OOMMF
Programming Manual

September 30, 2015

This manual documents release 1.2a6.

WARNING: In this alpha release, the documentation may not be up to date.

WARNING: This document in under construction.

Abstract

This manual provides source code level information on OOMMF (Object Oriented Micromagnetic Framework), a public domain micromagnetics program developed at the National Institute of Standards and Technology. Refer to the OOMMF User's Guide for an overview of the project and end-user details.

Contents

Disclaimer

We would appreciate acknowledgement if the software is used. When referencing OOMMF software, we recommend citing the NIST technical report, M. J. Donahue and D. G. Porter, "OOMMF User's Guide, Version 1.0," **NISTIR 6376**, National Institute of Standards and Technology, Gaithersburg, MD (Sept 1999).

1 Programming Overview of OOMMF

The OOMMF[1] (Object Oriented Micromagnetic Framework) project in the Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) is intended to develop a portable, extensible public domain micromagnetic program and associated tools. This manual aims to document the programming interfaces to OOMMF at the source code level. The main developers of this code are Mike Donahue and Don Porter.

The underlying numerical engine for OOMMF is written in C++, which provides a reasonable compromise with respect to efficiency, functionality, availability and portability. The interface and glue code is written primarily in Tcl/Tk, which hides most platform specific issues. Tcl and Tk are available for free download [2] from the Tcl Developer Xchange[3].

The code may actually be modified at 3 distinct levels. At the top level, individual programs interact via well-defined protocols across network sockets. One may connect these modules together in various ways from the user interface, and new modules speaking the same protocol can be transparently added. The second level of modification is at the Tcl/Tk script level. Some modules allow Tcl/Tk scripts to be imported and executed at run time, and the top level scripts are relatively easy to modify or replace. The lowest level is the C++ source code. The OOMMF extensible solver, OXS, is designed with modification at this level in mind.

If you want to receive e-mail notification of updates to this project, register your e-mail address with the "μMAG Announcement" mailing list:

http://www.ctcms.nist.gov/~rdm/email-list.html.

The OOMMF developers are always interested in your comments about OOMMF. See the Credits (Sec. 6) for instructions on how to contact them.

[1]http://math.nist.gov/oommf/

[2]http://purl.org/tcl/home/software/tcltk/choose.html

[3]http://purl.org/tcl/home/

2 Platform-Independent Make

UNDER CONSTRUCTION

Details on pimake go here.

Somewhere we should have documentation on feeding and breeding makerules.tcl files. Should that be here, or in a separate section? If the former, then should this section be renamed?

3 OOMMF Variable Types and Macros

The following typedefs are defined in the `oommf/pkg/oc/`*platform*`/ocport.h` header file; this file is created by the **pimake** build process (see `oommf/pkg/oc/procs.tcl`), and contains platform and machine specific information.

- `OC_BOOL` Boolean type, unspecified width.
- `OC_BYTE` Unsigned integer type exactly one byte wide.
- `OC_CHAR` Character type, may be signed or unsigned.
- `OC_UCHAR` Unsigned character type.
- `OC_SCHAR` Signed character type. If `signed char` is not supported by a given compiler, then this falls back to a plain `char`, so use with caution.
- `OC_INT2, OC_INT4` Signed integer with width of exactly 2, respectively 4, bytes.
- `OC_INT2m, OC_INT4m` Signed integer with width of at least 2, respectively 4, bytes. A type wider than the minimum may be specified if the wider type is handled faster by the particular machine.
- `OC_UINT2, OC_UINT4, OC_UINT2m, OC_UINT4m` Unsigned integer versions of the preceding.
- `OC_REAL4, OC_REAL8` Four byte, respectively eight byte, floating point variable. Typically corresponds to C++ “float” and “double” types.
- `OC_REAL4m, OC_REAL8m` Floating point variable with width of at least 4, respectively 8, bytes. A type wider than the minimum may be specified if the wider type is handled faster by the particular machine.
- `OC_REALWIDE` Widest type natively supported by the underlying hardware. This is usually the C++ “long double” type, but may be overridden by the

  `program_compiler_c++_typedef_realwide`

  option in the `oommf/config/platform/`*platform*`.tcl` file.

The `oommf/pkg/oc/`*platform*`/ocport.h` header file also defines the following macros for use with the floating point variable types:

- `OC_REAL8m_IS_DOUBLE` True if `OC_REAL8m` type corresponds to the C++ “double” type.
- `OC_REAL8m_IS_REAL8` True if `OC_REAL8m` and `OC_REAL8` refer to the same type.

- `OC_REAL4_EPSILON` The smallest value that can be added to a `OC_REAL4` value of "1.0" and yield a value different from "1.0". For IEEE 754 compatible floating point, this should be `1.1920929e-007`.
- `OC_SQRT_REAL4_EPSILON` Square root of the preceding.
- `OC_REAL8_EPSILON` The smallest value that can be added to a `OC_REAL8` value of "1.0" and yield a value different from "1.0". For IEEE 754 compatible floating point, this should be `2.2204460492503131e-016`.
- `OC_SQRT_REAL8_EPSILON, OC_CUBE_ROOT_REAL8_EPSILON` Square and cube roots of the preceding.
- `OC_FP_REGISTER_EXTRA_PRECISION` True if intermediate floating point operations use a wider precision than the floating point variable type; notably, this occurs with some compilers on x86 hardware.

Note that all of the above macros have a leading "`OC_`" prefix. The prefix is intended to protect against possible name collisions with system header files. Versions of some of these macros are also defined without the prefix; these definitions represent backward support for existing OOMMF extensions. All new code should use the versions with the "`OC_`" prefix, and old code should be updated where possible. The complete list of deprecated macros is:

```
BOOL, UINT2m, INT4m, UINT4m, REAL4, REAL4m, REAL8, REAL8m, REALWIDE,
REAL4_EPSILON, REAL8_EPSILON, SQRT_REAL8_EPSILON, CUBE_ROOT_REAL8_EPSILON,
FP_REGISTER_EXTRA_PRECISION
```

Macros for system identification:

- `OC_SYSTEM_TYPE` One of `OC_UNIX` or `OC_WINDOWS`.
- `OC_SYSTEM_SUBTYPE` For unix systems, this is either `OC_VANILLA` (general unix) or `OC_DARWIN` (Mac OS X). For Windows systems, this is generally `OC_WINNT`, unless one is running out of a Cygwin shell, in which case the value is `OC_CYGWIN`.

Additional macros and typedefs:

- `OC_POINTERWIDTH` Width of pointer type, in bytes.
- `OC_INDEX` Typedef for signed array index type; typically the width of this (integer) type matches the width of the pointer type, but is in any event at least four bytes wide and not narrower than the pointer type.
- `OC_UINDEX` Typedef for unsigned version of OC_INDEX. It is intended for special-purpose use only. In general, use OC_INDEX where possible.
- `OC_INDEX_WIDTH` Width of `OC_INDEX` type.

- `OC_BYTEORDER` Either “4321” for little endian machines, or “1234” for big endian.
- `OC_THROW(x)` Throws a C++ exception with value “x”.
- `OOMMF_THREADS` True threaded (multi-processing) builds.
- `OC_USE_NUMA` If true, then NUMA (non-uniform memory access) libraries are available.

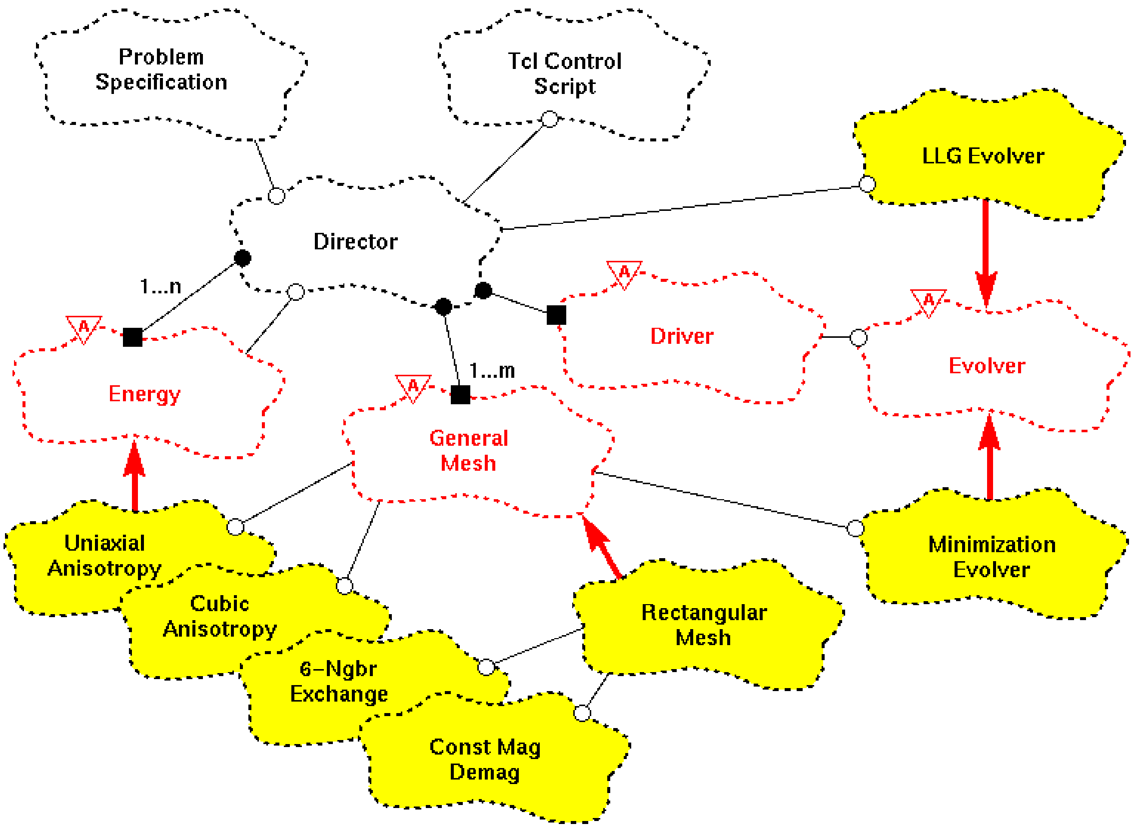

Figure 1: OXS top-level class diagram.

4 OOMMF eXtensible Solver

The OOMMF eXtensible Solver (OXS) top level architecture is shown in Fig. 1. The "Tcl Control Script" block represents the user interface and associated control code, which is written in Tcl. The micromagnetic problem input file is the content of the "Problem Specification" block. The input file should be a valid MIF 2.0 file (see the OOMMF User's Guide for details on the MIF file formats), which also happens to be a valid Tcl script. The rest of the architecture diagram represents C++ classes.

All interactions between the Tcl script level and the core solver are routed through the Director object. Aside from the Director, all other classes in this diagram are examples of `Oxs_Ext` objects—technically, C++ child classes of the abstract `Oxs_Ext` class. OXS is designed to be extended primarily by the addition of new `Oxs_Ext` child classes.

The general steps involved in adding an `Oxs_Ext` child class to OXS are:

1. Add new source code files to `oommf/app/oxs/local` containing your class definitions. The C++ non-header source code file(s) must be given the `.cc` extension. (Header files are typically denoted with the `.h` extension, but this is not mandatory.)

2. Run **pimake** to compile your new code and link it in to the OXS executable.

3. Add the appropriate `Specify` blocks to your input MIF 2.0 files.

The source code can usually be modeled after an existing `Oxs_Ext` object. Refer to the Oxsii section of the OOMMF User's Guide for a description of the standard `Oxs_Ext` classes, or Sec. 4.1 for an annotated example of an `Oxs_Energy` class. Base details on adding a new energy term are presented in Sec. 4.2.

The **pimake** application automatically detects all files in the `oommf/app/oxs/local` directory with the `.cc` extension, and searches them for `#include` requests to construct a build dependency tree. Then **pimake** compiles and links them together with the rest of the OXS files into the **oxs** executable. Because of the automatic file detection, no modifications are required to any files of the standard OOMMF distribution in order to add local extensions.

Local extensions are then activated by `Specify` requests in the input MIF 2.0 files. The object name prefix in the `Specify` block is the same as the C++ class name. All `Oxs_Ext` classes in the standard distribution are distinguished by an `Oxs_` prefix. It is recommended that local extensions use a local prefix to avoid name collisions with standard OXS objects. (C++ namespaces are not currently used in OOMMF for compatibility with some older C++ compilers.) The `Specify` block initialization string format is defined by the `Oxs_Ext` child class itself; therefore, as the extension writer, you may choose any format that is convenient. However, it is recommended that you follow the conventions laid out in the MIF 2.0 file format section of the OOMMF User's Guide.

4.1 Sample `Oxs_Energy` Class

This sections provides an extended dissection of a simple `Oxs_Energy` child class. The computational details are kept as simple as possible, so the discussion can focus on the C++ class structural details. Although the calculation details will vary between energy terms, the class structure issues discussed here apply across the board to all energy terms.

The particular example presented here is for simulating uniaxial magneto-crystalline energy, with a single anisotropy constant, `K1`, and a single axis, `axis`, which are uniform across the sample. The class definition (.h) and code (.cc) are displayed in Fig. 2 and 3, respectively.

```
/* FILE: exampleanisotropy.h
 *
 * Example anisotropy class definition.
 * This class is derived from the Oxs_Energy class.
 *
 */

#ifndef _OXS_EXAMPLEANISOTROPY
#define _OXS_EXAMPLEANISOTROPY
```

```
#include "energy.h"
#include "threevector.h"
#include "meshvalue.h"

/* End includes */

class Oxs_ExampleAnisotropy:public Oxs_Energy {
private:
  double K1;        // Primary anisotropy coeficient
  ThreeVector axis; // Anisotropy direction
public:
  virtual const char* ClassName() const; // ClassName() is
  /// automatically generated by the OXS_EXT_REGISTER macro.
  virtual BOOL Init();
  Oxs_ExampleAnisotropy(const char* name,  // Child instance id
Oxs_Director* newdtr, // App director
Tcl_Interp* safe_interp, // Safe interpreter
const char* argstr);  // MIF input block parameters

  virtual ~Oxs_ExampleAnisotropy() {}

  virtual void GetEnergyAndField(const Oxs_SimState& state,
                                 Oxs_MeshValue<REAL8m>& energy,
                                 Oxs_MeshValue<ThreeVector>& field
                                 ) const;
};


#endif // _OXS_EXAMPLEANISOTROPY
```

Figure 2: Example energy class definition.

```
/* FILE: exampleanisotropy.cc            -*-Mode: c++-*-
 *
 * Example anisotropy class implementation.
 * This class is derived from the Oxs_Energy class.
 *
 */

#include "exampleanisotropy.h"
```

```
// Oxs_Ext registration support
OXS_EXT_REGISTER(Oxs_ExampleAnisotropy);

/* End includes */

#define MU0          12.56637061435917295385e-7   /* 4 PI 10^7 */

// Constructor
Oxs_ExampleAnisotropy::Oxs_ExampleAnisotropy(
  const char* name,     // Child instance id
  Oxs_Director* newdtr, // App director
  Tcl_Interp* safe_interp, // Safe interpreter
  const char* argstr)   // MIF input block parameters
  : Oxs_Energy(name,newdtr,safe_interp,argstr)
{
  // Process arguments
  K1=GetRealInitValue("K1");
  axis=GetThreeVectorInitValue("axis");
  VerifyAllInitArgsUsed();
}

BOOL Oxs_ExampleAnisotropy::Init()
{ return 1; }

void Oxs_ExampleAnisotropy::GetEnergyAndField
(const Oxs_SimState& state,
 Oxs_MeshValue<REAL8m>& energy,
 Oxs_MeshValue<ThreeVector>& field
 ) const
{
  const Oxs_MeshValue<REAL8m>& Ms_inverse = *(state.Ms_inverse);
  const Oxs_MeshValue<ThreeVector>& spin = state.spin;
  UINT4m size = state.mesh->Size();

  for(UINT4m i=0;i<size;++i) {
    REAL8m field_mult = (2.0/MU0)*K1*Ms_inverse[i];
    if(field_mult==0.0) {
      energy[i]=0.0;
      field[i].Set(0.,0.,0.);
      continue;
    }
```

```
    REAL8m dot = axis*spin[i];
    field[i] = (field_mult*dot) * axis;
    if(K1>0) {
      energy[i] = -K1*(dot*dot-1.0); // Make easy axis zero energy
    } else {
      energy[i] = -K1*dot*dot; // Easy plane is zero energy
    }
  }
}
```

Figure 3: Example energy class code.

4.2 Writing a New Oxs_Energy Extension

Under construction.

5 References

[1] W. F. Brown, Jr., *Micromagnetics* (Krieger, New York, 1978).

[2] M. J. Donahue and D. G. Porter, *OOMMF User's Guide, Version 1.0*, Tech. Rep. NISTIR 6376, National Institute of Standards and Technology, Gaithersburg, MD (1999).

6 Credits

The main contributors to this document are Michael J. Donahue (michael.donahue@nist.gov) and Donald G. Porter (donald.porter@nist.gov), both of ITL/NIST.

If you have bug reports, contributed code, feature requests, or other comments for the OOMMF developers, please send them in an e-mail message to `<michael.donahue@nist.gov>`.

Index