program name (for instance EMECP or EMlacbed, then the following command line options can be used (all of them optional):

```
EMprogram [-h] [-t] [file.nml]
```

The arguments between square brackets are optional, and are defined as follows:

- -h: (h=help) this argument causes the program to display a list of all command line arguments and their meaning. The program will quit after printing the help message.
- -t: (t=template) this argument causes the program to create template files for all the input files used by the program. For each name-value entry in the template file (see section 7 for more details), a comment line is inserted with a brief explanation of the variable and its units, if appropriate. The user can then copy the template file to a new file with the namelist (.nml) extension and edit the file with a regular text editor. The program will quit after generating the template files. Each template file contains the default values for each parameter. If this option produces an error, then this is likely due to an incorrect setting of the shell variables in the previous section.
- file.nml: the main namelist file to be used by the program. If no name is present, then the default filename EMprogram.nml will be used. If the provided namelist file does not exist, the program will report an error and abort.

Note that the programs will also accept .json files instead of .nml files.

The programs in the top half of the table on the last page of this manual function in the way described above; all other programs have a simple command line interface, with user prompts for all relevant variables. Eventually, in a future release, all programs will make use of namelist or json input files and will likely be callable from a single user interface (i.e., DREAM.3D).

7 Program input files

All user input for the EMsoft programs is performed either via namelist files or via json files; the namelist is a special f90 IO format that allows one to list *name-value* pairs in an ordinary text file, and then this file will be read and interpreted by the program. The basic format of a namelist file is as follows:

```
&nmlname
var1 = val1
var2 = val2
...
/
```

The first line must begin with the ampersand character "&" and is followed (without spaces) by the internal name used by the program to identify the set of variables (this is similar to a *common block* in the older f77 standard). After the mandatory first line, which should not be altered, a list of name-value pairs follows. These can appear in any order, and variables may be omitted from the file, in which case the program will use a default value (this value can be found in the template file; see section 6). The final line of the file must contain a single forward slash character, "/", to end the namelist.

The json format (JavaScript Object Notation) is an xml-like version of the name-value pairs, and is very similar to the namelist format, with the exception of the presence of curly braces. For the example namelist file shown above, the equivalent json file would look as follows:

```
{
        "nmlname": {
                 "var1": val1,
                 "var2": val2,
                ...
        }
}
```

The EMsoft programs automatically detect which format of input file is used, so either one is fine. The json formatted input files will in a future release be generated by a number of filters in the DREAM.3D software package, whereas the namelist formatted files must be edited by hand, starting from the template files (created with option -t).

As a concrete example, consider the namelist file used to define a dislocation:[2]

```
&dislocationdata
id = 0.501
jd = 0.501
u = 1.0,0.0,1.0
bv = 0.5,0.0,0.5
/
```

[2]Note that this example will be superceded by a general json defect description file starting in release 3.2; The example is still useful for other nml files, though.

Internally, the program identifies this set of variables with the namelist identifier “dislocationdata,” as shown on the first line of the file. Then there are four name-value pairs: id and jd are real numbers in the range $[-1,1]$ and define the position of the defect; u represents the line direction, which is declared as a triplet of floating point numbers; and the burgers vector is represented by another triplet of floating point numbers and the variable name bv. The forward slash closes the file. Note that these entries can appear in any order. Comment lines are allowed in the namelist file; simply start the line with the f90 comment character (the exclamation mark “!”) and then type your comment. For variables that have components, such as the two vectors above, one may also define each component separately, in which case the file would look as follows:

```
&dislocationdata
id = 0.501
jd = 0.501
! these vectors are defined one component at a time
u(1) = 1.0
u(2) = 0.0
u(3) = 1.0
bv(1) = 0.5
bv(2) = 0.0
bv(3) = 0.5
/
```

It is not necessary to leave spaces before and after the “=” symbol, but it does improve the readability of the namelist file to do so. Since one can always recover the template file(s) by using the [-t] option when calling the program, there is no need to list all the name-value pairs in the namelist file; pairs which are not listed will take on their default value(s), which are the ones listed in the template file.

One important thing to note: when entering file names and other strings in name list files, one must use single quotes both at the start and at the end of the string. Some word processing programs have so-called “smart quotes” settings, which will introduce a different kind of quote, and the fortran programs will not recognize smart quotes as being the same as single quotes. Therefore, turn off the “smart quotes” option in your editor before you start editing namelist or json files.

8 Generating a crystal structure file

With very few exceptions, all programs in the EMsoft suite of programs require a crystallographic input file that defines the crystal system, crystal lattice parameters, space group number, and all unique positions in the asymmetric unit along with site occupations and Debye-Waller parameters. To keep things simple and portable across platforms, structure description files (*.xtal) are small HDF5-formatted files that contain the minimum amount of information needed to unambiguously define the complete unit cell.

To create a crystal structure description file, one uses the EMmkxtal program, which will prompt the user for the crystal system number, the lattice parameters (in nm and degrees), the space group number, and the atom positions in the asymmetric unit. Each atom entry is defined by five numbers: the fractional coordinates (which can be entered as real numbers or as fraction, e.g., 0.5 or 1/2), the site occupation parameter (a real number between 0 and 1), and the isotropic Debye-Waller factor in nm^2.

Once all atoms in the asymmetric unit have been entered, the program will ask for a structure file name; we recommend that all such files have the extension .xtal, to make them easily recognizable. All .xtal files will be automatically placed in the XtalFolder inside the main EMsoft data folder.

Starting with release 3.1, the EMmkxtal program can be called with the -w option, which changes the entering of atom coordinates to the use of the Wyckoff position symbols. After the space group number has been entered, the program will list the periodic table as well as the list of available Wyckoff symbols. The user then enters the atomic number, site occupation parameters and Debye-Waller factor first, followed by the Wyckoff position symbol. If this position requires any variables to be set, then the user will be asked to enter those values as well. Saving the file then proceeds as above.

9 List of programs

The list below shows all the available programs in the current Release (3.1) of the EMsoft suite, along with whether or not there is a manual for that program. Programs for which no manual is available are generally relatively easy to figure out. Programs indicated in blue are new in release 3.1. The abbreviation "DI" stands for Dictionary Indexing, a new approach to the indexing of electron diffraction patterns.

In the current release, many programs still have a command-line interaction with the user; this will be modified in a future release so that all programs will eventually make use of namelist files; at a much later point in the future, all programs will become accessible via a GUI, either the EMsoftToolbox or DREAM.3D. The top half of the table lists those programs for which extensive manual pages are available; in some cases there are IDL visualization routines as well. The EMmkxtal program is explained in the present document. The other programs in the bottom half of the list do not yet have manual pages, but are generally simple to figure out. Many of the programs in the bottom half of the table are simple illustrations of how to perform certain computations but they may have some general usefulness as well.

Note that several programs may be described in a single manual. Currently (release 3.1) the following manuals are available (in addition to this one): EMEBSD.pdf, EMIndexing.pdf, and HDFsupport.pdf (this last one is meant to explain our HDF5 support library for those who wish to write their own codes and make use of the HDF5 output format). Additional manuals will be added when we find the time to write them...

Program	Short description	Manual?
EMMCOpenCL	Monte Carlo simulation with OpenCL (SEM)	✓
EMOpenCLinfo	Lists information about the available compute hardware	
EMMC	Original Monte Carlo simulation, multithreaded (SEM)	
EMEBSDmaster	EBSD master pattern simulation (SEM)	✓
EMEBSD	Full EBSD pattern simulation (SEM)	✓
EMEBSDFull	Monte Carlo simulation, no master pattern (SEM)	✓
EMECPmaster	Electron channeling master pattern (SEM)	✓
EMECP	Electron channeling pattern (SEM)	✓
EMECPSingle	Single electron channeling pattern (SEM)	✓
EMPEDZA	Zone axis precession electron diffraction pattern (TEM)	
EMmergefiles	Merges version 3.0 Monte Carlo and master pattern files into a single file	
EMkinematical	Kinematical EBSD master pattern (SEM)	
EMECCI	ECCI defect images (SEM)	✓
EMKosselMaster	Electron Kossel master patterns (SEM)	✓
EMDPFit	Fit a diffraction pattern to determine detector parameters (SEM)	✓
EMEBSDDI	Dictionary indexing of EBSD patterns, "on-the-fly" (SEM)	✓
EMEBSDstatic	Dictionary indexing of EBSD patterns, precomputed dictionary (SEM)	✓
EMECPDI	Dictionary indexing of ECP patterns, "on-the-fly" (SEM)	✓
EMAverageOrient	Compute averaged orientations from DI (SEM)	
EMOrientationSimilarity	Compute an orientation similarity map from DI (SEM)	
EMKAM	Compute kernel average misorientation map from DI (SEM)	
EMsampleRFZ	Uniformly samples a Rodrigues Fundamental Zone for DI	
EMmkxtal	Make a crystal structure input file	✓
EMsoftinit	Initialize the EMsoft json information file	
EMlistSG	List the equivalent positions of any space group	
EMqg	List Fourier coefficient information	
EMfamily	Draw a family of planes (stereographic)	
EMstar	List the star of any reciprocal lattice point	
EMorbit	List the orbit of a point	
EMeqvrot	Convert an orientation into all equivalent orientations	
EMConvertOrientations	Convert a file of orientations into other orientation representations	
EMDisorientations	Compute disorientations between two lists of Euler angle triplets	
EMDisorientationsTwoPhase	Compute disorientations between two lists of Euler angle triplets	
EMOrientationViz	Create a PoV-Ray 3.7 input file based on a set of orientations	
EMZAgeom	Print zone axis geometry and symmetry information	
EMlatgeom	Perform some simple lattice geometry calculations	
EMstereo	Basic stereographic projection tool	
EMHOLZ	Basic kinematical HOLZ pattern zone axis simulation	
EMKikuchiMap	Basic kinematical Kikuchi band zone axis simulation	
EMorient	Stereographic projection for orientation relation	
EMxtalinfo	Makes postscript file with lots of useful information	
EMzap	Create postscript file with zone axis patterns	
EMdrawcell	Draw a unit cell (very primitive)	

10 Release Updates

3.1 (03/XX/17)

- Windows 10 and limited Linux CMake support
- dictionary indexing programs for EBSD, ECP, and PED diffraction modalities
- Monte Carlo and master pattern computations now employ a single output HDF5 file
- large number of additional utility programs
- substantial number of code tweaks and corrections

11 Future releases

The following future releases are currently planned:

- 3.2, Summer 2017: addition of updated TEM codes (currently still available in release 2.0); more complete integration with DREAM.3D. Support for the use of the Materials Data Facility; new EMsoftToolbox program.