OOMMF
User's Guide

November 12, 2026

This manual documents release 2.1a4.

This manual describes OOMMF (Object Oriented Micromagnetic Framework), a public domain micromagnetics program developed at the National Institute of Standards and Technology. The program is designed to be portable, flexible, and extensible, with a user-friendly graphical interface. The code is written in C++ and Tcl/Tk. Target systems include a wide range of Unix, Windows, and macOS platforms.

Table of Contents

Disclaimer v

1 Overview of OOMMF 1

2 Installation 3
- 2.1 Requirements . . . 3
- 2.2 Basic Installation . . . 4
  - 2.2.1 Download . . . 4
  - 2.2.2 Effects of the Installed Tcl/Tk . . . 5
  - 2.2.3 Check Your Platform Configuration . . . 5
  - 2.2.4 Compiling and Linking . . . 9
  - 2.2.5 Installing . . . 10
  - 2.2.6 Using OOMMF Software . . . 10
  - 2.2.7 Reporting Problems . . . 11
- 2.3 Advanced Installation . . . 11
  - 2.3.1 Reducing Disk Space Usage . . . 11
  - 2.3.2 Local Customizations . . . 11
  - 2.3.3 Optimization . . . 13
  - 2.3.4 Parallelization . . . 14
  - 2.3.5 Managing OOMMF Platform Names . . . 15
- 2.4 Platform Specific Installation Issues . . . 17
  - 2.4.1 Unix Configuration . . . 17
  - 2.4.2 macOS Configuration . . . 19
  - 2.4.3 Microsoft Windows Options . . . 19

3 Quick Start: Example OOMMF Session 21

4 OOMMF Architecture Overview 27

5 Command Line Launching 29

6 OOMMF Launcher/Control Interface: mmLaunch 33

7 OOMMF eXtensible Solver 35

7.1 OOMMF eXtensible Solver Interactive Interface: Oxsii . . . 36
7.2 OOMMF eXtensible Solver Batch Interface: Boxsi . . . 42
7.3 Standard Oxs_Ext Child Classes . . . 49
7.3.1 Atlases . . . 50
7.3.2 Meshes . . . 61
7.3.3 Energies . . . 62
7.3.4 Evolvers . . . 79
7.3.5 Drivers . . . 91
7.3.6 Field Objects . . . 97
7.3.7 MIF Support Classes . . . 116
7.4 Contributed Oxs_Ext Child Classes . . . 117

8 Micromagnetic Problem Editor: mmProbEd 118

9 Micromagnetic Problem File Source: FileSource 121

10 The 2D Micromagnetic Solver 123
10.1 The 2D Micromagnetic Interactive Solver: mmSolve2D . . . 124
10.2 OOMMF 2D Micromagnetic Solver Batch System . . . 130
10.2.1 2D Micromagnetic Solver Batch Interface: batchsolve . . . 130
10.2.2 2D Micromagnetic Solver Batch Scheduling System . . . 133

11 Data Table Display: mmDataTable 142

12 Data Graph Display: mmGraph 145

13 Vector Field Display: mmDisp 151

14 Data Archive: mmArchive 165

15 Documentation Viewer: mmHelp 168

16 Command Line Utilities 171
16.1 Bitmap File Format Conversion: any2ppm . . . 171
16.2 Making Data Tables from Vector Fields: avf2odt . . . 172
16.3 Vector Field File Format Conversion: avf2ovf . . . 177
16.4 Making Bitmaps from Vector Fields: avf2ppm . . . 181
16.5 Making PostScript and Scalable Vector Graphics from Vector Fields: avf2ps 184
16.6 Making Scalable Vector Graphics from Vector Fields: avf2svg . . . 188
16.7 Vector Field File Difference: avfdiff . . . 188
16.8 Field from Magnetization (3D): computefield . . . 191
16.9 Cyclic Redundancy Check: crc32 . . . 194
16.10 Killing OOMMF Processes: killoommf . . . 195
16.11 Last Oxsii/Boxsi Run: lastjob . . . 196

16.12 The OOMMF Host Server: launchhost . . . 197
16.13 Oxs Log File Review: logreview . . . 199
16.14 **H** Fields from Magnetization (2D): mag2hfield . . . 201
16.15 MIF Format Conversion: mifconvert . . . 202
16.16 PyMIF to MIF/Tcl Format Conversion: mifpy2tcl . . . 203
16.17 Process Nicknames: nickname . . . 204
16.18 ODT Derived Quantity Calculator: odtcalc . . . 205
16.19 ODT Table Concatenation: odtcat . . . 206
16.20 ODT Column Extraction: odtcols . . . 207
16.21 Oxs Package Management: oxspkg . . . 209
16.22 Oxs Regression Tests: oxsregression . . . 211
16.23 OOMMF and Process ID Information: pidinfo . . . 213
16.24 Platform-Independent Make: pimake . . . 214
16.25 Suspending Oxs Processes: suspend . . . 217

17 Problem Specification File Formats (MIF) 218
17.1 MIF 1.1 . . . 218
17.1.1 Material parameters . . . 219
17.1.2 Demag specification . . . 220
17.1.3 Part geometry . . . 221
17.1.4 Initial magnetization . . . 222
17.1.5 Experiment parameters . . . 223
17.1.6 Output specification . . . 225
17.1.7 Miscellaneous . . . 225
17.2 MIF 1.2 . . . 227
17.3 MIF 2.1 . . . 228
17.3.1 MIF 2.1 File Overview . . . 228
17.3.2 MIF 2.1 Extension Commands . . . 230
17.3.3 Specify Conventions . . . 239
17.3.4 Variable Substitution . . . 249
17.3.5 Sample MIF 2.1 File . . . 251
17.4 MIF 2.2 . . . 254
17.4.1 Differences between MIF 2.2 and MIF 2.1 Formats . . . 254
17.4.2 MIF 2.2 New Extension Commands . . . 254
17.4.3 Sample MIF 2.2 File . . . 259
17.5 Tips for writing MIF 2.x files . . . 263
17.6 PyMIF 1.0 . . . 266
17.6.1 Introduction to PyMIF 1.0 . . . 266
17.6.2 Sample PyMIF 1.0 File . . . 269

18 Data Table File Format (ODT) 271

19 Vector Field File Format (OVF) 273

19.1 The OVF 0.0 format 273
19.2 The OVF 1.0 format 275
19.2.1 Segment Header block 276
19.2.2 Data block 277
19.3 The OVF 2.0 format 280

20 Integer Field File Format (OIF) 283

21 Troubleshooting 287

22 Credits 292

Bibliography 294

Index 297

Disclaimer

NIST-developed software is provided by NIST as a public service. You may use, copy, and distribute copies of the software in any medium, provided that you keep intact this entire notice. You may improve, modify, and create derivative works of the software or any portion of the software, and you may copy and distribute such modifications or works. Modified works should carry a notice stating that you changed the software and should note the date and nature of any such change. Please explicitly acknowledge the National Institute of Standards and Technology as the source of the software.

NIST-developed software is expressly provided "AS IS." NIST MAKES NO WARRANTY OF ANY KIND, EXPRESS, IMPLIED, IN FACT, OR ARISING BY OPERATION OF LAW, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, AND DATA ACCURACY. NIST NEITHER REPRESENTS NOR WARRANTS THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT ANY DEFECTS WILL BE CORRECTED. NIST DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF THE SOFTWARE OR THE RESULTS THEREOF, INCLUDING BUT NOT LIMITED TO THE CORRECTNESS, ACCURACY, RELIABILITY, OR USEFULNESS OF THE SOFTWARE.

You are solely responsible for determining the appropriateness of using and distributing the software and you assume all risks associated with its use, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and the unavailability or interruption of operation. This software is not intended to be used in any situation where a failure could cause risk of injury or damage to property. The software developed by NIST employees is not subject to copyright protection within the United States.

Commercial equipment and software referred to on these pages are identified for informational purposes only, and does not imply recommendation of or endorsement by the National Institute of Standards and Technology, nor does it imply that the products so identified are necessarily the best available for the purpose.

We appreciate acknowledgement when the software is used. We recommend this format:

Donahue, M.J. and Porter, D.G. (2026), The Object Oriented MicroMagnetic Framework (OOMMF), Version 2.1a4, National Institute of Standards and Technology, [Software], https://math.nist.gov/oommf/ (Accessed November 12, 2026). DOI: 10.18434/T4/1502495

Chapter 1

Overview of OOMMF

Micromagnetics is a continuum model of the magnetization behavior of ferromagnetic materials. Developed by W. F. Brown Jr. in the 1940s and '50s [4, 5], it allows for a more detailed description of magnetization structure and behavior [24] than the earlier domain theory [33] or Stoner-Wohlfarth models [30]. Computing resources at that time, however, limited implementations to relatively simple systems [17]. It wasn't until the 1990s that fully 2D and later 3D systems of technological interest could be successfully modeled [1, 10].

The size of features such as domain walls, vortices, and cross-ties depends on material parameters (e.g., saturation magnetization M_s, magneto-crystalline anisotopy K, exchange coefficient A), applied field (H_{app}), and part geometry, but the typical length scale is on the order of a few nanometers. Arbitrary shapes and material variations may be modeled, though part size will be limited by the available computing power. Micromagnetics can be used to study static magnetization structures, for example ground states in small particles [15] and domain walls in film strips [21], quasi-static behavior to model M vs. H hystersis loops [16], and field-driven magnetization dynamics using the Landau-Lifshitz-Gilbert equation [13, 18, 22]. More recent extensions include spin-torque [23, 28, 31, 35, 36], thermal [12], and DMI [32] effects.

The goal of the OOMMF[1] (Object Oriented MicroMagnetic Framework) project in the Information Technology Laboratory[2] (ITL) at the National Institute of Standards and Technology[3] (NIST) is to develop a portable, extensible public domain micromagnetic program and associated tools [1, 5, 10]. The first release of OOMMF, based on a micromagnetic code previously developed by Robert McMichael and Michael Donahue, was made in October, 1998. That release included the 2D micromagnetic solver, problem editor, and several display widgets. Current releases implement a completely functional, fully 3D micromagnetics package, with the capability of being extended by other programmers so that people developing new code can build on the OOMMF foundation. The primary developers of OOMMF are Mike Donahue and Don Porter.

[1] https://math.nist.gov/oommf/
[2] https://www.nist.gov/itl/
[3] https://www.nist.gov/

OOMMF is written in C++, a widely-available, object-oriented language that can produce programs with good performance as well as extensibility. The code uses Tcl/Tk to create a portable user interface allowing OOMMF to operate across a wide range of Unix, Windows, and macOS platforms.

The code may be modified at three 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. At the lowest level, the C++ source is provided and can be modified (see the *OOMMF Programming Manual*[4]). The primary extension mechanism at this level is through the OOMMF eXtensible Solver, Oxs. The extensible nature of Oxs allows its capabilities to be varied as necessary for the problem at hand, and lets OOMMF users extend Oxs with external modules.[5]

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

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

The OOMMF developers are always interested in your comments about OOMMF. See the Credits for instructions on how to contact them, and for information on citing OOMMF.

[4] https://math.nist.gov/oommf/doc/
[5] https://math.nist.gov/oommf/contrib/oxsext/

Chapter 2

Installation

2.1 Requirements

OOMMF software is written in C++ and Tcl. It uses the Tcl-based Tk Windowing Toolkit to create graphical user interfaces that are portable to many varieties of Unix, Windows, and macOS. Optionally, Python 3 is needed if one wants to use the PyMIF micromagnetic simulation input format.

Tcl and Tk must be installed before installing OOMMF. Tcl and Tk are available for free from the Tcl Developer Xchange[1]. We recommend the latest stable versions of Tcl and Tk concurrent with this release of OOMMF. OOMMF requires at least version 8.5 of Tcl and Tk. OOMMF software does not support any alpha or beta versions of Tcl/Tk, and each release of OOMMF may not work with later releases of Tcl/Tk. Check the release dates of both OOMMF and Tcl/Tk to ensure compatibility.

A Tcl/Tk installation includes two shell programs. The names of these programs may vary depending on the Tcl/Tk version and the type of platform. The first shell program contains an interpreter for the base Tcl language. In the OOMMF documentation we refer to this program as `tclsh`. The second shell program contains an interpreter for the base Tcl language extended by additional Tcl commands supplied by the Tk toolkit. In the OOMMF documentation we refer to this program as `wish`. Consult your Tcl/Tk documentation to determine the actual names of these programs on your platform (for example, `tclsh86.exe` or `wish8.6`).

OOMMF applications communicate via TCP/IP network sockets. This means that OOMMF requires support for networking, even on a stand-alone machine. At a minimum, OOMMF must be able to access the loopback interface so that the host can talk to itself using TCP/IP.

OOMMF applications that use Tk require a windowing system and a valid display. On Unix systems, this means that an X server must be running. If you need to run OOMMF applications on a Unix system without display hardware or software, you may need to start

[1] https://www.tcl-lang.org/

the application with command line option `-tk 0` or use the Xvfb[2] virtual frame buffer.

To build OOMMF software from source code, you will need a C++ compiler that implements the features of the C++11 standard. You will need other software development utilities for your platform as well. We do development and test builds on the following platforms, although porting to others should not be too difficult:

Platform	Compilers
Windows	Microsoft Visual C++, Intel C++, MinGW g++, Cygwin g++
Linux/x86	Gnu g++, Intel C++
macOS	Clang C++, Gnu g++

2.2 Basic Installation

Follow the instructions in the following sections, in order, to prepare OOMMF software for use on your computer.

2.2.1 Download

The latest release of the OOMMF software may be retrieved from the OOMMF download page[3]. Each release is available in two formats. The first format is a gzipped tar file containing an archive of all the OOMMF source code. The second format is a `.zip` compressed archive containing source code and pre-compiled executables for Windows. Each Windows binary distribution is compatible with only a particular sequence of releases of Tcl/Tk. For example, a Windows binary release for Tcl/Tk 8.6.x is compatible with Tcl/Tk 8.6.0, 8.6.1, ...

For the first format, unpack the distribution archive using gunzip and tar:

```
gunzip -c oommf20a0.tar.gz | tar xvf -
```

For the other format(s), you will need a utility program to unpack the `.zip` archive. One utility program which is known to be suitable is UnZip[4].

Using your utility, unpack the `.zip` archive, e.g.

```
unzip oommf20a0_86.zip
```

For either distribution format, the unpacking sequence creates a subdirectory `oommf` which contains all the files and directories of the OOMMF distribution. If a subdirectory named `oommf` already existed (say, from an earlier OOMMF release), then files in the new distribution overwrite those of the same name already on the disk. Some care may be needed in that circumstance to be sure that the resulting mix of files from an old and a new OOMMF distribution combine to create a working set of files.

[2] https://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml
[3] https://math.nist.gov/oommf/software.html
[4] http://www.info-zip.org/pub/infozip/UnZip.html

2.2.2 Effects of the Installed Tcl/Tk

OOMMF interacts with your Tcl/Tk installation in several ways. One important restriction is that the major+minor release number of Tcl/Tk must match the major+minor release number of the Tcl/Tk that OOMMF was built against. For example, if OOMMF was built using Tcl/Tk 8.5.18, then the resulting executables can run with any past or future releases of Tcl/Tk from the 8.5.* series, but they won't run (for example) with Tcl/Tk 8.4.20 or 8.6.4.

Another restriction is that the width of memory addresses in Tcl/Tk and OOMMF must match. Most general-purpose operating systems today use primarily 64-bit memory addresses, but for backwards compatibility can also run programs using 32-bit memory addresses. However, a 64-bit executable cannot link against a 32-bit library, or vice versa. Therefore, if you have a 64-bit Tcl/Tk installed, then you will need a 64-bit OOMMF, and likewise a 32-bit Tcl/Tk needs a 32-bit OOMMF.

Another restriction is that while OOMMF can be built to run in parallel across multiple cpu cores on a shared memory machine using threads, to do this requires that the installed Tcl/Tk be thread-enabled. Typical Tcl/Tk installs on Windows and macOS are thread-enabled. Tcl/Tk installs on recent releases of Unix also tend to be thread-enabled, but some older versions have non-threaded Tcl/Tk installs. If your system Tcl/Tk install is non-threaded, then you can either build a non-threaded version of OOMMF, or else you can make an additional, threaded Tcl/Tk install, for example under your home directory or `/usr/local`. Be aware that if you have multiple Tcl/Tk installations on your system then you need to be careful to use the proper `tclsh` whenever you build or launch OOMMF.

If you download OOMMF with pre-built binaries, then it is imperative that you select the download that matches the major+minor release number and memory address width of the Tcl/Tk you want to run OOMMF with. On the other hand, if you build OOMMF from source, then the `tclsh` you use to run the build process is inspected to determine relevant information about the local Tcl/Tk environment. Some adjustment of the platform configuration file, as described in the next section, may be necessary. Also, in many cases the compilers used to build 32-bit and 64-bit executables are different—if you encounter build problems, double-check that the proper compiler is being used.

All of the OOMMF downloads containing pre-built binaries are built for use with thread-enabled Tcl/Tk. You will need to build from source if you want a non-threaded OOMMF. The build scripts will detect if the `tclsh` running the build procedure is non-threaded and will build OOMMF appropriately.

In all cases, use the platform configuration check described in the next section to verify the compatibility of your Tcl/Tk and OOMMF installs.

2.2.3 Check Your Platform Configuration

After downloading and unpacking the OOMMF software distribution, all the OOMMF software is contained in a subdirectory named `oommf`. Start a command line interface (a shell on Unix, or a console on Windows), and change the working directory to the directory `oommf`. Find the Tcl shell program installed as part of your Tcl/Tk installation. In this manual

we call the Tcl shell program `tclsh`, but the actual name of the executable depends on the release of Tcl/Tk and your platform type. Consult your Tcl/Tk documentation.

In the root directory of the OOMMF distribution is a file named `oommf.tcl`. It is the bootstrap application which is used to launch all OOMMF software. With the command line argument `+platform`, it will print a summary of your platform configuration when it is evaluated by `tclsh`. This summary describes your platform type, your C++ compiler, and your Tcl/Tk installation. As an example, here is the typical output on a macOS 10.9 system:

```
$ tclsh oommf.tcl +platform
<5426> oommf.tcl 1.2.0.6  info:
OOMMF release 1.2.0.6, snapshot 2015.03.25
Platform Name:    darwin
Tcl name for OS:  Darwin 13.4.0
C++ compiler:     /usr/bin/g++
 Version string:  Apple LLVM version 6.0 (clang-600.0.57) (based on
     LLVM 3.5svn) / Target: x86_64-apple-darwin13.4.0 / Thread model: posix
Shell details ---
 tclsh (running): /usr/bin/tclsh
     (links to /usr/bin/tclsh8.5)
     (links to /System/Library/Frameworks/Tcl.framework/Versions/8.5/tclsh8.5)
                      --> Version 8.5.9, 64 bit, threaded
 tclsh (OOMMF):   /usr/bin/tclsh8.5
                   --> Version 8.5.9, 64 bit, threaded
 filtersh:        /Users/xerxes/oommf/app/omfsh/darwin/filtersh
                   --> Version 8.5.9, 64 bit, threaded
 tclConfig.sh:    /System/Library/Frameworks/Tcl.framework/Versions/8.5/tclConfig.sh
                   --> Version 8.5.9
 wish (OOMMF):    /usr/bin/wish8.5
                   --> Version 8.5.9, Tk 8.5.9, 64 bit, threaded
 tkConfig.sh:     /System/Library/Frameworks/Tk.framework/Versions/8.5/tkConfig.sh
                   --> Tk Version 8.5.9
 Python (OOMMF):  /usr/bin/python3
                   --> Python version 3.9.6
OOMMF threads:    Yes: Default thread count = 2
OOMMF API index:  20150129
Temp file directory: /var/folders/dy/srfj33512f51kc5knp_lph_r0000gp/T/
```

If `oommf.tcl +platform` doesn't print a summary similar to the above, it should instead print an error message describing why it can't. Follow any instructions provided and repeat until `oommf.tcl +platform` successfully prints a summary of the platform configuration information.

The first line of the example summary reports that OOMMF recognizes the platform

by the name `darwin`. OOMMF software recognizes many of the more popular computing platforms, and assigns each a platform name. The platform name is used by OOMMF in index and configuration files and to name directories so that a single OOMMF installation can support multiple platform types. If `oommf.tcl +platform` reports the platform name to be "unknown", then you will need to add some configuration files to help OOMMF assign a name to your platform type, and associate with that name some of the key features of your computer. See the section on Managing OOMMF platform names for further instructions.

The second line reports the operating system version, which is mainly useful to OOMMF developers when fielding bug reports. The third line reports what C++ compiler will be used to build OOMMF from its C++ source code. If you downloaded an OOMMF release with pre-compiled binaries for your platform, you may ignore this line. Otherwise, if this line reports "none selected", or if it reports a compiler other than the one you wish to use, then you will need to tell OOMMF what compiler to use. To do that, you must edit the appropriate configuration file for your platform. Continuing the example above, one would edit the file `config/platforms/darwin.tcl`. Editing instructions are contained within the file. On other platforms the name `darwin` in `config/platforms/darwin.tcl` should be replaced with the platform name OOMMF reports for your platform. For example, on a 32-bit Windows machine using an x86 processor, the corresponding configuration file is `config/platforms/wintel.tcl`.

The next group of lines describe the Tcl configuration OOMMF finds on your platform. The first couple of lines, "tclsh (running)", describe the Tcl shell running the oommf.tcl script. After that, the "tclsh (OOMMF)" subgroup describes the Tcl shell that OOMMF will launch when it needs to run Tcl scripts. If the OOMMF binaries have been built, then there will also be a `filtersh` subgroup, which describes the augmented Tcl shell used to run many of the OOMMF support scripts. All of these shells should report the same version, bitness, and threading information. If OOMMF can't find `tclsh`, or if it finds the wrong one, you can correct this by setting the environment variable OOMMF_TCLSH to the absolute location of `tclsh`. (For information about setting environment variables, see your operating system documentation.)

Following the Tcl shell information, the `tclConfig.sh` lines report the name of the configuration file installed as part of Tcl, if any. Conventional Tcl installations on Unix systems and within the Cygwin environment on Windows have such a file, usually named `tclConfig.sh`. The Tcl configuration file records details about how Tcl was built and where it was installed. On Windows platforms, this information is recorded in the Windows registry, so it is normal to have `oommf.tcl +platform` report "none found". If `oommf.tcl +platform` reports "none found", but you know that an appropriate Tcl configuration file is present on your system, you can tell OOMMF where to find the file by setting the environment variable OOMMF_TCL_CONFIG to its absolute filename. In unusual circumstances, OOMMF may find a Tcl configuration file which doesn't correctly describe your Tcl installation. In that case, use the environment variable OOMMF_TCL_CONFIG to instruct OOMMF to use a different file that you specify, and, if necessary, edit that file to include a correct description of your Tcl installation.

Next, the `oommf.tcl +platform` reports similar information about the `wish` and Tk configuration. The environment variables `OOMMF_TK_CONFIG` and `OOMMF_WISH` may be used to tell OOMMF where to find the Tk configuration file and the `wish` program, respectively.

If Python 3 is install on your system, then the next two lines should report the location and version of the Python executable. The environment variable `OOMMF_PYTHON` or the corresponding setting in the `config/options.tcl` file can be used to override the default setting. The `OOMMF_NO_PYTHON` environment variable can be set to 1 to disable Python use. (Python is used to process PyMIF micromagnetic simulation input files.)

Following the Python or Tk information are some lines reporting "thread" build and run status. Threads are used by OOMMF to implement parallelism in the Oxs (`oxsii` and `boxsi`) 3D solvers on multi-processor/multi-core shared memory machines. In order to build or run a parallel version of OOMMF, you must have a thread-enabled version of Tcl. The Tcl thread status is indicated on the first thread status line. If Tcl is thread enabled, then the default OOMMF build process will create a threaded version of OOMMF. You can override this behavior if you wish to build a non-parallel version of OOMMF by editing the `oommf_threads` value in the `config/platforms/` file for your platform.

If Tcl and OOMMF threads are enabled, then the default number of threads run by the Oxs solvers is also reported. (This value may vary between machines, depending on the number of processors in the machine.) You can change this by setting (in order of increasing precedence) the `oommf_thread_count` value in the installation-wide `config/options` file, the `thread_count` value in the `config/platforms/` file for your platform, via the environment variable `OOMMF_THREADS`, or by the `oxsii`/`boxsi` command line option `-threads`.

By default, OOMMF sets no upper limit on the number of threads you may run in `oxsii` or `boxsi`. However, performance is degraded if you run more threads than available cpu cores. To protect against this, or to limit resource use on a shared machine, you may wish to set a hard limit on the maximum number of threads per `oxsii` or `boxsi` instance. This can be done by setting (in order of increasing precedence) the environment variable `OOMMF_THREADLIMIT`, the `thread_limit` value in the `config/platforms/` file for your platform, or the `oommf_thread_limit` value in the `config/options` file. (Note the precedence order is reversed compared to that for the default thread count.) If a limit is set then that value is displayed in the threads line of the `oommf.tcl +platform` output.

If NUMA support is provided on your platform (see below), then the following `oommf.tcl +platform` output line will indicate whether or not the build process will create NUMA-aware Oxs solvers.

After the thread and NUMA information, `oommf.tcl +platform` reports the directory that OOMMF will use to write temporary files. This directory is used, for example, to transfer magnetization data from the micromagnetic solvers to the **mmDisp** display module. You must have write access to this directory. It needs to have enough space to manage the dataflows of your simulations. It is also beneficial if this directory is local to the processors performing the calculations. If you don't like the OOMMF default, you may change it via the `path_directory_temporary` setting in the `config/platforms/` file for your platform. Or you can set the environment variable `OOMMF_TEMP`, which will override all other settings.

If any environment variables relevant to OOMMF are set, then `oommf.tcl +platform` will report these next, followed finally by any warnings about possible problems with your Tcl/Tk installation, such as if you are missing important header files.

If `oommf.tcl +platform` indicates problems with your Tcl/Tk installation, it may be easiest to re-install Tcl/Tk taking care to perform a conventional installation. OOMMF deals best with conventional Tcl/Tk installations. If you do not have the power to re-install an existing broken Tcl/Tk installation (perhaps you are not the sysadmin of your machine), you might still install your own copy of Tcl/Tk in your own user space. In that case, if your private Tcl/Tk installation makes use of shared libraries, take care that you do whatever is necessary on your platform to be sure that your private `tclsh` and `wish` find and use your private shared libraries instead of those from the system Tcl/Tk installation. This might involve setting an environment variable (such as LD_LIBRARY_PATH on Unix or PATH on Windows). If you use a private Tcl/Tk installation, you also want to be sure that there are no environment variables like TCL_LIBRARY or TK_LIBRARY that still refer to the system Tcl/Tk installation.

Additional Configuration Issues on Windows

A few other configurations should be checked on Windows platforms. First, note that absolute filenames on Windows makes use of the backslash (\) to separate directory names. On Unix and within Tcl the forward slash (/) is used to separate directory names in an absolute filename. In this manual we usually use the Tcl convention of forward slash as separator. In portions of the manual pertaining only to MS Windows we use the backslash as separator. There may be instructions in this manual which do not work exactly as written on Windows platforms. You may need to replace forward slashes with backward slashes in pathnames when working on Windows.

OOMMF software needs networking support that recognizes the host name `localhost`. It may be necessary to edit a file which records that `localhost` is a synonym for the loopback interface (127.0.0.1). If a file named `hosts` exists in your system area (for example, `C:\Windows\hosts`), be sure it includes an entry mapping 127.0.0.1 to `localhost`. If no `hosts` file exists, but a `hosts.sam` file exists, make a copy of `hosts.sam` with the name `hosts`, and edit the copy to have the `localhost` entry.

The directory that holds the `tclsh` and `wish` programs also holds several `*.dll` files that OOMMF software needs to find to run properly. Normally when the OOMMF bootstrap application or **mmLaunch** is used to launch OOMMF programs, they take care of making sure the necessary `*.dll` files can be found. As an additional measure, you might want to add the directory which holds the `tclsh` and `wish` programs to the list of directories stored in the PATH environment variable. All the directories in the PATH are searched for `*.dll` files needed when starting an executable.

2.2.4 Compiling and Linking

If you downloaded a distribution with pre-compiled executables, you may skip this section.

When building OOMMF software from source code, be sure the C++ compiler reported by `oommf.tcl +platform` is properly configured. In particular, if you are running on a Windows system, please read carefully the notes in Advanced Installation pertaining to your compiler.

The compiling and linking of the C++ portions of OOMMF software are guided by the application pimake ("Platform Independent Make") which is distributed as part of the OOMMF release. To begin building OOMMF software with `pimake`, first change your working directory to the root directory of the OOMMF distribution:

```
cd .../path/to/oommf
```

If you unpacked the new OOMMF release into a directory `oommf` which contained an earlier OOMMF release, use `pimake` to build the target `upgrade` to clear away any source code files which were in a former distribution but are not part of the latest distribution:

```
tclsh oommf.tcl pimake upgrade
```

Next, build the target `distclean` to clear away any old executables and object files which are left behind from the compilation of the previous distribution:

```
tclsh oommf.tcl pimake distclean
```

Next, to build all the OOMMF software, run `pimake` without specifying a target:

```
tclsh oommf.tcl pimake
```

On some platforms, you cannot successfully compile OOMMF software if there are OOMMF programs running. Check that all OOMMF programs have terminated (including those in the background) before trying to compile and link OOMMF.

When `pimake` calls on a compiler or other software development utility, the command line is printed, so that you may monitor the build process. Assuming a proper configuration for your platform, `pimake` should be able to compile and link all the OOMMF software without error. If `pimake` reports errors, please first consult Troubleshooting to see if a fix is already documented. If not, please send both the *complete* output from `pimake` and the output from `oommf.tcl +platform` to the OOMMF developers when you e-mail to ask for help.

2.2.5 Installing

The current OOMMF release does not support an installation procedure. For now, simply run the executables from the directories in which they were unpacked/built.

2.2.6 Using OOMMF Software

To start using OOMMF software, run the OOMMF bootstrap application. This may be launched from the command line interface:

```
tclsh oommf.tcl
```

If you prefer, you may launch the OOMMF bootstrap application `oommf.tcl` using whatever graphical "point and click" interface your operating system provides. By default, the OOMMF bootstrap application will start up a copy of the OOMMF application **mmLaunch** in a new window.

If you publish material created with the aid of OOMMF, please refer to Credits for citation information.

2.2.7 Reporting Problems

If you encounter problems when installing or using OOMMF, please report them to the OOMMF developers. The `oommf.tcl +platform` command has been designed in large part to help OOMMF developers debug installation problems, so **PLEASE** be sure to include the complete output from `oommf.tcl +platform` in your report. See also the section on troubleshooting for additional instructions.

2.3 Advanced Installation

The following sections provide instructions for some additional installation options.

2.3.1 Reducing Disk Space Usage

To delete the intermediate files created when building the OOMMF software from source code, use pimake to build the target `objclean` in the root directory of the OOMMF distribution.

```
tclsh oommf.tcl pimake objclean
```

Running your platform `strip` utility on the OOMMF executable files should also reduce their size somewhat.

2.3.2 Local Customizations

OOMMF software supports local customization of some of its features. The files controlling these options are `config/options.tcl` and `config/platforms/<platform>.tcl`, where "`<platform>`" is the identifier for the platform, e.g., `windows-x86_64` for Windows, `linux-x86_64` for Linux, and `darwin` for macOS. Although you may edit these files directly, it is better to create a subdirectory named `local` and create a file with the same name containing your changes. (Separating your changes from the distributed versions of these files makes it easier to port your changes to new versions of OOMMF.) For example, to change the settings in the `options.tcl` file, create the file `config/local/options.tcl`. To modify the platform file on Windows, create the file `config/platforms/local/windows-x86_64.tcl`.

The `options.tcl` and platform files are Tcl script files, and you may use any of the standard Tcl commands and constructs inside them. There are also OOMMF-specific commands available in these files, in particular `Oc_Option` for `options.tcl`, and `$config SetValue` for platform files.

The `options.tcl` file controls settings which are platform agnostic. For example, this file sets the default network port for the host service directory application, the location of **Oxs** and the width of curves in **mmGraph**. Editing instructions are provided in `options.tcl`. As an example, the setting

```
Oc_Option Add mmGraph Ow_GraphWin default_curve_width 1
```

in `options.tcl` can be overridden in `local/options.tcl` with

```
Oc_Option Add mmGraph Ow_GraphWin default_curve_width 2
```

if wider lines are desired. The `options.tcl` file also controls some build and optimization controls, as discussed next section.

The platform configuration files control settings that are platform dependent, such as the location for temporary files, compiler selection, and compiler and linker search paths. These files contain a block of code demarcated by the string `LOCAL CONFIGURATION` that lists end-user configuration options. These controls are all commented out (i.e., deactivated by a leading `#` character, which is the comment marker in Tcl), so you can copy this entire block into your `local/<platform>.tcl` file and uncomment only the options you want to change. For example, if you are using MacPorts[5] on macOS, then you may want to edit your `config/platforms/local/darwin.tcl` file to have the lines

```
## EXTERNAL PACKAGE SUPPORT:
## Extra include directories for compiling:
$config SetValue program_compiler_extra_include_dirs /opt/local/include
#
## Extra directories to search for libraries.
$config SetValue program_linker_extra_lib_dirs /opt/local/lib
```

to allow the compiler and linker to search in the MacPorts default install area.

There are a number of environment variables that can be set to override the values in the above configuration files, including

OOMMF_CPP: Two item list specifying the C++ compiler and the option to compile only (no link); e.g., for a Bourne compatible shell

```
export OOMMF_CPP=g++ -c
```

selects the GNU C++ compiler. To select the Visual C++ compiler at the Windows command prompt, use

[5] https://www.macports.org

```
set OOMMF_CPP=cl /c
```

Overrides the `<platform>.tcl` `program_compiler_c++_override` setting.

OOMMF_HOSTPORT: Host server listen port. Overrides the `options.tcl` "`Net_Host port`" value.

OOMMF_OUTDIR: Output file directory for **Oxs** runs; overrides default which is the directory holding the run `.mif` simulation specification file.

OOMMF_OXSLOGDIR: Directory for **Oxs** log files. Overrides `options.tcl` setting "`OxsLogs directory`".

OOMMF_TEMP: Directory for temporary files. Overrides auto-selection.

OOMMF_THREADS: Default number of execution threads for **Oxs** runs. Overrides `<platform>.tcl` `thread_count` setting.

Be aware that environment variable settings hold only for applications launched from the given shell instance, unless the value is set in the shell start-up configuration.

Where applicable, command line options override both configuration file and environment variable settings. For example, **Oxsii** and **Boxsi** support the `-threads` option that specifies the number of execution threads.

The OOMMF installation can also be customized by the installation of third party and experimental extensions to the OOMMF eXtensible Solver (Oxs). These are maintained using the **oxspkg** package management tool.

2.3.3 Optimization

In the interest of successful compilation of a usable software package "out of the box," the default configuration for OOMMF does not stretch to maximize optimization. However, the configuration files `config/options.tcl` and `config/platforms/<platform>.tcl` include optimization controls. In the former,

```
Oc_Option Add * Platform optlevel 2
```

selects a relatively high but robust level of optimization, equivalent typically to a compiler option of "-O2" or similar. You may try increasing `optlevel` to 3. If you are familiar with your compiler's command line options, then for finer control you can experiment with the `<platform>.tcl` file options

```
program_compiler_c++_remove_flags
```

and

```
program_compiler_c++_add_flags
```

The remove flags control selects options to be removed from the compiler command line. The setting value should be a Tcl list of regular expressions. (Refer to the Tcl documentation[6] on the regexp command for details.) The add flags value is a list of values to append to the compile command (applied after the remove flags request, if any). An extended discussion of these controls can be found in the "Configuration Files" section of the *OOMMF Programming Manual*[7].

2.3.4 Parallelization

The OOMMF Oxs 3D solvers (`oxsii` and `boxsi`) can be built thread-enabled to allow parallel processing on multi-processor/multi-core machines. In order to build and run a parallel version of OOMMF, you must have a thread-enabled version of Tcl. Most standard binary releases of Tcl today are thread-enabled, so OOMMF releases that include pre-built executables are built thread-enabled. If you build OOMMF from source, then by default OOMMF will be built thread-enabled if your Tcl is thread-enabled. As explained earlier, you can check thread build status with the `tclsh oommf.tcl +platform` command. If you want to force a non-threaded build of OOMMF, then edit the `config/platforms/` file for your platform. In the section labeled `LOCAL CONFIGURATION`, you will find a line that looks like

```
# $config SetValue oommf_threads 0
```

Uncomment this line (i.e., remove the leading '`#`' character) to force a non-threaded build. Then run

```
tclsh oommf.tcl pimake distclean
tclsh oommf.tcl pimake
```

from the OOMMF root directory to create a fresh build.

You can use the `tclsh oommf.tcl +platform` command to see the default number of compute threads that will be run by the Oxs 3D solver programs `oxsii` and `boxsi`. You can modify the default as explained in the Check Your Platform Configuration and Local Customizations sections, or you can override the default at run time via the command line option `-threads` to `oxsii` and `boxsi`.

Some multi-processor machines have a non-uniform memory architecture (NUMA), which means that although each processor can access all of system memory, some parts of memory can be accessed faster than others. Typically this is accomplished by dividing the system memory and processors into "nodes." Memory accesses within a node are faster than accesses between nodes, and depending on the architecture access latency and bandwidth may be different between different node pairs. Examples of machines with NUMA include some multi-processor AMD Opteron and Intel Xeon boxes.

[6] https://www.tcl-lang.org/doc/
[7] https://math.nist.gov/oommf/doc/

Computer programs such as OOMMF can run on NUMA machines without making any special allowances for the memory architecture. However, a program that is written to take advantage of the faster local (intra-node) memory accesses can sometimes run significantly faster. OOMMF contains NUMA-aware code, but this code is highly operating system specific. At present, OOMMF can be built with NUMA support only on Linux (32- and 64-bit) systems. To do this, you must install the operating system NUMA support packages "numactl" and "numactl-devel". The names may vary somewhat between Linux distributions, but the first typically includes the executable `numactl` and the second includes the header file `numa.h`. Once the numactl package is installed, you can run the command

```
numactl --hardware
```

to get an overview of the memory architecture on your machine. If this shows you have only one node, then there is no advantage to making a NUMA-aware build of OOMMF.

The next step is to edit the `config/platforms` for your platform. For example, on a 64-bit Linux box this file is `config/platforms/linux-x86_64.tcl`. In the section labeled `LOCAL CONFIGURATION`, find the line

```
# $config SetValue use_numa 1
```

Edit this to remove the leading '`#`' character. Alternatively (and, actually, preferably), create a `local` subdirectory and make a local configuration file with the same platform name; e.g., `config/platforms/local/linux-x86_64.tcl` on a 64-bit Linux machine. Add the line

```
$config SetValue use_numa 1
```

to this file. (The advantage of using a `config/platforms/local` file is that you can make changes without modifying the original OOMMF source code, which makes it easier to port your local changes to future releases of OOMMF.) If this is done correctly, then the command '`tclsh oommf.tcl +platform`' will show that NUMA support is enabled. Then simply run '`tclsh oommf.tcl pimake distclean`' and '`tclsh oommf.tcl pimake`' from the OOMMF root directory to build a NUMA-aware version of OOMMF.

To activate the NUMA-aware code, you must specify the `-numanodes` option on the `oxsii/boxsi` command line, or set the environment variable `OOMMF_NUMANODES`. Check the Oxs documention for details.

2.3.5 Managing OOMMF Platform Names

OOMMF software classifies computing platforms into different types using the scripts in the directory `config/names` relative to the root directory of the OOMMF distribution. Each type of computing platform is assigned a unique name. These names are used as directory names and in index and configuration files so that a single OOMMF installation may contain platform-dependent sections for many different types of computing platforms.

To learn what name OOMMF software uses to refer to your computing platform, run

```
tclsh oommf.tcl +platform
```

in the OOMMF root directory.

Changing the name OOMMF assigns to your platform

First, use pimake to build the target `distclean` to clear away any compiled executables built using the old platform name.

```
tclsh oommf.tcl pimake distclean
```

Then, to change the name OOMMF software uses to describe your platform from `foo` to `bar`, simply rename the file

`config/names/foo.tcl` to `config/names/bar.tcl`

and

`config/platforms/foo.tcl` to `config/platforms/bar.tcl`.

After renaming your platform type, you should recompile your executables using the new platform name.

Adding a new platform type

If `oommf.tcl +platform` reports the platform name `unknown`, then none of the scripts in `config/names/` recognizes your platform type. As an example, to add the platform name `foo` to OOMMF's vocabulary of platform names, create the file `config/names/foo.tcl`. The simplest way to proceed is to copy an existing file in the directory `config/names` and edit it to recognize your platform.

The files in `config/names` include Tcl code like this:

```
Oc_Config New _ \
  [string tolower [file rootname [file tail [info script]]]] {
    # In this block place the body of a Tcl proc which returns 1
    # if the machine on which the proc is executed is of the
    # platform type identified by this file, and which returns 0
    # otherwise.
    #
    # The usual Tcl language mechanism for discovering details
    # about the machine on which the proc is running is to
    # consult the global Tcl variable 'tcl_platform'.  See the
    # existing files for examples, or contact the OOMMF
    # developers for further assistance.
}
```

After creating the new platform name file `config/names/foo.tcl`, you also need to create a new platform file `config/platforms/foo.tcl`. A reasonable starting point is to copy the file `config/platforms/unknown.tcl` for editing. Contact the OOMMF developers for assistance.

Please consider contributing your new platform recognition and configuration files to the OOMMF developers for inclusion in future releases of OOMMF software.

Resolving platform name conflicts

If the script `oommf.tcl +platform` reports "Multiple platform names are compatible with your computer", then there are multiple files in the directory `config/names/` that return 1 when run on your computer. For each compatible platform name reported, edit the corresponding file in `config/names/` so that only one of them returns 1. Experimenting using `tclsh` to probe the Tcl variable `tcl_platform` should assist you in this task. If that fails, you can explicitly assign a platform type corresponding to your computing platform by matching its hostname. For example, if your machine's host name is `foo.bar.net`:

```
Oc_Config New _ \
  [string tolower [file rootname [file tail [info script]]]] {
    if {[string match foo.bar.net [info hostname]]} {
        return 1
    }
    # Continue with other tests...
}
```

Contact the OOMMF developers if you need further assistance.

2.4 Platform Specific Installation Issues

The installation procedure discussed in the previous sections applies to all platforms (Unix, Windows, macOS). There are, however, some details which pertain only to a particular platform. These issues are discussed below.

2.4.1 Unix Configuration

Missing Tcl/Tk files

The basic installation procedure should be sufficient to install OOMMF on most Unix systems. Sometimes, however, the build will fail due to missing Tcl header files (`tcl.h`, `tk.h`) or libraries (e.g., `libtcl.so`, `libtk.so`). This problem can usually be solved by installing a "development" version of Tcl/Tk, which may be found on the operating system installation disks, or may be available from the system vendor. There are also binary releases of Tcl/Tk for a number of systems available from ActiveState, under the name ActiveTcl[8]. Alternatively, one may download the sources for Tcl and Tk from the Tcl Developer Xchange[9], and build and install Tcl/Tk from source. The Tcl/Tk build follows the usual Unix `configure`, `make`, `make install` build convention.

[8] https://www.activestate.com/products/tcl/
[9] https://www.tcl-lang.org/

Compiler Optimization Options

On most systems, OOMMF builds by default with relatively unaggressive compiler optimization options. As discussed earlier (Optimization), you may edit the appropriate `oommf/config/platforms/` file to change the default compilation options. However, on some common systems (e.g., Linux, some BSD variants) OOMMF will try to deduce the hardware architecture (i.e., the CPU subtype, such as Pentium 3 vs. Pentium 4) and apply architecture-specific options to the compile commands. This is probably what you want if OOMMF is to be run only on the system on which it was built, or if it is run on a homogeneous cluster. If, instead, you intend to run OOMMF on a heterogeneous cluster you may need to restrict the compiler options to those supported across your target machines. In that case, open the appropriate configuration file in the `oommf/config/platforms/` directory, and look for the lines

```
# You can override the GuessCPU results by directly setting or
# unsetting the cpuopts variable, e.g.,
#
#    set cpuopts [list -march=skylake]
# or
#    unset cpuopts
#
```

Uncomment either the "unset cpuopts" line to make a generic build, or else edit the "set cpuopts" line to an appropriate common-denominator architecture and uncomment that line.

In a similar vein, some compilers support a "-fast" switch, which usually creates an architecture-specific executable. The same considerations apply in this case.

An advanced alternative would be to define separate OOMMF "platforms" for each CPU subtype in your cluster. At a minimum, this would involve creating separate platform name files in `oommf/config/names/` for each subtype, and then making copies of the appropriate `oommf/config/platforms` file for each new platform. The platform name files would have to be written so as to reliably detect the CPU subtype on each machine. See "Managing OOMMF platform names" for details on creating platform name files.

Portland Group pgCC compiler on Linux

The platform build scripts for Linux, `oommf/config/platforms/lintel.tcl` (32-bit) and `oommf/config/platforms/linux-x86_64.tcl` (64-bit) contain sections supporting the Portland Group pgCC compiler. Non-threaded builds of OOMMF using this compiler run fine, but threaded builds segfault when running `Oxsii/Boxsi`. The source of this problem is not known at this time.

2.4.2 macOS Configuration

The build procedure for macOS is the same as for Unix. The platform name is "darwin". If the platform configuration check does not find a C++ compiler, then you will have to install one. One option is the Xcode command line developer tools provided by Apple. You can install these from a Terminal window via the command

```
xcode-select --install
```

You should run this command even if you install the full Xcode IDE. Refer to your system documentation for details.

2.4.3 Microsoft Windows Options

This section lists installation options for Microsoft Windows.

Using Microsoft Visual C++

If you are building OOMMF software from source using the Microsoft Visual C++ command line compiler, `cl.exe`, it is necessary to set up the path and some environment variables before running the compiler. There is a batch file distributed with Visual C++ that you can run to do this. The name of the file varies between Visual C++ releases, but for example may be `vcvarsall.bat` or `setenv.cmd`. For 64-bit builds you may need to include the "amd64" option on the batch file command line. You may want to set up your system so this batch file gets run automatically when you open a command window. See your compiler and system documentation for details.

Using MinGW g++

Both 32-bit and 64-bit builds are supported using the MinGW ports of g++. (The 32-bit and 64-bit versions of g++ are separate downloads.) Use a standard Windows Tcl/Tk, such as the ActiveTcl[10] release from ActiveState. You will also need to edit the appropriate platform file to select g++ as the compiler. If you are using a 32-bit Tcl/Tk and g++, then the platform file is `oommf\config\platforms\wintel.tcl`. For 64-bit Tcl/Tk and g++ the platform file is `oommf\config\platforms\windows-x86_64.tcl`.

Using the Cygwin toolkit

The Cygwin Project[11] is a free port of the GNU development environment to Windows, which includes the GNU C++ compiler g++ and X11. To build OOMMF within the Cygwin environment, start up a Cygwin or Cygwin64 shell and follow the usual Unix build procedure. The platform name will be `cygtel` or `cygwin-x86_64`, according to whether you are running

[10] https://www.activestate.com/products/tcl/
[11] https://www.cygwin.com/

a 32- or 64-bit Cygwin `tclsh`, respectively. The resulting OOMMF build requires the Cygwin environment to run, so it will need to be launched from a Cygwin shell. Moreover, OOMMF on Cygwin uses X11 as the windowing interface, so you will need to have the Cygwin port of X11 installed; typically OOMMF will be started from an X11 xterm or equivalent. Of course, you will also need the Tcl and Tk packages installed (called `tcl` and `tcl-tk`, respectively, by the Cygwin package manager). To build OOMMF from source you will need the `gcc-g++`, `tcl-devel`, and `tcl-tk-devel` packages and dependencies.

If you get errors saying a child process couldn't be forked (typically with either "resource temporarily unavailable" or "Loaded to different address" error messages), then follow this procedure:

1. Exit all Cygwin processes
2. Use Windows Explorer or a Windows command shell to launch `c:\cygwin\bin\ash.exe`
3. Run `/bin/rebaseall` inside the ash shell.

Additional information on this problem can be found in the Cygwin documentation.

The Cygwin versions of Tcl/Tk prior to 8.6 were not threaded, so OOMMF built with Tcl/Tk 8.5 and older will likewise not be threaded. This limitation is removed with the Cygwin Tcl/Tk 8.6 release.

Setting the TCL_LIBRARY environment variable

If you encounter difficulties during OOMMF start up, you may need to set the environment variable TCL_LIBRARY. (NOTE: This is almost never necessary!)

Bring up the Control Panel (e.g., by selecting **Settings|Control Panel** off the Start menu), and select **System**. Go to the **Environment** tab, and enter TCL_LIBRARY as the Variable, and the name of the directory containing `init.tcl` for the Value, e.g.,

```
%SystemDrive%\Program Files\Tcl\lib\tcl8.0
```

Click **Set** and **OK** to finish.

Chapter 3

Quick Start: Example OOMMF Session

STEP 1: Start up the mmLaunch window.

- At the command prompt, when you are in the OOMMF root directory, type

  `tclsh oommf.tcl`

  (The name of the Tcl shell, rendered here as `tclsh`, may vary between systems.) Alternatively, you may launch `oommf.tcl` using whatever "point and click" interface is provided by your operating system.
- This will bring up a small window labeled **mmLaunch**. It will come up in background mode, so you will get another prompt in your original window, even before the **mmLaunch** window appears.

STEP 2: Gain access to other useful windows.

- The **mmLaunch** window is divided into two columns. The right column provides a list of running applications. This will normally be empty when you first start **mmLaunch**. The left column, labeled "Programs," provides a collection of buttons that launch applications when clicked:
  - **mmArchive:** auto-saves tabular and field data files
  - **mmDataTable:** displays current values of tabular (scalar) outputs
  - **mmDisp:** displays scalar and vector fields
  - **mmGraph:** makes x-y plots
  - **mmProbEd:** problem editor for **mmSolve2D** or **Oxsii**
  - **mmSolve2D:** 2D solver interactive interface
  - **Oxsii:** 3D solver interactive interface
- Click on **mmDisp**, **mmGraph**, and/or **mmDataTable**, depending on what form of output you want to view. Use **mmArchive** to save data to disk.

STEP 3a: Run a 2D problem.

Load problem:

- In the **mmLaunch** window, click on the **mmProbEd** button.
- In the mmProbEd window, make menu selection **File|Open...** An **Open File** dialog window will appear. In this window:
  - Double click in the **Path** subwindow to change directories. Several sample problems can be found in the directory `oommf/app/mmpe/examples`.
  - To load a problem, double click on a `*.mif` file (e.g., prob1.mif) from the list above the **Filter:** subwindow.
- Modify the problem as desired by clicking on buttons from the main **mmProbEd** window (e.g., **Material Parameters**), and fill out the pop-up forms. A completely new problem may be defined this way.
- If desired, the defined problem may be stored to disk via the **File|Save as...** menu selection. The 2D solver **mmSolve2D** reads problem definitions directly from **mmProbEd**, but **Oxsii** requires file input.

Initialize solver:

- In the **mmLaunch** window, click on the **mmSolve2D** button to launch an instance of the program mmSolve2D.
- Wait for the new solver instance to appear in the **Running Applications** column in the **mmLaunch** window.
- Check the box next to the **mmSolve2D** entry in the **Running Applications** column. A window containing an **mmSolve2D** interface will appear.
- In the **mmSolve2D** window:
  - Check **Problem Description** under **Inputs**.
  - Check **mmProbEd** under **Source Threads**.
  - Click **LoadProblem**.
  - A status line will indicate the problem is loading.
  - When the problem is fully loaded, more buttons appear.
  - Check **Scheduled Outputs**.
  - For each desired output (**TotalField**, **Magnetization**, and/or **DataTable**), specify the frequency of update:
    1. Check desired output. This will exhibit the possible output destinations under the Destination Threads heading. Output applications such as **mmDisp**, **mmGraph**, and/or **mmDataTable** must be running to appear in this list.
    2. Check the box next to the desired Destination Thread. This will exhibit Schedule options.
    3. Choose a schedule:
       * **Iteration:** fill in number and check the box.

    - **ControlPoint:** fill in number and check the box.
    - **Interactive:** whenever you click corresponding Interactive output button.

Start calculation:

- In the **mmSolve2D** window, start the calculation with **Run** (which runs until problem completion) or **Relax** (which runs until the next control point is reached).
- If you requested mmDataTable output, check the boxes for the desired quantities on the **mmDataTable** window under the **Data** menu, so that they appear and are updated as requested in your schedule.
- Similarly, check the box for the desired X, Y1, and Y2 quantities on the **mmGraph** window(s) under the **X**, **Y1** and **Y2** menus.

Save and/or display results:

- Vector field data (magnetization and effective field) may be viewed using **mmDisp**. You can manually save data to disk using the **File|Save as...** menu option in **mmDisp**, or you can send scheduled output to **mmArchive** for automatic storage. For example, to save the magnetization state at the end of each control point, start up an instance of **mmArchive** and select the **ControlPoint** check box for **mmArchive** on the **Magnetization** schedule in the solver. This may be done before starting the calculation. (Control points are points in the simulation where the applied field is stepped. These are typically equilibrium states, but depending on the input `*.mif` file, may be triggered by elapsed simulation time or iteration count.)
- Tabular data may be saved by sending scheduled output from the solver to **mmArchive**, which automatically saves all the data it receives. Alternatively, **mmGraph** can be used to save a subset of the data: schedule output to **mmGraph** as desired, and use either the interactive or automated save functionality of **mmGraph**. You can set up the solver data scheduling before the calculation is started, but you must wait for the first data point to configure **mmGraph** before saving any data. As a workaround, you may configure **mmGraph** by sending it the initial solver state interactively, and then use the **Options|clear Data** menu item in **mmGraph** to remove the initializing data point. If you want to inspect explict numeric values, use **mmDataTable**, which displays single sets of values in a tabular format. **mmDataTable** has no data save functionality.

Midcourse control:

- In the **mmSolve2D** window, buttons can stop and restart the calculation:
    - **Reset:** Return to beginning of problem.
    - **LoadProblem:** Restart with a new problem.
    - **Run:** Apply a sequence of fields until all complete.
    - **Relax:** Run the ODE at the current applied field until the next control point is reached.

  - **Pause:** Click anytime to stop the solver. Continue simulation from paused point with **Run** or **Relax**.
  - **Field−:** Apply the previous field again.
  - **Field+:** Apply the next field in the list.
- Output options can be changed and new output windows opened.
- When the stopping criteria for the final control point are reached, **mmSolve2D** will pause to allow the user to interactively output final results.

STEP 3b: Run a 3D problem.

Launch solver:

- In the **mmLaunch** window, click on the **Oxsii** button to launch an instance of the program **Oxsii**.
- Wait for the new solver instance to appear in the **Running Applications** column in the **mmLaunch** window.
- Check the box next to the **Oxsii** entry in the **Running Applications** column. A window containing an **Oxsii** interface will appear.

Load problem:

- In the **Oxsii** window, select the **File|Load...** menu option. A **Load Problem** dialog box will appear. On this window:
  - Double click in the **Path** subwindow to change directories. Numerous sample problems can be found in the directory `oommf/app/oxs/examples`.
  - To load a problem, double click on a `*.mif` file (e.g., stdprob1.mif) from the list above the **Filter:** subwindow.

  The native input format for the 3D solver is the MIF 2 format, which must be composed by hand using a plain text editor. (See the Oxs_Ext Child Class documentation for additional details.) However, MIF 1.1 (i.e., 2D problem) files are readable by **Oxsii**, or may be converted to the MIF 2.1 format using the command line tool **mifconvert**. **mmProbEd** also supports an extension to the MIF 1.1 format, namely MIF 1.2, which provides limited 3D functionality. MIF 1.2 files may also be read directly by **Oxsii**. Either way, to run in **Oxsii** a problem created by **mmProbEd**, the problem must first be saved to disk via the **File|Save as...** menu option in **mmProbEd**.
- The status line in the **Oxsii** interface window will indicate the problem is loading.
- When the problem is fully loaded, the status line will show "Pause", and the top row of buttons (**Reload**, **Reset**, ...) will become active. Also, the Output list will fill with available outputs.
- Set up scheduled outputs. For each desired output
  1. Select the source from the Output list.
  2. Select the receiver from the Destination list.

3. Specify the frequency of update:
   - **Step:** fill in number and check the box.
   - **Stage:** fill in number and check the box.
   - **Done:** produces output when problem completes.

   You can also transmit data interactively by clicking on the **Send**.

   The items in the Output list will vary depending on the problem that was loaded. The items in the Destination list reflect the OOMMF data display and archiving programs currently running.

Start calculation:

- In the **Oxsii** window, start the calculation with **Run**, **Relax**, or **Step**.
- If you requested mmDataTable output, check the boxes for the desired quantities on the **mmDataTable** window under the **Data** menu, so that they appear and are updated as requested in your schedule.
- Similarly, check the box for the desired X, Y1, and Y2 quantities on the **mmGraph** window(s) under the **X**, **Y1** and **Y2** menus.

Save and/or display results:

- Vector field data (magnetization and fields) may be viewed using **mmDisp**. You can manually save data to disk using the **File|Save as...** menu option in **mmDisp**, or you can send scheduled output to **mmArchive** for automatic storage. For example, to save the magnetization state at the end of each problem stage, start up an instance of **mmArchive** and select the **Stage** check box for the `Magnetization` output, `mmArchive` destination pair. (Stages denote points in the simulation where some significant event occurs, such as when an equilibrium is reached or some preset simulation time index is met. These criteria are set by the input MIF file.)
- Tabular data may be saved by sending scheduled output from the solver to **mmArchive**, which automatically saves all the data it receives. Alternatively, **mmGraph** can be used to save a subset of the data: schedule output to **mmGraph** as desired, and use either the interactive or automated save functionality of **mmGraph**. You can set up the solver data scheduling before the calculation is started, but you must wait for the first data point to configure **mmGraph** before saving any data. As a workaround, you may configure **mmGraph** by sending it the initial solver state interactively, and then use the **Options|clear Data** menu item in **mmGraph** to remove the initializing data point. If you want to inspect explict numeric values, use **mmDataTable**, which displays single sets of values in a tabular format. **mmDataTable** has no data save functionality.

Midcourse control:

- In the **Oxsii** window, buttons can stop and restart the calculation:
  - **Reload:** Reload the same file from disk.
  - **Reset:** Return to problem start.

  - **Run:** Step through all stages until complete.
  - **Relax:** Run until the current stage termination criteria are met.
  - **Step:** Do one solver iteration and then pause.
  - **Pause:** Click anytime to stop the solver. Continue simulation from paused point with **Run**, **Relax** or **Step**.
  - **Stage:** Interactively change the current stage index by either typing the desired stage number (counting from 0) into the **Stage** entry box or by moving the associated slider.
- Output options can be changed and new output windows opened. The **Send** button in the **Oxsii** Schedule subwindow is used to interactively send output to the selected Output + Destination pair.
- When the stage termination (stopping) criteria of the final stage are met, **Oxsii** will pause to allow the user to interactively output final results via the **Send** button. (This differs from the Boxsi batch interface which terminates automatically when the final stage is complete.)

STEP 4: Exit OOMMF.

- Individual OOMMF applications can be terminated by selecting the **File|Exit** menu item from their interface window.
- Selecting **File|Exit** on the **mmLaunch** window will close the **mmLaunch** window, and also the interface windows for any **mmArchive**, **mmSolve2D**, and **Oxsii** applications. However, those applications will continue to run in the background, and their interfaces may be re-displayed by starting a new **mmLaunch** instance.
- To kill all OOMMF applications, select the **File|Exit All OOMMF** option from the **mmLaunch** menu bar.

Chapter 4

OOMMF Architecture Overview

Before describing each of the applications which comprise the OOMMF software, it is helpful to understand how these applications work together. OOMMF is not structured as a single program. Instead it is a collection of programs, each specializing in some task needed as part of a micromagnetic simulation system. An advantage of this modular architecture is that each program may be improved or even replaced without a need to redesign the entire system.

The OOMMF programs work together by providing services to one another. The programs communicate using localhost Internet (TCP/IP) connections. When two OOMMF applications are in the relationship that one is requesting a service from the other, it is convenient to introduce some clarifying terminology. Let us refer to the application that is providing a service as the "server application" and the application requesting the service as the "client application." Note that a single application can be both a server application in one service relationship and a client application in another service relationship.

Each server application provides its service on a particular Internet port, and needs to inform potential client applications how to obtain its service. Each client application needs to be able to look up possible providers of the service it needs. The intermediary which brings server applications and client applications together is another application called the "account service directory." Each account service directory keeps track of all the services provided by OOMMF server applications running under its user account on its host and the corresponding Internet ports at which those services may be obtained. OOMMF server applications register their services with the corresponding account service directory application. OOMMF client applications look up service providers running under a particular user ID in the corresponding account server directory application.

The account service directory applications simplify the problem of matching servers and clients, but they do not completely solve it. OOMMF applications still need a mechanism to find out how to obtain the service of the account service directory! Another application, called the "host service directory" serves this function. Its sole purpose is to tell OOMMF applications where to obtain the services of account service directories on that host. It provides this service on a "well-known" port that is configured into the OOMMF software.

By default, this is port 15136. OOMMF software can be customized to use a different port number.

These service directory applications are vitally important to the operation of the total OOMMF micromagnetic simulation system. However, it would be easy to overlook them. They act entirely behind the scenes without a user interface window. Furthermore, they are usually not launched directly by the user. (The notable exception involves **launchhost**, which is used when multiple host servers are needed to isolate groups of OOMMF applications running on one machine.) When any server application needs to register its service, if it finds that these service directory applications are not running, it launches new copies of them. In this way the user can be sure that if any OOMMF server application is running, then so are the service directory applications needed to direct clients to its service. After all server applications terminate, and there are no longer any services registered with a service directory application, it terminates as well. Similarly, when all service directory applications terminate, the host service directory application exits. The command line utility **pidinfo** can be used to check the current status of the host and account service directory applications.

In the sections which follow, the OOMMF applications are described in terms of the services they provide and the services they require.

Chapter 5

Command Line Launching

Some of the OOMMF applications are platform-independent Tcl scripts. Some of them are Tcl scripts that require special platform-dependent interpreters. Others are platform-dependent, compiled C++ applications. It is possible that some of them will change status in later releases of OOMMF. Each of these types of application requires a different command line for launching. Rather than require all OOMMF users to manage this complexity, we provide a pair of programs that provide simplified interfaces for launching OOMMF applications.

The first of these is used to launch OOMMF applications from the command line. Because its function is only to start another program, we refer to this program as the "bootstrap application." The bootstrap application is the Tcl script `oommf.tcl`. In its simplest usage, it takes a single argument on the command line, the name of the application to launch. For example, to launch **mmGraph**, the command line is:

```
tclsh oommf.tcl mmGraph
```

The search for an application matching the name is case-insensitive. (Here, as elsewhere in this document, the current working directory is assumed to be the OOMMF root directory. For other cases, adjust the pathname to `oommf.tcl` as appropriate.) As discussed earlier, the name of the Tcl shell, rendered here as `tclsh`, may vary between systems.

If no command line arguments are passed to the bootstrap application, by default it will launch the application **mmLaunch**.

Any command line arguments to the bootstrap application that begin with the character '`+`' modify its behavior. For a summary of all command line options recognized by the bootstrap application, run:

```
tclsh oommf.tcl +help
```

The command line arguments `+bg` and `+fg` control how the bootstrap behaves after launching the requested application. It can exit immediately after launching the requested application in background mode (`+bg`), or it can block until the launched application exits (`+fg`). Each application registers within the OOMMF system whether it prefers to be launched in foreground or background mode. If neither option is requested on the command line, the bootstrap launches the requested application in its preferred mode.

The first command line argument that does not begin with the character `+` is interpreted as a specification of which application should be launched. As described above, this is usually the simple name of an application. When a particular version of an application is required, though, the bootstrap allows the user to include that requirement as part of the specification. For example:

```
tclsh oommf.tcl "mmGraph 1.1"
```

will guarantee that the instance of the application mmGraph it launches is of at least version 1.1. If no copy of mmGraph satisfying the version requirement can be found, an error is reported.

The rest of the command line arguments that are not recognized by the bootstrap are passed along as arguments to the application the bootstrap launches. Since the bootstrap recognizes command line arguments that begin with `+` and most other applications recognize command line arguments that begin with `-`, confusion about which options are provided to which programs can be avoided. For example,

```
tclsh oommf.tcl +help mmGraph
```

prints out help information about the bootstrap and exits without launching mmGraph. However,

```
tclsh oommf.tcl mmGraph -help
```

launches mmGraph with the command line argument `-help`. mmGraph then displays its own help message.

Most OOMMF applications accept the standard options listed below. Some of the OOMMF applications accept additional arguments when launched from the command line, as documented in the corresponding sections of this manual. The `-help` command line option can also be used to view the complete list of available options. When an option argument is specified as `<0|1>`, `0` typically means off, no or disable, and `1` means on, yes or enable.

-console Display a console widget in which Tcl commands may be interactively typed into the application. Useful for debugging.

-cwd directory Set the current working directory of the application.

-help Display a help message and exit.

-nickname <name> Associates the specified *name* as a nickname for the process. The string *name* should contain at least one non-numeric character. Nicknames can also be set at launch time via the `Destination` command (Sec. 17.3.2) in `MIF 2.x` files, or after a process is running via the **nickname** command line application. Nicknames are used by the MIF 2.x `Destination` command to associate `Oxs` output streams with particular application instances. Multiple `-nickname` options may be used to set multiple nicknames. (Technical detail: Nickname functionality is only available to processes that connect to an account server.)

-tk <0|1> Disable or enable Tk. Tk must be enabled for an application to display graphical widgets. However, when Tk is enabled on Unix platforms the application is dependent on an X Windows server. If the X Windows server dies, it will kill the application. Long-running applications that do not inherently use display widgets support disabling of Tk with `-tk 0`. Other applications that must use display widgets are unable to run with the option `-tk 0`. To run applications that require `-tk 1` on a Unix system with no display, one might use Xvfb[1] .

-version Display the version of the application and exit.

In addition, those applications which enable Tk accept additional Tk options, such as `-display`. See the Tk documentation for details.

The bootstrap application should be infrequently used by most users. The application **mmLaunch** provides a more convenient graphical interface for launching applications. The main uses for the bootstrap application are launching **mmLaunch**, launching **pimake**, launching programs which make up the OOMMF Batch System and other programs that are inherently command line driven, and in circumstances where the user wishes to precisely control the command line arguments passed to an OOMMF application or the environment in which an OOMMF application runs.

Platform Issues

On most Unix platforms, if `oommf.tcl` is marked executable, it may be run directly, i.e., without specifying `tclsh`. This works because the first few lines of the `oommf.tcl` Tcl script are:

```
#!/bin/sh
# \
exec tclsh "$0" ${1+"$@"}
```

When run, the first `tclsh` on the execution path is invoked to interpret the `oommf.tcl` script. If the Tcl shell program cannot be invoked by the name `tclsh` on your computer, edit the first lines of `oommf.tcl` to use the correct name. Better still, use symbolic links or some other means to make the Tcl shell program available by the name `tclsh`. The latter solution will not be undone by file overwrites from OOMMF upgrades.

If in addition, the directory `.../path/to/oommf` is in the execution path, the command line can be as simple as:

```
oommf.tcl appName
```

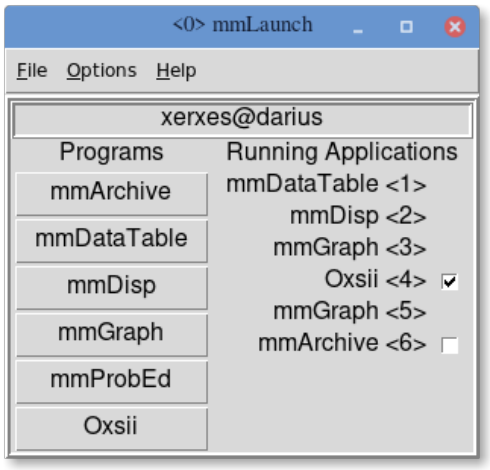
from any working directory.

[1] https://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml

On Windows platforms, because `oommf.tcl` has the file extension `.tcl`, it is normally associated by Windows with the `wish` interpreter. The `oommf.tcl` script has been specially written so that either `tclsh` or `wish` is a suitable interpreter. This means that simply double-clicking on an icon associated with the file `oommf.tcl` (say, in Windows Explorer) will launch the bootstrap application with no arguments. This will result in the default behavior of launching the application **mmLaunch**, which is suitable for launching other OOMMF applications. (If this doesn't work, refer back to the Windows Options.)

Chapter 6

OOMMF Launcher/Control Interface: mmLaunch



Overview

The application **mmLaunch** launches, monitors, and controls other OOMMF applications. It is the OOMMF application which is most closely connected to the account service directory and host service directory applications that run behind the scenes. It also provides user interfaces to any applications, notably **Oxsii** and **mmSolve2D**, that do not have their own user interface window.

Launching

mmLaunch should be launched using the bootstrap application. The command line is

```
tclsh oommf.tcl mmLaunch [standard options]
```

Controls

The **mmLaunch** interface consists of two columns. The **Programs** column contains buttons labeled with the names of OOMMF applications that may be launched under the account managed by this account service directory. Clicking on one of these buttons launches the corresponding application. Only one click is needed, though there will be some delay before the launched application displays a window to the user. Multiple clicks will launch multiple copies of the application.

The second column, labeled **Running Applications**, contains a list of OOMMF applications that are registered with the account server. If the account server is not already running, then **mmLaunch** will start it, in which case there may be a slight delay. Each entry in the **Running Applications** list includes both the application name and an ID number by which multiple copies of the same application may be distinguished. This ID number is also displayed in the title bar of the corresponding application's user interface window. When an application exits, its entry is automatically removed from the **Running Applications** list.

Any of the running applications that do not provide their own interface window will be displayed in **mmLaunch** with a checkbutton. The checkbutton toggles the display of an interface which **mmLaunch** provides on behalf of that application. The only OOMMF applications currently using this service are the 3D solvers **Oxsii** and **Boxsi**, the 2D solvers **mmSolve2D and batchsolve**, and the archive application **mmArchive**. These interfaces are described in the documentation for the corresponding applications.

The menu selection **File|Exit** terminates the **mmLaunch** application, and the **File|Exit All OOMMF** selection terminates all applications in the **Running Applications** list, and then exits **mmLaunch**. The **Options** menu has one entry, the **Click to raise** toggle. If this is enabled then a left mouse click on any of the **Running Applications** will raise that application to the front of the screen display stack, while a right or `Shift`+left mouse click will lower it to the back. For tabbing keyboard traversal the corresponding keystrokes are `space` and `Shift+space`. The menu **Help** provides the usual help facilities.

Chapter 7

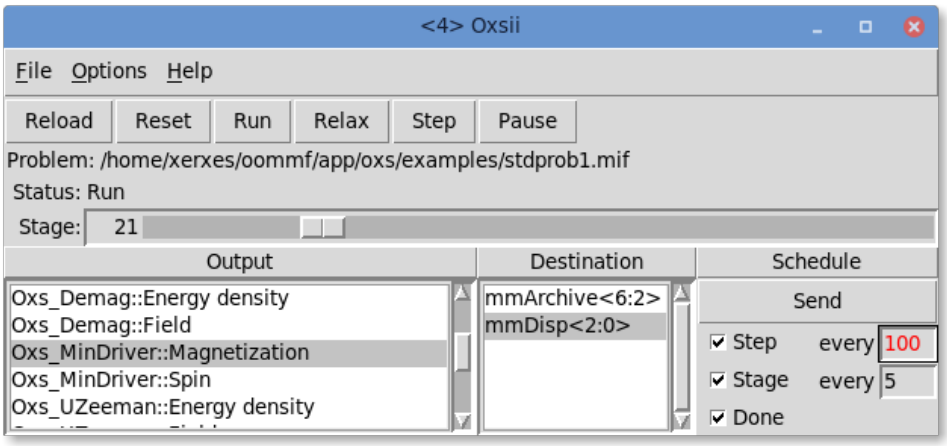
OOMMF eXtensible Solver

The Oxs (OOMMF eXtensible Solver) is an extensible micromagnetic computation engine capable of solving problems defined on three-dimensional grids of rectangular cells holding three-dimensional spins. There are two interfaces provided to Oxs: the interactive interface Oxsii intended to be controlled primarily through a graphical user interface, and the batch mode Boxsi, which has extended command line controls making it suitable for use in shell scripts.

Problem definition for Oxs is accomplished using input files in the MIF 2 format. This is an extensible format; the standard OOMMF modules are documented below. Files in the MIF 1.1 and MIF 1.2 formats are also accepted. They are passed to **mifconvert** for conversion to MIF 2 format "on-the-fly."

Note on Tk dependence: Some MIF 2 problem descriptions rely on external image files; examples include those using the `Oxs_ImageAtlas` class (Sec. 7.3.1), or those using the MIF 2 `ReadFile` command with the `image` translation specification (Sec. 17.3.2). If the image file is not in the PPM P3 (text) format, then the **any2ppm** application may be launched to read and convert the file. Since **any2ppm** requires Tk, at the time the image file is read a valid display must be available. See the **any2ppm** documentation for details.

7.1 OOMMF eXtensible Solver Interactive Interface: Oxsii



Overview

The application **Oxsii** is the graphical, interactive user interface to the Oxs micromagnetic computation engine. Within the OOMMF architecture, **Oxsii** is both a server and a client application. **Oxsii** is a client of data table display and storage applications, and vector field display and storage applications. **Oxsii** is the server of a solver control service for which the only client is **mmLaunch**. It is through this service that **mmLaunch** provides a user interface window (shown above) on behalf of **Oxsii**.

A micromagnetic problem is communicated to **Oxsii** via a MIF 2 file, which defines a collection of Oxs_Ext objects that comprise the problem model. The problem description includes a segmentation of the lifetime of the simulation into stages. Stages mark discontinuous changes in model attributes, such as applied fields, and also serve to mark overall simulation progress. **Oxsii** provides controls to advance the simulation, stopping between iterations, between stages, or only when the run is complete. Throughout the simulation, the user may save and display intermediate results, either interactively or via scheduling based on iteration and stage counts.

Problem descriptions in the MIF 1.1 and MIF 1.2 formats can also be input. They are automatically passed to **mifconvert** for implicit conversion to MIF 2 format. MIF 2 files are actually Tcl scripts with a handful of extra commands. This makes them extremely flexible, but they can also be difficult to compose from scratch. In many cases it may be easier to first use **mmProbEd** set to MIF 1.2 mode to create a draft problem. Use the `File|Save as` menu option in **mmProbEd** to write the problem to disk. If MIF 1.2 provides all the needed functionality, then one can keep the problem in that format, loading it directly into **Oxsii** and **Boxsi**, and continue to edit it in **mmProbEd** as necessary. If and when some more advanced functionality available only in MIF 2.x is needed, the **mifconvert** command line application can be run manually to create a MIF 2.x file from the MIF 1.2 version. The

MIF 2.x file can then be hand edited to suit. Do note, however, that there is no MIF 2.x to MIF 1.2 back conversion support.

See MIF 2.x tips for additional recommendations on writing MIF 2.x files.

Launching

Oxsii may be started either by selecting the **Oxsii** button on mmLaunch, or from the command line via

```
tclsh oommf.tcl oxsii [standard options] [-exitondone <0|1>] \
   [-logfile logname] [-loglevel level] [-nice <0|1>] [-nocrccheck <0|1>] \
   [-nologfile] [-numanodes nodes] [-outdir dir] [-parameters params] \
   [-pause <0|1>] [-quiet level] [-restart <0|1|2>] [-restartfiledir dir] \
   [-restartfilename name] [-threads count] [miffile]
```

where

-exitondone <0|1> Whether to exit after solution of the problem is complete. Default is to simply await the interactive selection of another problem to be solved.

-logfile logname Write log and error messages to file *logname*. The default log file is `oommf/oxsii.log` (formerly `oxsii.errors`).

-loglevel level Controls the detail level of log messages, with larger values of *level* producing more output. Default value is 1.

-nice <0|1> If enabled (i.e., 1), then the program will drop its scheduling priority after startup. The default is 1, i.e., to yield scheduling priority to other applications.

-nocrccheck <0|1> On simulation restarts the CRC of the MIF file and the run parameters are normally compared against the CRC of the original MIF file and parameters as recorded in the restart file. If the CRCs or parameters don't match then an error is thrown to alert the user that the run state is changed. If this option is enabled (i.e., 1) then those checks are disabled.

-nologfile If this option is set then logging information is not written to disk but is instead written to (only) stderr (regardless of `-quiet level`).

-numanodes <nodes> This option is available on NUMA-aware builds of Oxs. The *nodes* parameter must be either a comma separated list of 0-based node numbers, the keyword "auto", or the keyword "none". In the first case, the numbers refer to memory nodes. These must be passed on the command line as a single parameter, so either ensure there are no spaces in the list, or else protect the spaces with outlying quotes. For example, `-numanodes 2,4,6` or `-numanodes "2, 4, 6"`. Threads are assigned to the nodes in order, in round-robin fashion. The user can either assign all the system nodes

to the **Oxsii** process, or may restrict **Oxsii** to run on a subset of the nodes. In this way the user may reserve specific processing cores for other processes (or other instances of **Oxsii**). Although it varies by system, typically there are multiple processing cores associated with each memory node. If the keyword "auto" is selected, then the threads are assigned to a fixed node sequence that spans the entire list of memory nodes. If the keyword "none" is selected, then threads are not tied to nodes by **Oxsii**, but are instead assigned by the operating system. In this last case, over time the operating system is free to move the threads among processors. In the other two cases, each thread is tied to a particular node for the lifetime of the **Oxsii** instance. See also the discussion on threading considerations in the Boxsi documentation.

The default value for *nodes* is "none", which allows the operating system to assign and move threads based on overall system usage. This is also the behavior obtained when the Oxs build is not NUMA-aware. On the other hand, if a machine is dedicated primarily to running one instance of **Oxsii**, then **Oxsii** will likely run fastest if the thread count is set to the number of processing cores on the machine, and *nodes* is set to "auto". If you want to run multiple copies of **Oxsii** simultaneously, or run **Oxsii** in parallel with some other application(s), then set the thread count to a number smaller than the number of processing cores and restrict **Oxsii** to some subset of the memory nodes with the `-numanodes` option and an explicit nodes list.

The default behavior is modified (in increasing order of priority) by the `numanodes` setting in the active `oommf/config/platform/` platform file, by the `numanodes` setting in the `oommf/config/options.tcl` or `oommf/config/local/options.tcl` file, or by the environment variable `OOMMF_NUMANODES`. The `-numanodes` command line option, if any, overrides all.

-outdir dir Specifies the directory where output files are written by **mmArchive**. This option is useful when the default output directory is inaccessible or slow. The environment variable `OOMMF_OUTDIR` sets the default output directory. If `OOMMF_OUTDIR` is set to the empty string, or not set at all, then the default is the directory holding the MIF file. If this option is specified on the command line, or if `OOMMF_OUTDIR` is set, then the **Oxsii** `File|Load` dialog box includes a control to change the output directory.

-parameters params Sets MIF 2 file parameters. The *params* argument should be a list with an even number of arguments, corresponding to name + value pairs. Each "name" must appear in a `Parameter` statement (Sec. 17.3.2) in the input MIF file. The entire name + value list must be quoted so it is presented to **Oxsii** as a single item on the command line. For example, if `A` and `Ms` appeared in `Parameter` statements in the MIF file, then an option like

```
-parameters "A 13e-12 Ms 800e3"
```

could be used to set `A` to 13e-12 and Ms to 800e3. The quoting mechanism is specific to the shell/operating system; refer to your system documentation for details.

-pause <0|1> If disabled (i.e., 0), then the program automatically shifts into "Run" mode after loading the specified *miffile*. The default is 1, i.e., to "Pause" once the problem is loaded. This switch has no effect if miffile is not specified.

-quiet level Controls amount of output written to stdout and stderr. If *level* is 0 then log messages are echoed to stderr. The default setting is 1.

-restart <0|1> Controls the initial setting of the restart flag, and thereby the load restart behavior of any `miffile` specified on the command line. The restart flag is described in the Controls section below (page 41). The default value is 0, i.e., no restart.

-restartfiledir dir Specifies the directory where restart files are written. The default is determined by the environment variable `OOMMF_RESTARTFILEDIR`, or if this is not set then by `OOMMF_OUTDIR`. If neither environment variable is set then the default is the directory holding the MIF file. Write access is required to the restart file directory. Also, you may want to consider whether the restart files should be written to a local temporary directory or a network mount.

-restartfilename name Sets the name of the restart (checkpoint) file, overriding the name determined by the MIF file. If *name* is not an absolute path then the full path will be *name* appended to the restart directory (see the `-restartfiledir` entry above). One use of this option is to restart runs where the restart file name from the original run includes an externally derived component (perhaps a timestamp) that won't be recreated by the MIF file logic on the restart run.

NOTE: If this command line option is used, then *name* will be restart file name for **all** jobs launched from this instance of **Oxsii**.

-threads <count> The option is available on threaded builds of Oxs. The *count* parameter is the number of threads to run. The default count value is set by the `oommf_thread_count` value in the `config/platforms/` file for your platform, but may be overridden by the `OOMMF_THREADS` environment variable or this command line option. In most cases the default count value will equal the number of processing cores on the system; this can be checked via the command `tclsh oommf.tcl +platform`.

miffile Load and solve the problem found in *miffile*, which must be either in the MIF 2 format, or convertible to that format by **mifconvert**. Optional.

All the above switches are optional.

Since **Oxsii** does not present any user interface window of its own, it depends on **mmLaunch** to provide an interface on its behalf. The entry for an instance of **Oxsii** in the **Running Applications** column of any running copy of **mmLaunch** has a checkbutton next to it. This button toggles the presence of a user interface window through which the user may control that instance of **Oxsii**.

Inputs

Unlike **mmSolve2D**, **Oxsii** loads problem specifications directly from disk (via the `File|Load` menu selection), rather than through **mmProbEd** or **FileSource**. Input files for **Oxsii** must be either in the MIF 2 format, or convertible to that format by the command line tool **mifconvert**. There are sample MIF 2 files in the directory `oommf/app/oxs/examples`. MIF files may be edited with any plain text editor.

Outputs

Once a problem has been loaded, the scroll box under the Output heading will fill with a list of available outputs. The contents of this list will depend upon the `Oxs_Ext` objects specified in the input MIF file. Refer to the documentation for those objects for specific details. To send output from **Oxsii** to another OOMMF application, highlight the desired selection under the Output heading, make the corresponding selection under the Destination heading, and then specify the output timing under the Schedule heading. Outputs may be scheduled by the step or stage, and may be sent out interactively by pressing the **Send** button. The initial output configuration is set by `Destination` and `Schedule` commands in the input MIF file (Sec. 17.3.2).

Outputs fall under two general categories: scalar (single-valued) outputs and vector field outputs. The scalar outputs are grouped together as the `DataTable` entry in the Output scroll box. Scalar outputs include such items as total and component energies, average magnetization, stage and iteration counts, max torque values. When the `DataTable` entry is selected, the Destination box will list all OOMMF applications accepting datatable-style input, i.e., all currently running **mmDataTable**, **mmGraph**, and **mmArchive** processes.

The vector field outputs include pointwise magnetization, various total and partial magnetic fields, and torques. Unlike the scalar outputs, the vector field outputs are listed individually in the Output scroll box. Allowed destinations for vector field output are running instances of **mmDisp** and **mmArchive**. Caution is advised when scheduling vector field output, especially with large problems, because the output may run many megabytes.

Controls

The **File** menu button holds five entries: Load, Show Console, Close Interface, Suspend Oxsii and Exit Oxsii. **File|Load** launches a dialog box that allows the user to select an input MIF problem description file. **File|Show Console** brings up a Tcl shell console running off the **Oxsii** interface Tcl interpreter. This console is intended primary for debugging purposes. In particular, output from MIF Report commands (Sec. 17.3.2) may be viewed here. **File|Close Interface** will remove the interface window from the display, but leaves the solver running. This effect may also be obtained by deselecting the **Oxsii** interface button in the **Running Applications** list in **mmLaunch**. **File|Suspend Oxsii** causes the **Oxsii** solver to write the current simulation state into a checkpoint (`.restart`) file and exit. The checkpoint file may be used to resume the simulation at a later time using the command line `-restart` or

File|Load dialog `Restart` checkbutton controls. On Unix systems the Suspend operation can also be triggered by sending a `SIGUSR1` signal to the **Oxsii** process. (See the Oxs_Driver documentation, Sec. 7.3.5 page 93, for information on checkpoint files.) The final entry, **File|Exit Oxsii**, terminates the **Oxsii** solver and closes the interface window.

The **Options** menu holds three entries: Clear Schedule, Checkpoint Control and Restart Flag. The first clears all Step and Stage selections from the active output schedules, exactly as if the user clicked through the interactive schedule interface one output and destination at a time and disabled each schedule-enabling checkbutton. This control can be used after loading a problem to override the effect of any `Schedule` commands in the MIF file. (In contrast, the **Clr Sched** button in the **File|Load** dialog clears the existing schedule **before** loading the MIF file.) The Checkpoint Control option opens a dialog box that reports the name of the checkpoint (`.restart`) file and allows interactive setting of the checkpoint interval and checkpoint file disposal. The restart flag controls problem load behavior. In normal usage, the restart flag is not set and the selected problem loads and runs from the beginning. Conversely, if the restart flag is set, then when a problem is loaded a check is made for a checkpoint file. If the checkpoint file is not found, then an error is raised. Otherwise, the information in the checkpoint file is used to resume the problem from the state saved in that file. The restart flag can be set from the Options menu, the `File|Load` dialog box, or from the command line. See the Oxs_Driver documentation, Sec. 7.3.5 page 93, for information on checkpoint files.

The **Help** menu provides the usual help facilities.

The row of buttons immediately below the menu bar provides simulation progress control. These buttons become active once a problem has been loaded. The first button, **Reload**, re-reads the most recent problem MIF input file, re-initializes the solver, and pauses. **Reset** is similar, except the file is not re-read. The remaining four buttons, **Run**, **Relax**, **Step** and **Pause** place the solver into one of four *run-states*. In the Pause state, the solver sits idle awaiting further instructions. If **Step** is selected, then the solver will move forward one iteration and then Pause. In Relax mode, the solver takes at least one step, and then runs until it reaches a stage boundary, at which point the solver is paused. In Run mode, the solver runs until the end of the problem is reached. Interactive output is available in all modes; the scheduled outputs occur appropriately as the step and stage counts advance.

Directly below the progress control buttons are two display lines, showing the name of the input MIF file and the current run-state. Below the run-state `Status` line is the stage display and control bar. The simulation stage may be changed at any time by dragging the scroll bar or by typing the desired stage number into the text display box to the left of the scroll bar. Valid stage numbers are integers from 0 to $N-1$, where N is the number of stages specified by the MIF input file.

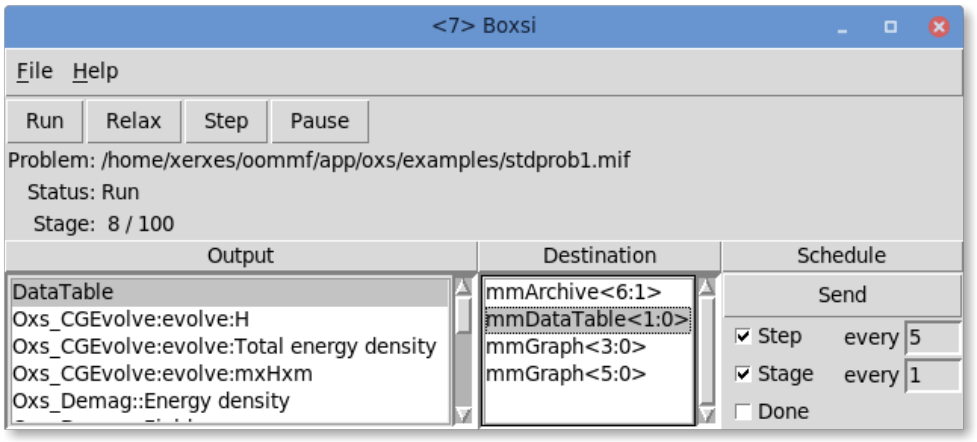
Details

The simulation model construction is governed by the Specify blocks in the input MIF file. Therefore, all aspects of the simulation are determined by the specified Oxs_Ext classes.

Refer to the appropriate Oxs_Ext class documentation for simulation and computational details.

Error and status messages are written to the log file (default `oommf/oxsii.log`). This is a plain text file where each record is prefixed with a timestamp and source identification. The start of each simulation is recorded with a `Start:` record, and an `End` record marks successful completion of the simulation. If you have multiple simulations running simultaneously, either on the same machine or on different machines but logging to the same (networked) file, then records from the different simulations will be intermingled. The OOMMF command line application **logreview** can be used as an aid for reading **Oxsii** log files. See also the command line application **lastjob**, which relies on the log file to determine the last simulation run and optionally restart interrupted simulations.

7.2 OOMMF eXtensible Solver Batch Interface: Boxsi



Overview

The application **Boxsi** provides a batch mode interface to the Oxs micromagnetic computation engine. A restricted graphical interface is provided, but **Boxsi** is primarily intended to be controlled by command line arguments, and launched by the user either directly from the shell prompt or from inside a batch file.

Within the OOMMF architecture, **Boxsi** is both a server and a client application. It is a client of data table display and storage applications, and vector field display and storage applications. **Boxsi** is the server of a solver control service for which the only client is **mmLaunch**. It is through this service that **mmLaunch** provides a user interface window (shown above) on behalf of **Boxsi**.

A micromagnetic problem is communicated to **Boxsi** through a MIF 2 file specified on the command line and loaded from disk. The MIF 1.x formats are also accepted; they are converted to the MIF 2 format by an automatic call to **mifconvert**.

Launching

Boxsi must be started from the command line. The syntax is

```
tclsh oommf.tcl boxsi [standard options] [-exitondone <0|1>] \
   [-kill tags] [-logfile logname] [-loglevel level] [-nice <0|1>] \
   [-nocrccheck <0|1>] [-nologfile] [-numanodes nodes] [-outdir dir] \
   [-parameters params] [-pause <0|1>] [-quiet level] \
   [-regression_test flag] [-regression_testname basename] \
   [-restart <0|1|2>] [-restartfiledir dir] [-restartfilename name] \
   [-threads count] miffile
```

where

-exitondone <0|1> Whether to exit after solution of the problem is complete, or to await the interactive selection of the **File|Exit** command. The default is 1, i.e., automatically exit when done.

-kill tags On termination, sends requests to other applications to shutdown too. The *tags* argument should be either a list of destination tags (which are declared by `Destination` commands, Sec. 17.3.2) from the input MIF file, or else the keyword `all`, which is interpreted to mean all the destination tags.

-logfile logname Write log and error messages to file *logname*. The default log file is `oommf/boxsi.log` (formerly `boxsi.errors`).

-loglevel level Controls the detail level of log messages, with larger values of *level* producing more output. Default value is 1.

-nice <0|1> If enabled (i.e., 1), then the program will drop its scheduling priority after startup. The default is 0, i.e., to retain its original scheduling priority.

-nocrccheck <0|1> On simulation restarts the CRC of the MIF file and the run parameters are normally compared against the CRC of the original MIF file and parameters as recorded in the restart file. If the CRCs or parameters don't match then an error is thrown to alert the user that the run state is changed. If this option is enabled (i.e., 1) then those checks are disabled.

-nologfile If this option is set then logging information is not written to disk but is instead written to (only) stderr (regardless of `-quiet level`). This may be useful in particular when running off a batch queue, where all stdout and stderr output is collected anyway by the job scheduler. Any potential issues arising from multiple jobs attempting simultaneous writes to a shared log file are also bypassed.

-numanodes <nodes> This option is available on NUMA-aware builds of Oxs. The *nodes* parameter must be either a comma separated list of 0-based node numbers, the keyword

"auto", or the keyword "none". In the first case, the numbers refer to memory nodes. These must be passed on the command line as a single parameter, so either ensure there are no spaces in the list, or else protect the spaces with outlying quotes. For example, `-numanodes 2,4,6` or `-numanodes "2, 4, 6"`. Threads are assigned to the nodes in order, in round-robin fashion. The user can either assign all the system nodes to the **Boxsi** process, or may restrict **Boxsi** to run on a subset of the nodes. In this way the user may reserve specific processing cores for other processes (or other instances of **Boxsi**). Although it varies by system, typically there are multiple processing cores associated with each memory node. If the keyword "auto" is selected, then the threads are assigned to a fixed node sequence that spans the entire list of memory nodes. If the keyword "none" is selected, then threads are not tied to nodes by **Boxsi**, but are instead assigned by the operating system. In this last case, over time the operating system is free to move the threads among processors. In the other two cases, each thread is tied to a particular node for the lifetime of the **Boxsi** instance. See also the discussion on threading considerations below.

The default value for *nodes* is "none", which allows the operating system to assign and move threads based on overall system usage. This is also the behavior obtained when the Oxs build is not NUMA-aware. On the other hand, if a machine is dedicated primarily to running one instance of **Boxsi**, then **Boxsi** will likely run fastest if the thread count is set to the number of processing cores on the machine, and *nodes* is set to "auto". If you want to run multiple copies of **Boxsi** simultaneously, or run **Boxsi** in parallel with some other application(s), then set the thread count to a number smaller than the number of processing cores and restrict **Boxsi** to some subset of the memory nodes with the `-numanodes` option and an explicit nodes list.

The default behavior is modified (in increasing order of priority) by the `numanodes` setting in the active `oommf/config/platform/` platform file, by the `numanodes` setting in the `oommf/config/options.tcl` or `oommf/config/local/options.tcl` file, or by the environment variable `OOMMF_NUMANODES`. The `-numanodes` command line option, if any, overrides all.

-outdir dir Specifies the directory where output files are written by **mmArchive**. This option is useful when the default output directory is inaccessible or slow. The environment variable `OOMMF_OUTDIR` sets the default output directory. If `OOMMF_OUTDIR` is set to the empty string, or not set at all, then the default is the directory holding the MIF file.

-parameters params Sets MIF 2 file parameters. The *params* argument should be a list with an even number of arguments, corresponding to name + value pairs. Each "name" must appear in a `Parameter` statement (Sec. 17.3.2) in the input MIF file. The entire name + value list must be quoted so it is presented to **Boxsi** as a single item on the command line. For example, if `A` and `Ms` appeared in `Parameter` statements in the MIF file, then an option like

```
-parameters "A 13e-12 Ms 800e3"
```

could be used to set `A` to 13e-12 and Ms to 800e3. The quoting mechanism is specific to the shell/operating system; refer to your system documentation for details.

-pause <0|1> If enabled (i.e., 1), then the program automatically pauses after loading the specified problem file. The default is 0, i.e., to automatically move into "Run" mode once the problem is loaded.

-quiet level Controls amount of output written to stdout and stderr. If *level* is 0 then log messages are echoed to stderr. The default setting is 1.

-regression_test flag This option is used internally by the **oxsregression** command line utility to run regression tests. Default value is 0 (no test).

-regression_testname basename This option is used internally by the **oxsregression** command line utility to control temporary file names during regression testing.

-restart <0|1|2> If the restart option is 0 (the default), then the problem loads and runs from the beginning. If set to 1, then when loading the problem a check is made for a pre-existing restart (checkpoint) file. If one is found, then the problem resumes from the state saved in that file. If no checkpoint file is found, then an error is raised. If the restart option is set to 2, then a checkpoint file is used if one can be found, but if not then the problem loads and runs from the beginning without raising an error. See the Oxs_Driver documentation, Sec. 7.3.5 page 93, for information on checkpoint files.

-restartfiledir dir Specifies the directory where restart files are written. The default is determined by the environment variable `OOMMF_RESTARTFILEDIR`, or if this is not set then by `OOMMF_OUTDIR`. If neither environment variable is set then the default is the directory holding the MIF file. Write access is required to the restart file directory. Also, you may want to consider whether the restart files should be written to a local temporary directory or a network mount.

-restartfilename name Sets the name of the restart (checkpoint) file, overriding the name determined by the MIF file. If *name* is not an absolute path then the full path will be *name* appended to the restart directory (see the `-restartfiledir` entry above). One use of this option is to restart runs where the restart file name from the original run included an externally derived component (perhaps a timestamp) that won't be recreated by the MIF file logic on the restart run.

-threads <count> The option is available on threaded builds of Oxs. The *count* parameter is the number of threads to run. The default count value is set by the `oommf_thread_count` value in the `config/platforms/` file for your platform, but may be overridden by the `OOMMF_THREADS` environment variable or this command line option. In most cases the default count value will equal the number of processing cores on the system; this can be checked via the command `tclsh oommf.tcl +platform`.

miffile Load and solve the problem found in *miffile*, which must be either in the MIF 2 format, or convertible to that format by **mifconvert**. Required.

Although **Boxsi** cannot be launched by **mmLaunch**, nonetheless a limited graphical interactive interface for **Boxsi** is provided through **mmLaunch**, in the same manner as is done for **Oxsii**. Each running instance of **Boxsi** is included in the **Running Applications** list of **mmLaunch**, along with a checkbutton. This button toggles the presence of a user interface window.

Inputs

Boxsi loads problem specifications directly from disk as requested on the command line. The format for these files is the MIF 2 format, the same as used by the **Oxsii** interactive interface. The MIF 1.1 and MIF 1.2 formats used by the 2D solver **mmSolve2D** can also be input to **Boxsi**, which will automatically call the command line tool **mifconvert** to convert from the MIF 1.x format to the MIF 2 format "on-the-fly." Sample MIF 2 files can be found in the directory `oommf/app/oxs/examples`.

Outputs

The lower panel of the **Boxsi** interactive interface presents Output, Destination, and Schedule sub-windows that display the current output configuration and allow interactive modification of that configuration. These controls are identical to those in the **Oxsii** user interface; refer to the **Oxsii** documentation (Sec. 7.1) for details. The only difference between **Boxsi** and **Oxsii** with respect to outputs is that in practice **Boxsi** tends to rely primarily on `Destination` and `Schedule` commands in the input MIF file (Sec. 17.3.2) to setup the output configuration. The interactive output interface is used for incidental runtime monitoring of the job.

Controls

The runtime controls provided by the **Boxsi** interactive interface are a restricted subset of those available in the **Oxsii** interface. If the runtime controls provided by **Boxsi** are found to be insufficient for a given task, consider using **Oxsii** instead.

The **File** menu holds six entries: Show Console, Close Interface, Clear Schedule, Checkpoint Control, Suspend Boxsi and Exit Oxsii. **File|Show Console** brings up a Tcl shell console running off the **Boxsi** interface Tcl interpreter. This console is intended primary for debugging purposes. **File|Close Interface** will remove the interface window from the display, but leaves the solver running. This effect may also be obtained by deselecting the **Boxsi** interface button in the **Running Applications** list in **mmLaunch**. **File|Clear Schedule** will disable all currently active output schedules, exactly as if the user clicked through the interactive schedule interface one output and destination at a time and disabled each schedule-enabling checkbutton. **File|Checkpoint Control** opens a dialog box that reports

the name of the checkpoint (`.restart`) file and allows interactive setting of the checkpoint interval and checkpoint file disposal. **File|Suspend Boxsi** causes the **Boxsi** solver to write the current simulation state into a checkpoint (`.restart`) file and exit. The checkpoint file may be used to resume the simulation at a later time using the `-restart` command line control. On Unix systems the Suspend operation can also be triggered by sending a `SIGUSR1` signal to the **Boxsi** process. (See the Oxs_Driver documentation, Sec. 7.3.5 page 93, for information on checkpoint files.) The final entry, **File|Exit Boxsi**, terminates the **Boxsi** solver and closes the interface window. Note that there is no **File|Load** menu item; the problem specification file must be declared on the **Boxsi** command line.

The **Help** menu provides the usual help facilities.

The row of buttons immediately below the menu bar provides simulation progress control. These buttons—**Run**, **Relax**, **Step** and **Pause**—become active once the micromagnetic problem has been initialized. These buttons allow the user to change the run state of the solver. In the Pause state, the solver sits idle awaiting further instructions. If **Step** is selected, then the solver will move forward one iteration and then Pause. In Relax mode, the solver takes at least one step, and then runs until it reaches a stage boundary, at which point the solver is paused. In Run mode, the solver runs until the end of the problem is reached. When the problem end is reached, the solver will either pause or exit, depending upon the setting of the `-exitondone` command line option.

Normally the solver progresses automatically from problem initialization into Run mode, but this can be changed by the `-pause` command line switch. Interactive output is available in all modes; the scheduled outputs occur appropriately as the step and stage counts advance.

Directly below the run state control buttons are three display lines, showing the name of the input MIF file, the current run-state, and the current stage number/maximum stage number. Both stage numbers are 0-indexed.

Details

As with **Oxsii**, the simulation model construction is governed by the Specify blocks in the input MIF file, and all aspects of the simulation are determined by the specified Oxs_Ext classes. Refer to the appropriate Oxs_Ext class documentation for simulation and computational details.

Error and status messages are written to the log file (default `oommf/boxsi.log`). This is in the same log format as used by **Oxsii**. It is a plain text file where each record is prefixed with a timestamp and source identification. The start of each simulation is recorded with a `Start:` record, and an `End` record marks successful completion of the simulation. If you have multiple simulations running simultaneously, either on the same machine or on different machines but logging to the same (networked) file, then records from the different simulations will be intermingled. The OOMMF command line application **logreview** can be used as an aid for reading **Boxsi** log files. See also the command line application **lastjob**, which relies on the log file to determine the last simulation run and optionally restart interrupted simulations.

Threading considerations

As an example, suppose you are running on a four dual-core processor box, where each of the four processors is connected to a separate memory node. In other words, there are eight cores in total, and each pair of cores shares a memory node. Further assume that the processors are connected via point-to-point links such as AMD's HyperTransport or Intel's QuickPath Interconnect.

If you want to run a single instance of **Boxsi** as quickly as possible, you might use the `-threads 8` option, which, assuming the default value of `-numanodes none` is in effect, would allow the operating system to schedule the eight threads among the system's eight cores as it sees fit. Or, you might reduce the thread count to reserve one or more cores for other applications. If the job is long running, however, you may find that the operating system tries to run multiple threads on a single core—perhaps in order to leave other cores idle so that they can be shut down to save energy. Or, the operating system may move threads away from the memory node where they have allocated memory, which effectively reduces memory bandwidth. In such cases you might want to launch **Boxsi** with the `-numanodes auto` option. This overrides the operating systems preferences, and ties threads to particular memory nodes for the lifetime of the process. (On Linux boxes, you should also check the "cpu frequency governor" and "huge page support" selection and settings.)

If you want to run two instances of **Boxsi** concurrently, you might launch each with the `-threads 4` option, so that each job has four threads for the operating system to schedule. If you don't like the default scheduling by the operating system, you can use the `-numanodes` option, but what you **don't** want to do is launch two jobs with `-numanodes auto`, because the "auto" option assigns threads to memory nodes from a fixed sequence list, so both jobs will be assigned to the same nodes. Instead, you should manually assign the nodes, with a different set to each job. For example, you may launch the first job with `-numanodes 0,1` and the second job with `-numanodes 2,3`. One point to keep in mind when assigning nodes is that some node pairs are "closer" (with respect to memory latency and bandwidth) than others. For example, memory node 0 and memory node 1 may be directly connected via a point-to-point link, so that data can be transferred in a single "hop." But sending data from node 0 to node 2 may require two hops (from node 0 to node 1, and then from node 1 to node 2). In this case `-numanodes 0,1` will probably run faster than `-numanodes 0,2`.

The `-numanodes` option is only available on Linux boxes if the "numactl" and "numactl-devel" packages are installed. The `numactl` command itself can be used to tie jobs to particular memory nodes, similar to the `boxsi -numanodes` option, except that `-numanodes` ties threads whereas `numactl` ties jobs. The `numactl -hardware` command will tell you how many memory nodes are in the system, and also reports a measure of the (memory latency and bandwidth) distance between nodes. This information can be used in selecting nodes for the `boxsi -numanodes` option, but in practice the distance information reported by `numactl` is often not reliable. For best results one should experiment with different settings, or run memory bandwidth tests with different node pairs.

Batch Scheduling Systems

OOMMF jobs submitted to a batch queuing system (e.g., Condor, PBS, NQS) can experience sporadic failures caused by interactions between separate OOMMF jobs running simultaneously on the same compute node. These problems can be prevented by using the OOMMF command line utility **launchhost** to isolate each job.

7.3 Standard Oxs_Ext Child Classes

An Oxs simulation is built as a collection of `Oxs_Ext` (Oxs Extension) objects. These are defined via Specify blocks in the input MIF 2 file. The reader will find the information and sample MIF file provided in the MIF documentation chapter to be a helpful adjunct to the material presented below. Additional example MIF 2 files can be found in the directory `oommf/app/oxs/examples`.

This section describes the `Oxs_Ext` classes available in the standard OOMMF distribution, organized into 8 categories: atlases, meshes, energies, evolvers, drivers, scalar field objects, vector field objects, and MIF support classes. The documentation for each class explains its Specify block initialization options and provides a list of some sample MIF files from the `oommf/app/oxs/examples` directory that use the class. The standard `Oxs_Ext` objects, i.e., those that are distributed with OOMMF, can be identified by the `Oxs_` prefix in their names. Additional `Oxs_Ext` classes may be available on your system. Check local documentation for details.

Each of the energy classes provides a three-component field output "Field," a scalar field output "Energy density," and a scalar output "Energy." The solver and driver classes produce various scalar and field outputs, as described in their documentation below. The user may construct additional scalar (DataTable) outputs from the default field outputs via the `user_output` facility; see the MIF Specify Conventions section for details.

The following `Oxs_Ext` classes are currently provided:

- **Atlases**
  `Oxs_BoxAtlas`
  `Oxs_EllipseAtlas`
  `Oxs_EllipsoidAtlas`
  `Oxs_FileAtlas`
  `Oxs_ImageAtlas`
  `Oxs_MultiAtlas`
  `Oxs_ScriptAtlas`
  `Oxs_ParallelepipedAtlas`

- **Meshes**
  `Oxs_RectangularMesh`
  `Oxs_PeriodicRectangularMesh`

- **Energies**

Oxs_CubicAnisotropy
Oxs_Demag
Oxs_Exchange6Ngbr
Oxs_ExchangePtwise
Oxs_FixedZeeman
Oxs_RandomSiteExchange
Oxs_ScriptUZeeman
Oxs_SimpleDemag
Oxs_StageZeeman
Oxs_TransformZeeman
Oxs_TwoSurfaceExchange
Oxs_UniaxialAnisotropy
Oxs_UniformExchange
Oxs_UZeeman

- **Evolvers**
  Oxs_CGEvolve
  Oxs_EulerEvolve
  Oxs_RungeKuttaEvolve
  Oxs_SpinXferEvolve

- **Drivers**
  Oxs_TimeDriver
  Oxs_MinDriver

- **Scalar Field Objects**
  Oxs_AtlasScalarField
  Oxs_LinearScalarField
  Oxs_RandomScalarField
  Oxs_ScriptScalarField
  Oxs_UniformScalarField
  Oxs_VecMagScalarField
  Oxs_ScriptOrientScalarField
  Oxs_AffineOrientScalarField
  Oxs_AffineTransformScalarField
  Oxs_ImageScalarField

- **Vector Field Objects**
  Oxs_AtlasVectorField
  Oxs_FileVectorField
  Oxs_PlaneRandomVectorField
  Oxs_RandomVectorField
  Oxs_ScriptVectorField
  Oxs_UniformVectorField
  Oxs_ScriptOrientVectorField
  Oxs_AffineOrientVectorField
  Oxs_AffineTransformVectorField
  Oxs_MaskVectorField
  Oxs_ImageVectorField

- **MIF Support Classes**
  Oxs_LabelValue

7.3.1 Atlases

Geometric volumes of spaces are specified in Oxs via *atlases*, which divide their domain into one or more disjoint subsets called *regions*. Included in each atlas definition is the atlas *bounding box*, which is an axes parallel rectangular parallelepiped containing all the regions. There is also the special *universe* region, which consists of all points outside the regions specified in the atlas. The universe region is not considered to be part of any atlas, and the `universe` keyword should not be used to label any of the atlas regions.

The most commonly used atlas is the simple `Oxs_BoxAtlas`. For combining multiple atlases, use `Oxs_MultiAtlas`.

Oxs_BoxAtlas: An axes parallel right rectangular prism, containing a single region that is coterminous with the atlas itself. The specify block has the form

```
Specify Oxs_BoxAtlas:atlasname {
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    name regionname
}
```

where *xmin, xmax, ...* are coordinates in meters, specifying the extents of the volume being defined. The *regionname* label specifies the name assigned to the region contained in the atlas. The **name** entry is optional; if not specified then the region name is taken from the object instance name, i.e., *atlasname*.

Examples: `sample.mif`, `cgtest.mif`.

Oxs_ParallelepipedAtlas: Similar to `Oxs_BoxAtlas`, this class defines a region bounded by a parallelepiped described by one vertex and the three edges connected to that vertex. The faces aren't restricted to being parallel to the mesh coordinate planes. Note, however, that when rendered onto the coordinate mesh the edges and faces of the hexahedron will follow the staircase outline of the mesh. The Specify block has the form

```
Specify Oxs_ParallelepipedAtlas:atlasname {
    corner { x y z }
    u { ux uy uz }
    v { vx vy vz }
    w { wx wy wz }
    autorange { xcell ycell zcell [ xbase ybase zbase ] }
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    name regionname
}
```

where **corner** specifies one corner of the hexahedron, and **u**, **v**, and **w** are three vectors specifying the edge directions and lengths protruding from **corner**. An error will be raised if **u**, **v**, and **w** are co-planar.

The default bounding box for the atlas is the tightly fit coordinate-parallel cuboid. If this is used to define the simulation bounding box then an error will be raised by the mesh object if the bounding box is not an integral number of cells across. The first three parameters to the optional **autorange** entry, *xcell*, *ycell*, *zcell*, force the expansion of the atlas bounding box to an integral number of cells along each coordinate axis. If

six parameters are specified, then the last three specify an alignment point on the simulation mesh, and the atlas bounding box is adjusted to align with that mesh. Otherwise the expanded atlas bounding box is centered about the parallelepiped.

The optional **xrange**, **yrange**, and **zrange** entries override the bounding box computation along the corresponding axis. In particular, in some circumstances the **autorange** calculation may yield an unnecessary margin outside the active cells in a simulation; the range options can be used to trim such margins.

The *regionname* label specifies the name assigned to the region contained inside the parallelepiped. The **name** entry is optional; if not specified then the region name is taken from the object instance name, i.e., *atlasname*.

Note: For rotated parallelepipeds it may be easiest in the MIF file to first describe the edge vectors in an unrotated coordinate system and then use Tcl `expr` commands to apply rotation matrices to orient **u**, **v**, and **w** before creating the atlas. See the sample `parallelepiped.mif` file as an example.

Examples: `parallelepiped.mif`.

Oxs_EllipseAtlas: Defines a volume in the shape of a right elliptical cylinder with axes parallel to the coordinate axes. This functionality can be obtained using appropriate Tcl scripts with the `Oxs_ScriptAtlas` class, but this class is somewhat easier to use and runs faster. The Specify block has the form

```
Specify Oxs_EllipseAtlas:atlasname {
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    margin { margins }
    axis axisdir
    name { regions }
}
```

Here *xmin, xmax, ...* are coordinates in meters, specifying the bounding box for the atlas, similar to the layout of the Specify block for the `Oxs_BoxAtlas` class. The **margin** setting combines with the bounding box to determine the extent of the elliptical cylinder. The *margins* value is a list consisting of one, three, or six values, in units of meters. If the full six values $\{m_0, m_1, \ldots, m_5\}$ are specified they determine the bounding box for the elliptical cylinder as $[xmin + m_0, xmax - m_1] \times [ymin + m_2, ymax - m_3] \times [zmin + m_4, zmax - m_5]$. If three values are given then they are interpreted as margins for the x-coordinates, y-coordinates, and z-coordinates, respectively. If a single margin value is listed then that value is applied along all six faces. If the two margin values for a given coordinate are not equal, then the center of the cylinder will be shifted from the center of the atlas. If a margin value is negative then part of the

cylinder will be clipped at the atlas boundary. If **margin** is not given then the default is 0.

The *axisdir* should be one of x, y, or z, specifying the axis of symmetry for the cylinder. If not given the default is z.

The **name** setting is a list of one or two elements. A single value specifies the region name for the interior of the elliptical cylinder. In this case the exterior is automatically assigned to the global "universe" region. In the case of a two element list, the first element is the name assigned to the interior of the cylinder, the second element is the name assigned to the exterior of the cylinder. If desired, either one may be specified as "universe" to assign the corresponding volume to the global universe region. If **name** is not specified then it is treated by default as a one element list using the atlas object instance name, i.e., *atlasname*, as the interior region name.

Examples: `ellipse.mif`, `ellipsea.mif`.

Oxs_EllipsoidAtlas: Conceptually analogous to `Oxs_EllipseAtlas`, this class defines an ellipsoidal region with axes parallel to the coordinate axes. With appropriate Tcl scripts, the `Oxs_ScriptAtlas` class can provide the same functionality, but this class is somewhat easier to use and runs faster. The Specify block has the form

```
Specify Oxs_EllipsoidAtlas:atlasname {
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    margin { margins }
    name { regions }
}
```

All entries are interpreted in the same manner as for the `Oxs_EllipseAtlas` class.

Examples: `ellipsoid.mif` and `ellipsoid.mif`. See `ellipsoid-atlasproc.mif` and `ellipsoid-fieldproc.mif` for examples equivalent to `ellipsoid.mif` using Tcl scripts.

Oxs_FileAtlas: The `Oxs_FileAtlas` class defines regions through an imported file, somewhat similar to `Oxs_ImageAtlas`, but using a three dimensional OIF file in place of a two dimensional image. The `Oxs_FileAtlas` Specify block has the following form:

```
Specify Oxs_FileAtlas:name {
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    file filename
    regionmap {
```

region_name {index ...}
region_name {index ...}
...
region_name {index ...}
}
`autofmt` *format_string*
}

Here *xmin, xmax, ...* are coordinates in meters, specifying the extents of the axes-parallel rectangular parallelepiped enclosing the total volume being identified. If **xrange**/**yrange**/**zrange** are not specified, then the atlas bounding box will be imported from the OIF `x/y/zbase` and `x/y/zstepsize` records, if they exist.

The atlas volume is subdivided into regions using the names as given in the **regionmap** entry. The integer field (OIF) file *filename* is used to assign points to the various regions, with the file value (index) at each 3D location in *filename* mapping to the *region_name* in **regionmap** associated with the corresponding *index*. Each *region_name* may have as many indices associated with it as desired, but each index can map to at most one *region_name*. If optional **autofmt** is defined, then any file index not specified in **regionmap** is assigned to the region name generated by interpreting *format_string* (for example "`region%d`") as a C-style format string with the file index as a parameter. If **autofmt** is not declared, or is declared as an empty string, then an error will be raised if any file index is found that does not map to a *region_name*. Alternatively, one can set *format_string* to a region name string, such as "`universe`," to have all unmapped file indices assigned to that region.

If **regionmap** is missing then one will be constructed from the OIF file "`labels`" record, if any. The `labels` record is treated as equivalent to a **regionmap** with each `label` element as a region name associated with corresponding element index. For example, the OIF file record

```
# label: Fe Ni Co
```

is equivalent to the `Oxs_FileAtlas` Specify entry

```
regionmap {
   Fe 0
   Ni 1
   Co 2
}
```

An error will be raised if neither **regionmap** nor `labels` exist.

Regardless of the source of the volume extent, the data from *filename* are translated and scaled to fit that volume. You can truncate the extent by placing the `Oxs_FileAtlas` inside an `Oxs_MultiAtlas` with smaller, explicit `x/y/zregion` settings:

```
Specify Oxs_MultiAtlas:atlas {
   atlas { Oxs_FileAtlas {
      xrange { 0 240e-9 }
      yrange { 0 150e-9 }
      zrange { 0  18e-9 }
      regionmap {
         nonmagnetic {0 3}
         cobalt 1
           iron 2
      }
      file filelayers.oif
   }}
   xrange { 150e-9 240e-9 }
   yrange {  90e-9 150e-9 }
   zrange {   0     12e-9 }
}
```

Here `filelayers.oif` is imported with its natural size and location ($(0,400)\times(0,200)\times(0,20)$, in nanometers), but only the corner $(200,400)\times(100,200)\times(0,10)$ is used in the simulation. Conversely, if the `Oxs_FileAtlas` volume is encompassed by the `Oxs_MultiAtlas` `x/y/zrange` settings, then the `Oxs_FileAtlas` regions would cover only a subset of the full simulation volume.

Examples: `filelayers.mif`

Oxs_ImageAtlas: This class is designed to allow an image file to be used to define regions in terms of colors in the image. It is intended for use in conjunction with the `Oxs_AtlasScalarField` and `Oxs_AtlasVectorField` classes in circumstances where a discrete number of distinct species (materials or grains) are being modeled. This provides a generalization of the mask file functionality of the 2D solver (Sec. 17.1.3).

For situations requiring continuous variation in material parameters, the script field classes should be used in conjunction with the `ReadFile` MIF extension command. See the `ColorField` sample proc in the `ReadFile` documentation of Sec. 17.3.2 for an example of this technique.

The `Oxs_ImageAtlas` Specify block has the following form:

```
Specify Oxs_ImageAtlas:name {
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    viewplane view
    image pic
    colormap {
```

```
        color-1 region_name
        color-2 region_name
        ...
        color-n region_name
    }
    colorfunction function_type
    matcherror max_color_distance
}
```

The **xrange**, **yrange**, **zrange** entries specify the extent of the atlas, in meters. The **viewplane** *view* value should be one of the three two-letter codes `xy`, `zx` or `yz`, which specify the mapping of the horizontal and vertical axes of the image respectively to axes in the simulation. The image is scaled as necessary along each dimension to match the atlas extents along the corresponding axes. The image is overlaid through the entire depth of the perpendicular dimension, i.e., along the axis absent from the `viewplane` specification. If variation in the perpendicular direction is desired, for example to model a multilayer structure, then multiple `Oxs_ImageAtlas` objects with multiple images may be combined inside an `Oxs_MultiAtlas` object, as in example file `imagelayers.mif`. An alternative is to construct an equivalent three-dimensional OIF file and use the Oxs_FileAtlas class.

If the image aspect ratio doesn't match the ratio of the viewplane ranges, then the scaling will stretch or contract the image along one axis. One workaround for this is to set the extents in the `Oxs_ImageAtlas` to match the image aspect ratio, and use a separate atlas (perhaps an `Oxs_BoxAtlas`) to define the mesh and simulation extents. This approach can also be used to translate the image relative to the simulation extents. For an example see `imageatlas2.mif`.

The **image** entry specifies the name of the image file to use. If the file path is relative, then it will be taken with respect to the directory containing the MIF file. The image format may be any of those recognized by `any2ppm`. The file will be read directly by Oxs if it is in one of the PPM or Microsoft BMP (uncompressed) formats, otherwise `any2ppm` will be automatically launched to perform the conversion.

The **colormap** value is an even length list of color + region name pairs. The colors may be specified in any of several ways. The most explicit is to use one of the Tk numeric formats, `#rgb`, `#rrggbb`, `#rrrgggbbb` or `#rrrrggggbbbb`, where each r, g, and b is one hex digit (i.e., 0-9 or A-F) representing the red, green and blue components of the color, respectively. For example, `#F00` is bright (full-scale) red, `#800` would be a darker red, while `#FF0` and `#FFFF00` would both be bright yellow. Refer to the `Tk_GetColor` documentation for details. For shades of gray the special notation `grayD` or `greyD` is available, where D is a decimal value between 0 and 100, e.g., `grey0` is black and `grey100` is white. Alternatively, one may use any of the symbolic names defined in the `oommf/config/colors.config` file, such as `red`, `white` and `skyblue`. When comparing symbolic names, spaces and capitalization are ignored. The list of

symbolic names can be extended by adding additional files to the `Color filename` option in the `options.tcl` customization file. Finally, one *color* in the `colormap` list may optionally be the special keyword "default". All pixels that don't match any of the other specified colors (as determined by the `matcherror` option) are assigned to region paired with `default`.

Each of the specified colors should be distinct, but the region names are allowed to be repeated as desired. The region names may be chosen arbitrarily, except the special keyword "universe" is reserved for points not in any of the regions. This includes all points outside the atlas bounding box defined by the `xrange`, `yrange`, `zrange` entries, but may also include points inside that boundary.

The **colorfunction** option is an alternative to **colormap** which is useful when the number of regions is very large, for example with machine-generated grain maps[19] as in example file `grains.mif`. The *function_type* field must be "`auto`". With this option the image is scanned and region names are automatically generated and assigned to each distinct color (as defined by **matcherror**). The region name list may be accessed inside MIF 2.2 files via the `GetAtlasRegions` and `GetAtlasRegionByPosition` commands.

At least one of `colormap` or `colorfunction` must be specified. If both are used, then color designations from `colormap` take precedence over `colorfunction`.

Pixels in the image are assigned to regions by comparing the color of the pixel to the list of colors designated by `colormap/colorfunction`. If the pixel color is closer to a designated color than *max_color_distance*, then the colors are considered matched. If a pixel color matches exactly one designated color, then the pixel is assigned to the corresponding region. If a pixel color matches more than one designated color, the pixel is assigned to the region corresponding to the closest match. If a pixel color doesn't match any designated color, then it is assigned to the *default region*, which is the region paired with the "default" keyword. If `default` does not explicitly appear in the `colormap` colors list, then `universe` is made the default region.

To calculate the distance between two colors, each color is first converted to a scaled triplet of floating point red, green, and blue values, (r, g, b), where each component lies in the interval $[0, 1]$, with $(0, 0, 0)$ representing black and $(1, 1, 1)$ representing white. For example, $(0, 0, 1)$ is bright blue. Given two colors in this representation, the distance is computed using the standard Euclidean norm with uniform weights, i.e., the distance between (r_1, g_1, b_1) and (r_2, g_2, b_2) and is

$$\sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}.$$

Since the difference in any one component is at most 1, the distance between any two colors is at most $\sqrt{3}$.

As explained above, two colors are considered to match if the distance between them is less than the specified **matcherror** value. If *max_color_distance* is sufficiently small,

then it may easily happen that a pixel's color does not match any of the specified region colors, so the pixel would be assigned to the default region. On the other hand, if *max_color_distance* is larger than $\sqrt{3}$, then all colors will match, and no pixels will be assigned to the default region. If `matcherror` is not specified, then the default value for *max_color_distance* is 3, which means all colors match.

The following example should help clarify these matters.

```
Specify Oxs_ImageAtlas:atlas {
    xrange { 0 400e-9 }
    yrange { 0 200e-9 }
    zrange { 0  20e-9 }
    image  mypic.gif
    viewplane "xy"
    colormap {
        blue   cobalt
        red    permalloy
        green  universe
        default cobalt
    }
    matcherror .1
}
```

Blue pixels get mapped to the "cobalt" region and red pixels to the "permalloy" region. Green pixels are mapped to the "universe" non-region, which means they are considered to be outside the atlas entirely. This is a fine point, but comes into play when atlases with overlapping bounding boxes are brought together inside an `Oxs_MultiAtlas`. To which region would an orange pixel be assigned? The scaled triplet representation for orange is $(1, 0.647, 0)$, so the distance to blue is 1.191, the distance to red is 0.647, and the distance to green is 1.06. Thus the closest color is red, but 0.647 is outside the `matcherror` setting of 0.1, so orange doesn't match any of the colors and is hence assigned to the default region, which in this case is cobalt. On the other hand, if `matcherror` had been set to say 1, then orange and red would match and orange would be assigned to the permalloy region.

Pixels with colors that are equidistant to and match more than one color in the colormap will be assigned to one of the closest color regions. The user should not rely on any particular selection, that is to say, the explicit matching procedure in this case is not defined.

Examples: `imageatlas.mif`, `imageatlas2.mif`, `imagelayers.mif`, `grains.mif`, `grill.mif`.

Oxs_MultiAtlas: This atlas is built up as an ordered list of other atlases. The set of regions defined by the `Oxs_MultiAtlas` is the union of the regions of all the atlases

contained therein. The sub-atlases need not be disjoint, however each point is assigned to the region in the first sub-atlas in the list that contains it, so the regions defined by the `Oxs_MultiAtlas` are effectively disjoint.

The `Oxs_MultiAtlas` specify block has the form

```
Specify Oxs_MultiAtlas:name {
    atlas    atlas_1_spec
    atlas    atlas_2_spec
    ...
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
}
```

Each *atlas_spec* may be either a reference to an atlas defined earlier and outside the current Specify block, or else an inline, embedded atlas definition. The bounding box **xrange**, **yrange** and **zrange** specifications are each optional. If not specified the corresponding range for the atlas bounding box is taken from the minimal bounding box containing all the sub-atlases.

If the atlases are not disjoint, then the regions as defined by an `Oxs_MultiAtlas` can be somewhat different from those of the individual component atlases. For example, suppose `regionA` is a rectangular region in `atlasA` with corner points (5,5,0) and (10,10,10), and `regionB` is a rectangular region in `atlasB` with corner points (0,0,0) and (10,10,10). When composed in the order `atlasA`, `atlasB` inside an `Oxs_MultiAtlas`, `regionA` reported by the `Oxs_MultiAtlas` will be the same as `regionA` reported by `atlasA`, but `regionB` as reported by the `Oxs_MultiAtlas` will be the "L" shaped volume of those points in `atlasB`'s `regionB` not inside `regionA`. If the `Oxs_MultiAtlas` is constructed with `atlasB` first and `atlasA` second, then `regionB` as reported by the `Oxs_MultiAtlas` would agree with that reported by `atlasB`, but `regionA` would be empty.

NOTE: The `attributes` key label (cf. Sec. 17.3.3) is not supported by this class.

Examples: `manyregions-multiatlas.mif`, `spinvalve.mif`, `spinvalve-af.mif`, `yoyo.mif`.

Oxs_ScriptAtlas: An atlas where the regions are defined via a Tcl script. The specify block has the form

```
Specify Oxs_ScriptAtlas:name {
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    regions { rname_1 rname_2 ... rname_n }
```

```
    script_args { args_request }
    script Tcl_script
}
```

Here *xmin, xmax, ...* are coordinates in meters, specifying the extents of the axes-parallel rectangular parallelepiped enclosing the total volume being identified. This volume is subdivided into n sub-regions, using the names as given in the **regions** list. The **script** is used to assign points to the various regions. Appended to the script are the arguments requested by **script_args**, in the manner explained in the User Defined Support Procedures of the MIF 2 file format documentation. The value *args_request* should be a subset of {`relpt rawpt minpt maxpt span`}. If `script_args` is not specified, the default value `relpt` is used. When executed, the return value from the script should be an integer in the range 1 to n, indicating the user-defined region in which the point lies, or else 0 if the point is not in any of the n regions. Region index 0 is reserved for the implicit "universe" region, which is all-encompassing. The following example may help clarify the discussion:

```
proc Octs { cellsize x y z xmin ymin zmin xmax ymax zmax } {
    set xindex [expr {int(floor(($x-$xmin)/$cellsize))}]
    set yindex [expr {int(floor(($y-$ymin)/$cellsize))}]
    set zindex [expr {int(floor(($z-$zmin)/$cellsize))}]
    set octant [expr {1+$xindex+2*$yindex+4*$zindex}]
    if {$octant<1 || $octant>8} {
       return 0
    }
    return $octant
}

Specify Oxs_ScriptAtlas:octant {
    xrange {-20e-9 20e-9}
    yrange {-20e-9 20e-9}
    zrange {-20e-9 20e-9}
    regions { VIII V VII VI IV I III II }
    script_args { rawpt minpt maxpt }
    script { Octs 20e-9 }
}
```

This atlas divides the rectangular volume between $(-20,-20,-20)$ and $(20,20,20)$ (nm) into eight regions, corresponding to the standard octants, I through VIII. The `Octs` Tcl procedure returns a value between 1 and 8, with 1 corresponding to octant VIII and 8 to octant II. The canonical octant ordering starts with I as the $+x,+y,+z$ space, proceeds counterclockwise in the $+z$ half-space, and concludes in the $-z$ half-space with V directly beneath I, VI beneath II, etc. The ordering computed algorith-

mically in `Octs` starts with 1 for the $-x, -y, -z$ space, 2 for the $+x, -y, -z$ space, 3 for the $-x, +y, -z$ space, etc. The conversion between the two systems is accomplished by the ordering of the `regions` list.

Examples: `manyregions-scriptatlas.mif`, `octant.mif`, `pattern.mif`, `tclshapes.mif`, `diskarray.mif`, `ellipsoid-atlasproc.mif`.

7.3.2 Meshes

Meshes define the discretization impressed on the simulation. There should be exactly one mesh declared in a MIF 2 file. The usual (finite) mesh type is `Oxs_RectangularMesh`. For simulations that are periodic along one or more axes, use the `Oxs_PeriodicRectangularMesh` type.

Oxs_RectangularMesh: This mesh is comprised of a lattice of rectangular prisms. The specify block has the form

```
Specify Oxs_RectangularMesh:name {
    cellsize { xstep ystep zstep }
    atlas atlas_spec
}
```

This creates an axes parallel rectangular mesh across the entire space covered by **atlas**. The mesh sample rates along each axis are specified by **cellsize** (in meters). The mesh is cell-based, with the center of the first cell one half step in from the minimal extremal point (xmin,ymin,ymax) for *atlas_spec*. The *name* is commonly set to "mesh", in which case the mesh object may be referred to by other `Oxs_Ext` objects by the short name `:mesh`.

Examples: `sample.mif`, `stdprob3.mif`, `stdprob4.mif`.

Oxs_PeriodicRectangularMesh: Like the `Oxs_RectangularMesh`, this mesh is also comprised of a lattice of rectangular prisms. However, in this case the mesh is declared to be periodic along one or more of the axis directions. The specify block has the form

```
Specify Oxs_PeriodicRectangularMesh:name {
    cellsize { xstep ystep zstep }
    atlas atlas_spec
    periodic periodic_axes
}
```

The **atlas** and **cellsize** values are the same as for the `Oxs_RectangularMesh` class. The *periodic_axis* value should be a string consisting of one or more of the letters "x", "y", or "z", denoting the periodic direction(s). `Oxs_Ext` objects that are incompatible with `Oxs_PeriodicRectangularMesh` will issue an error message at runtime. For example,

the Oxs_Demag class supports periodicity for up to two directions, but not three. Also, some third-party extensions provide independent periodicity support using the older Oxs_RectangularMesh class rather than Oxs_PeriodicRectangularMesh.

Examples: pbcbrick.mif, pbcstripes.mif.

7.3.3 Energies

The following subsections describe the available energy terms. In order to be included in the simulation energy and field calculations, each energy term must be declared in its own, top-level Specify block, i.e., energy terms should not be declared inline inside other Oxs_Ext objects. There is no limitation on the number of energy terms that may be specified in the input MIF file. Many of these terms have spatially varying parameters that are initialized via *field_object_spec* in their Specify initialization block.

Outputs: For each magnetization configuration, three standard outputs are provided by all energy terms: the scalar output "Energy," which is the total energy in joules contributed by this energy term, the scalar field output "Energy density," which is a cell-by-cell map of the energy density in J/m^3, and the three-component vector field output "Field," which is the pointwise field in A/m. If the code was compiled with the macro NDEBUG not defined, then there will be an additional scalar output, "Calc count," which counts the number of times the term has been calculated in the current simulation. This is intended for debugging purposes only; this number should agree with the "Energy calc count" value provided by the evolver.

Anisotropy Energy

Oxs_UniaxialAnisotropy: Uniaxial magneto-crystalline anisotropy. The Specify block has the form

```
Specify Oxs_UniaxialAnisotropy:name {
    K1 K
    Ha H
    axis u
}
```

Exactly one of either **K1** or **Ha** should be specified, where **K1** is the crystalline anisotropy constant (in J/m^3), and **Ha** is the anistropy field (in A/m). In either case, **axis** is the anisotropy direction. **K1**, **Ha**, and **axis** may each be varied cellwise across the mesh: **K1** and **Ha** are initialized with scalar field objects, while **axis** takes a vector field object. (A constant value will be interpreted as a uniform field object having the stated value, as usual.) The axis direction must be non-zero at each point, and will be normalized to unit magnitude before being used.

The axis direction is an easy axis if **K1** (or **Ha**) is >0, in which case the cellwise anisotropy energy density (in J/m^3) is given by

$$E_i = K_i(1 - \mathbf{m}_i \cdot \mathbf{u}_i)^2 \qquad \text{or} \qquad \frac{1}{2}\,\mu_0 M_s H_i (1 - \mathbf{m}_i \cdot \mathbf{u}_i)^2,$$

respectively. (Here m_i is the unit magnetization and M_s the saturation magnetization in cell i.) Otherwise, if **K1** (or **Ha**) is < 0, the axis direction is the normal to the easy plane and the cellwise anisotropy energy density is given by

$$E_i = -K_i(\mathbf{m}_i \cdot \mathbf{u}_i)^2 \qquad \text{or} \qquad -\frac{1}{2}\,\mu_0 M_s H_i (\mathbf{m}_i \cdot \mathbf{u}_i)^2.$$

The formulae in the two cases (easy axis vs. easy plane) differ by a constant offset, and in each case the energy is non-negative.

Examples: `diskarray.mif`, `stdprob3.mif`, `grill.mif`.

Oxs_CubicAnisotropy: Cubic magneto-crystalline anisotropy. The Specify block has the form

```
Specify Oxs_CubicAnisotropy:name {
    K1 K
    Ha H
    axis1 u1
    axis2 u2
}
```

Exactly one of either **K1** or **Ha** should be specified, where **K1** is the crystalline anisotropy constant (in J/m^3), and **Ha** is the anistropy field (in A/m). In either case, **axis1** and **axis2** are two anisotropy directions; the third anisotropy axis u_3 is computed as the vector product, $u_1 \times u_2$. For each cell, the axis directions are easy axes if **K1** (or **Ha**) is >0, or hard axes if **K1** (or **Ha**) is <0. All may be varied cellwise across the mesh. **K1** or **Ha** is initialized with a scalar field object, and the axis directions are initialized with vector field objects. (Constant values will be interpreted as uniform fields with the indicated value, as usual.) The **axis1** and **axis2** directions must be mutually orthogonal and non-zero at each point (u_1 and u_2 are automatically scaled to unit magnitude before use).

The anisotropy energy density (in J/m^3) for cell i is given by

$$E_i = K_i \left(a_1^2 a_2^2 + a_2^2 a_3^2 + a_3^2 a_1^2\right),$$

or

$$E_i = \frac{1}{2}\,\mu_0 M_s H_i \left(a_1^2 a_2^2 + a_2^2 a_3^2 + a_3^2 a_1^2\right),$$

where $a_1 = \mathbf{m} \cdot \mathbf{u}_1$, $a_2 = \mathbf{m} \cdot \mathbf{u}_2$, $a_3 = \mathbf{m} \cdot \mathbf{u}_3$, for reduced (normalized) magnetization m and orthonormal anisotropy axes $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ at cell i. In the second form, M_s is the saturation magnetization in cell i. For each cell, if **K1** (resp. **Ha**) is >0 then the computed energy will be non-negative, otherwise for **K1** (resp. **Ha**) <0 the computed energy will be non-positive.

Examples: `cgtest.mif`, `sample2.mif`, `grill.mif`.

Exchange Energy

Oxs_Exchange6Ngbr: Standard 6-neighbor exchange energy. The exchange energy density contribution from cell i is given by

$$E_i = \sum_{j \in N_i} A_{ij} \frac{\mathbf{m}_i \cdot (\mathbf{m}_i - \mathbf{m}_j)}{\Delta_{ij}^2} \tag{7.1}$$

where N_i is the set consisting of the 6 cells nearest to cell i, A_{ij} is the exchange coefficient between cells i and j in J/m, and Δ_{ij} is the discretization step size between cell i and cell j (in meters).

The Specify block for this term has the form

```
Specify Oxs_Exchange6Ngbr:name {
    default_A value
    atlas atlas_spec
    A {
        region-1 region-1 A11
        region-1 region-2 A12
        ...
        region-m region-n Amn
    }
}
```

or

```
Specify Oxs_Exchange6Ngbr:name {
    default_lex value
    atlas atlas_spec
    lex {
        region-1 region-1 lex11
        region-1 region-2 lex12
        ...
        region-m region-n lexmn
    }
}
```

where **lex** specifies the magnetostatic-exchange length, in meters, defined by lex = $\sqrt{2A/(\mu_0 M_s^2)}$.

In the first case, the **A** block specifies A_{ij} values on a region by region basis, where the regions are labels declared by *atlas_spec*. This allows for specification of A both inside a given region (e.g., A_{ii}) and along interfaces between regions (e.g., A_{ij}). By symmetry, if A_{ij} is specified, then the same value is automatically assigned to A_{ji} as well. The **default_A** value is applied to any otherwise unassigned A_{ij}.

In the second case, one specifies the magnetostatic-exchange length instead of A, but the interpretation is otherwise analogous.

Although one may specify A_{ij} (resp. lex_{ij}) for any pair of regions i and j, it is only required and only active if the region pair are in contact. If long-range exchange interaction is required, use `Oxs_TwoSurfaceExchange`.

In addition to the standard energy and field outputs, `Oxs_Exchange6Ngbr` provides three scalar outputs involving the angle between spins at neighboring cells:

- **Max Spin Ang:** maximum angle, in degrees, between neigboring spins for the current magnetization state. Two spins i and j are neighbors if A_{ij} is non-zero. If $A_{ij} < 0$ then `Max Spin Ang` is measured as the difference from 180°, because in that case the low-energy state has the spins anti-parallel.
- **Stage Max Spin Ang:** Maximum value of `Max Spin Ang` for the current stage.
- **Run Max Spin Ang:** Maximum value obtained by `Max Spin Ang` during the simulation.

These values are also accessible inside the MIF file through the `GetStateData` command.

Examples: `grill.mif`, `spinvalve.mif`, `tclshapes.mif`.

Oxs_UniformExchange: Similar to `Oxs_Exchange6Ngbr`, except the exchange constant A (or exchange length lex) is uniform across all space. The Specify block is very simple, consisting of either the label **A** with the desired exchange coefficient value in J/m, or the label **lex** with the desired magnetostatic-exchange length in meters. Since `A` (resp. `lex`) is not spatially varying, it is initialized with a simple constant (as opposed to a scalar field object).

In addition to the standard energy and field outputs, `Oxs_UniformExchange` provides the three scalar outputs `Max Spin Ang`, `Stage Max Spin Ang`, and `Run Max Spin Ang` as described for `Oxs_Exchange6Ngbr`. These values are also accessible inside the MIF file through the `GetStateData` command.

Examples: `sample.mif`, `cgtest.mif`, `stdprob3.mif`.

Oxs_ExchangePtwise: The exchange coefficient A_i is specified on a point-by-point (or cell-by-cell) basis, as opposed to the pairwise model used by `Oxs_Exchange6Ngbr`. The exchange energy density at a cell i is computed across its nearest 6 neighbors, N_i, using the formula

$$E_i = \sum_{j \in N_i} A_{ij,\text{eff}} \frac{\mathbf{m}_i \cdot (\mathbf{m}_i - \mathbf{m}_j)}{\Delta_{ij}^2}$$

where Δ_{ij} is the discretization step size from cell i to cell j in meters, and

$$A_{ij,\text{eff}} = \frac{2A_iA_j}{A_i + A_j},$$

with $A_{ij,\text{eff}} = 0$ if A_i and A_j are 0.

Note that $A_{ij,\text{eff}}$ satisfies the following properties:

$$\begin{aligned} A_{ij,\text{eff}} &= A_{ji,\text{eff}} \\ A_{ij,\text{eff}} &= A_i \qquad \text{if } A_i = A_j \\ \lim_{A_i \downarrow 0} A_{ij,\text{eff}} &= 0. \end{aligned}$$

Additionally, if A_i and A_j are non-negative,

$$\min(A_i, A_j) \le A_{ij,\text{eff}} \le \max(A_i, A_j).$$

Evaluating the exchange energy with this formulation of $A_{ij,\text{eff}}$ is equivalent to finding the minimum possible exchange energy between cells i and j under the assumption that A_i and A_j are constant in each of the two cells. Similar considerations are made in computing the exchange energy for a 2D variable thickness model [26].

The Specify block for `Oxs_ExchangePtwise` has the form

```
Specify Oxs_ExchangePtwise:name {
    A scalarfield_spec
}
```

where *scalarfield_spec* is an arbitrary scalar field object returning the desired exchange coefficient in J/m.

In addition to the standard energy and field outputs, `Oxs_ExchangePtwise` provides the three scalar outputs `Max Spin Ang`, `Stage Max Spin Ang`, and `Run Max Spin Ang` as described for `Oxs_Exchange6Ngbr`. These values are also accessible inside the MIF file through the `GetStateData` command.

Example: `antidots-filled.mif`.

Oxs_RandomSiteExchange: A randomized exchange energy. The Specify block has the form

```
Specify Oxs_RandomSiteExchange:name {
    linkprob probability
    Amin A_lower_bound
    Amax A_upper_bound
}
```

Each adjacent pair of cells i, j, is given **linkprob** probability of having a non-zero exchange coefficient A_{ij}. Here two cells are adjacent if they lie in each other's 6-neighborhood. If a pair is found to have a non-zero exchange coefficient, then A_{ij} is drawn uniformly from the range [`Amin`, `Amax`]. The exchange energy is computed using (7.1), the formula used by the `Oxs_Exchange6Ngbr` energy object. The value A_{ij} for each pair of cells is determined during problem initialization, and is held fixed thereafter. The limits *A_lower_bound* and *A_upper_bound* may be any real numbers; negative values may be used to weaken the exchange interaction arising from other exchange energy terms. The only restriction is that *A_lower_bound* must not be greater than *A_upper_bound*. The `linkprob` value *probability* must lie in the range $[0, 1]$.

In addition to the standard energy and field outputs, `Oxs_RandomSiteExchange` provides the three scalar outputs `Max Spin Ang`, `Stage Max Spin Ang`, and `Run Max Spin Ang` as described for `Oxs_Exchange6Ngbr`. These values are also accessible inside the MIF file through the `GetStateData` command.

Example: `randexch.mif`.

Oxs_TwoSurfaceExchange: Provides long-range bilinear and biquadratic exchange. Typically used to simulate RKKY-style coupling across non-magnetic spacers in spinvalves. The specify block has the form

```
Specify Oxs_TwoSurfaceExchange:name {
    sigma value
    sigma2 value
    surface1 {
        atlas atlas_spec
        region region_label
        scalarfield scalarfield_spec
        scalarvalue fieldvalue
        scalarside side
    }
    surface2 {
        atlas atlas_spec
        region region_label
        scalarfield scalarfield_spec
        scalarvalue fieldvalue
```

```
        scalarside side
    }
    report_max_spin_angle report_angle
}
```

Here **sigma** and **sigma2** are the bilinear and biquadratic surface (interfacial) exchange energies, in J/m^2. Either is optional, with default value 0.

The **surface1** and **surface2** sub-blocks describe the two interacting surfaces. Each description consists of 5 name-values pairs, which must be listed in the order shown. In each sub-block, *atlas_spec* specifies an atlas, and *region_label* specifies a region in that atlas. These bound the extent of the desired surface. The following **scalarfield**, **scalarvalue** and **scalarside** entries define a discretized surface inside the bounding region. Here *scalarfield_spec* references a scalar field object, *fieldvalue* should be a floating point value, and *side* should be one of `<`, `<=`, `>=`, or `>`. Any point for which the scalar field object takes a value less than, less than or equal, greater than or equal, or greater than, respectively, the `scalarvalue` value is considered to be "inside" the surface. (Values `-` and `+` for *side* are deprecated synonyms for `<=` and `>=`.) The discretized surface determined is the set of all points on the problem mesh that are in the bounding region, are "inside" the surface, and have a (nearest-) neighbor that is "outside" (i.e., not inside) the surface. A neighbor is determined by the mesh; in a typical rectangular mesh each cell has six neighbors.

In this way, 2 discrete lists of cells representing the two surfaces are obtained. Each cell from the first list (representing `surface1`) is then matched with the closest cell from the second list (i.e., from `surface2`). Note the asymmetry in this matching process: each cell from the first list is included in exactly one match, but there may be cells in the second list that are included in many match pairs, or in none. If the two surfaces are of different sizes, then in practice typically the smaller will be made the first surface, because this will usually lead to fewer multiply-matched cells, but this designation is not required.

The resulting exchange energy density at cell i on one surface from matching cell j on the other is given by

$$E_{ij} = \frac{\sigma\left[1 - \mathbf{m}_i \cdot \mathbf{m}_j\right] + \sigma_2\left[1 - (\mathbf{m}_i \cdot \mathbf{m}_j)^2\right]}{\Delta_{ij}}$$

where σ and σ_2, respectively, are the bilinear and biquadratic surface exchange coefficients between the two surfaces, in J/m^2, $\mathbf{m}_i$ and $\mathbf{m}_j$ are the normalized, unit spins (i.e., magnetization directions) at cells i and j, and Δ_{ij} is the discretization cell size in the direction from cell i towards cell j, in meters. Note that if σ is negative, then the surfaces will be anti-ferromagnetically coupled. Likewise, if σ_2 is negative, then the biquadratic term will favor orthogonal alignment.

The following example produces an antiferromagnetic exchange coupling between the lower surface of the "top" layer and the upper surface of the "bottom" layer, across a middle "spacer" layer. The simple `Oxs_LinearScalarField` object is used here to provide level surfaces that are planes orthogonal to the z-axis. In practice this example might represent a spinvalve, where the top and bottom layers would be composed of ferromagnetic material and the middle layer could be a copper spacer.

```
Specify Oxs_MultiAtlas:atlas {
    atlas { Oxs_BoxAtlas {
        name top
        xrange {0 500e-9}
        yrange {0 250e-9}
        zrange {6e-9 9e-9}
    } }
    atlas { Oxs_BoxAtlas {
        name spacer
        xrange {0 500e-9}
        yrange {0 250e-9}
        zrange {3e-9 6e-9}
    } }
    atlas { Oxs_BoxAtlas {
        name bottom
        xrange {0 500e-9}
        yrange {0 250e-9}
        zrange {0 3e-9}
    } }
}

Specify Oxs_LinearScalarField:zheight {
    vector {0 0 1}
    norm   1.0
}

Specify Oxs_TwoSurfaceExchange:AF {
    sigma -1e-4
    surface1 {
               atlas  :atlas
              region  bottom
         scalarfield  :zheight
         scalarvalue  3e-9
          scalarside  <=
    }
    surface2 {
```

```
                atlas  :atlas
               region  top
          scalarfield  :zheight
          scalarvalue  6e-9
           scalarside  >=
    }
}
```

In addition to the standard energy and field outputs, `Oxs_TwoSurfaceExchange` provides three optional scalar outputs, `Max Spin Ang`, `Stage Max Spin Ang`, and `Run Max Spin Ang`, as described for `Oxs_Exchange6Ngbr`. Note, however, that "neighboring spins" are those that interact through exchange coupling. For `Oxs_TwoSurfaceExchange` this means spins that are not necessarily spatially close, and so the maximum "neighbor" spin angle is less meaningful in this context. These outputs are disabled by default, but can be activated by setting the **report_max_spin_angle** option to 1. These values are always accessible inside the MIF file via the `GetStateData` command, regardless of the **report_max_spin_angle** setting.

Example: `spinvalve-af.mif`.

Self-Magnetostatic Energy

Oxs_Demag: Standard demagnetization energy term, built upon the assumption that the magnetization is constant in each cell. It computes the average demagnetization field in each cell using formulae from [2, 25, 11] and convolution via the Fast Fourier Transform. This class supports non-periodic simulations if the mesh object in the MIF file is of the `Oxs_RectangularMesh` type; for simulations periodic along one or two coordinate axes direction use the `Oxs_PeriodicRectangularMesh` class. The specify block has the form

```
Specify Oxs_Demag:name {
    asymptotic_order error_order
    demag_tensor_error relerror
}
```

The demag kernel is computed using a combination of analytic formulae for near field terms, high-order asymptotic formulae for far field terms, and summed subdivided cell asymptotic formulae for midrange terms, where the offset R between cell pairs determines the field range (based on extensions of earlier work[20]). The transition R values are selected to give the best computation speed while meeting the error requested by *relerror*. The demag kernel computation is a one-time operation performed during problem initialization, so the kernel computation time is generally of relatively minor concern, and accordingly the default value for *relerror* is 1e-15, i.e., nearly full double-precision accuracy.

Asymptotic formulae are used to compute the demag kernel for larger cell offset pair distances R. By default an expansion with error $\mathcal{O}(1/R^{11})$ is used, but lower orders can be requested through the *error_order* option. Valid values for *error_order* are 5, 7, 9, and 11, where *error_order*=5 is the dipole approximation.

There is also backward support for the now deprecated option **asymptotic_radius**, which set the cutoff between the analytic and asymptotic computation forms in units of cells. If **asymptotic_radius** is specified then it is converted to a more-or-less equivalent value for *relerror*, with the special values of 0 and -1 mapping to *relerror*=1 and 1e-16, respectively.

The example file `demagtensor.mif` can be used to extract the computed demagnetization tensor coefficients for a specified cell geometry; see the description at the top of that file for usage details.

Examples: `sample.mif`, `cgtest.mif`, `pbcbrick.mif`, `demagtensor.mif`.

Oxs_SimpleDemag: This is the same as the `Oxs_Demag` object, except that periodicity is not supported and asymptotic formulae are not used. The implementation does not use any of the symmetries inherent in the demagnetization kernel, or special properties of the Fourier Transform when applied to a real (non-complex) function. As a result, the source code is considerably simpler than for `Oxs_Demag`, but the run time performance and memory usage are poorer. `Oxs_SimpleDemag` is included for validation checks, and as a base for user-defined demagnetization implementations. The Specify initialization string for `Oxs_SimpleDemag` is an empty string, i.e., {}.

Example: `squarecubic.mif`.

Zeeman Energy

Oxs_UZeeman: Uniform (homogeneous) applied field energy. This class is frequently used for simulating hysteresis loops. The specify block takes an optional **multiplier** entry, and a required field range list **Hrange**. The field range list should be a compound list, with each sublist consisting of 7 elements: the first 3 denote the x, y, and z components of the start field for the range, the next 3 denote the x, y, and z components of the end field for the range, and the last element specifies the number of (linear) steps through the range. If the step count is 0, then the range consists of the start field only. If the step count is bigger than 0, then the start field is skipped over if and only if it is the same field that ended the previous range (if any).

The fields specified in the range entry are nominally in A/m, but these values are multiplied by `multiplier`, which may be used to effectively change the units. For example,

```
Specify Oxs_UZeeman {
    multiplier 795.77472
```

```
    Hrange {
        { 0 0 0 10 0 0 2 }
        { 10 0 0 0 0 0 1 }
    }
}
```

The applied field steps between 0 mT, 5 mT, 10 mT and back to 0 mT, for four stages in total. If the first field in the second range sublist was different from the second field in the first range sublist, then a step would have been added between those field values, so five stages would have resulted. In this example, note that 795.77472=$0.001/\mu_0$.

In addition to the standard energy and field outputs, the `Oxs_UZeeman` class provides these four scalar outputs:

- **B:** Magnitude of the applied field, in mT. This is a non-negative quantity.
- **Bx:** Signed amplitude of the x-component of the applied field, in mT.
- **By:** Signed amplitude of the y-component of the applied field, in mT.
- **Bz:** Signed amplitude of the z-component of the applied field, in mT.

Examples: `sample.mif`, `cgtest.mif`, `marble.mif`.

Oxs_FixedZeeman: Non-uniform, non-time varying applied field. This can be used to simulate a biasing field. The specify block takes one required parameter, which defines the field, and one optional parameter, which specifies a multiplication factor.

```
Specify Oxs_FixedZeeman:name {
    field vector_field_spec
    multiplier multiplier
}
```

The default value for *multiplier* is 1. The field units, after scaling by *multiplier*, should be A/m.

Examples: `spinvalve.mif`, `spinvalve-af.mif`, `yoyo.mif`.

Oxs_ScriptUZeeman: Spatially uniform applied field, potentially varying as a function of time and stage, determined by a Tcl script. The Specify block has the form

```
Specify Oxs_ScriptUZeeman:name {
    script_args { args_request }
    script Tcl_script
    multiplier multiplier
    stage_count number_of_stages
}
```

Here **script** indicates the Tcl script to use. The script is called once each iteration. Appended to the script are the arguments requested by **script_args**, in the manner explained in the User Defined Support Procedures of the MIF 2 file format documentation. The value *args_request* should be a subset of {`stage stage_time total_time base_state_id current_state_id`}. The units for the time options are seconds. The two `state_id` options are intended for use with the MIF `GetStateData` command; refer to the documentation on that command in the MIF 2.1 section for details. If `script_args` is not specified, the default argument list is {`stage stage_time total_time`}.

The return value from the script should be a 6-tuple of numbers, {`Hx`, `Hy`, `Hz`, `dHx`, `dHy`, `dHz`}, representing the applied field and the time derivative of the applied field. The field as a function of time must be differentiable for the duration of each stage. Discontinuities are permitted between stages. If a time evolver is being used, then it is very important that the time derivative values are correct; otherwise the evolver will not function properly. This usual symptom of this problem is a collapse in the time evolution step size.

The field and its time derivative are multiplied by the **multiplier** value before use. The final field value should be in A/m; if the Tcl script returns the field in T, then a `multiplier` value of $1/\mu_0$ (approx. 795774.72) should be applied to convert the Tcl result into A/m. The default value for `multiplier` is 1.

The **stage_count** parameter informs the `Oxs_Driver` as to how many stages the `Oxs_ScriptUZeeman` object wants. A value of 0 (the default) indicates that the object is prepared for any range of stages. The `stage_count` value given here must be compatible with the `stage_count` setting in the driver Specify block (page 92).

The following example produces a sinusoidally varying field of frequency 1 GHz and amplitude 800 A/m, directed along the x-axis.

```
proc SineField { total_time } {
    set PI [expr {4*atan(1.)}]
    set Amp 800.0
    set Freq [expr {1e9*(2*$PI)}]
    set Hx [expr {$Amp*sin($Freq*$total_time)}]
    set dHx [expr {$Amp*$Freq*cos($Freq*$total_time)}]
    return [list $Hx 0 0 $dHx 0 0]
}

Specify Oxs_ScriptUZeeman {
   script_args total_time
   script SineField
}
```

In addition to the standard energy and field outputs, the `Oxs_ScriptUZeeman` class

provides these four scalar outputs:

- **B:** Magnitude of the applied field, in mT. This is a non-negative quantity.
- **Bx:** Signed amplitude of the x-component of the applied field, in mT.
- **By:** Signed amplitude of the y-component of the applied field, in mT.
- **Bz:** Signed amplitude of the z-component of the applied field, in mT.

Examples: `acsample.mif`, `pulse.mif`, `rotate.mif`, `varalpha.mif`, `yoyo.mif`.

Oxs_TransformZeeman: A Tcl script specifies a spatially uniform, but time and stage varying linear transform that is applied to a spatially varying but temporally static field. This essentially combines the `Oxs_ScriptUZeeman` and `Oxs_FixedZeeman` classes.

The Specify block has the form

```
Specify Oxs_TransformZeeman:name {
    field vector_field_spec
    type transform_type
    script Tcl_script
    script_args { args_request }
    multiplier multiplier
    stage_count number_of_stages
}
```

The **field** specified by *vector_field_spec* is evaluated during problem initialization and held throughout the life of the problem. On each iteration, the specified Tcl **script** is called once. Appended to the script are the arguments requested by **script_args**, as explained in the User Defined Support Procedures of the MIF 2 file format documentation. The value for `script_args` should be a subset of {`stage stage_time total_time`}. The default value for `script_args` is the complete list in the aforementioned order. The time arguments are specified in seconds.

The script return value should define a 3x3 linear transform and its time derivative. The transform must be differentiable with respect to time throughout each stage, but is allowed to be discontinuous between stages. As noted in the `Oxs_ScriptUZeeman` documentation, it is important that the derivative information be correct. The transform is applied pointwise to the fixed field obtained from *vector_field_spec*, which is additionally scaled by *multiplier*. The **multiplier** entry is optional, with default value 1.0.

The **type** *transform_type* value declares the format of the result returned from the Tcl script. Recognized formats are `identity`, `diagonal`, `symmetric` and `general`. The most flexible is `general`, which indicates that the return from the Tcl script is a list of 18 numbers, defining a general 3x3 matrix and its 3x3 matrix of time derivatives. The matrices are specified in row-major order, i.e., $M_{1,1}$, $M_{1,2}$, $M_{1,3}$, $M_{2,1}$, $M_{2,2}$, ... Of

course, this is a long list to construct; if the desired transform is symmetric or diagonal, then the `type` may be set accordingly to reduce the size of the Tcl result string. Scripts of the `symmetric` type return 12 numbers, the 6 upper diagonal entries in row-major order, i.e., $M_{1,1}$, $M_{1,2}$, $M_{1,3}$, $M_{2,2}$, $M_{2,3}$, $M_{3,3}$, for both the transformation matrix and its time derivative. Use the `diagonal` type for diagonal matrices, in which case the Tcl script result should be a list of 6 numbers.

The simplest *transform_type* is `identity`, which is the default. This identifies the transform as the identity matrix, which means that effectively no transform is applied, aside from the `multiplier` option which is still active. For the `identity` transform type, `script` and `script_args` should not be specified, and `Oxs_TransformZeeman` becomes a clone of the `Oxs_FixedZeeman` class.

The following example produces a 1000 A/m field that rotates in the xy-plane at a frequency of 1 GHz:

```
proc Rotate { freq stage stagetime totaltime } {
   global PI
   set w [expr {$freq*2*$PI}]
   set ct [expr {cos($w*$totaltime)}]
   set mct [expr {-1*$ct}]      ;# "mct" is "minus cosine (w)t"
   set st [expr {sin($w*$totaltime)}]
   set mst [expr {-1*$st}]      ;# "mst" is "minus sine (w)t"
   return [list  $ct $mst  0 \
                 $st $ct   0 \
                   0   0   1 \
                 [expr {$w*$mst}] [expr {$w*$mct}] 0 \
                 [expr {$w*$ct}]  [expr {$w*$mst}] 0 \
                        0                 0         0]
}

Specify Oxs_TransformZeeman {
  type general
  script {Rotate 1e9}
  field {0 1000. 0}
}
```

This particular effect could be obtained using the `Oxs_ScriptUZeeman` class, because the `field` is uniform. But the field was taken uniform only to simplify the example. The *vector_field_spec* may be any Oxs vector field object. For example, the base field could be large in the center of the sample, and decay towards the edges. In that case, the above example would generate an applied rotating field that is concentrated in the center of the sample.

The **stage_count** parameter informs the `Oxs_Driver` as to how many stages the

`Oxs_TransformZeeman` object wants. A value of 0 (the default) indicates that the object is prepared for any range of stages. The `stage_count` value given here must be compatible with the `stage_count` setting in the driver Specify block (page 92).

In addition to the standard energy and field outputs, the `Oxs_TransformZeeman` class provides these four scalar outputs:

- **Bx:** Average value of the x-component of the applied field in mT.
- **By:** Average value of the y-component of the applied field in mT.
- **Bz:** Average value of the z-component of the applied field in mT.
- **B:** Magnitude of the applied field average in mT, i.e., $\mathrm{B} = \sqrt{(\mathrm{Bx})^2 + (\mathrm{By})^2 + (\mathrm{Bz})^2}$.

Note: If the vector average of the **field** specified by *vector_field_spec* is **0** (due for example to symmetries) then the above outputs will all be 0 (up to rounding error) throughout the simulation run.

Examples: `sample2.mif`, `tickle.mif`, `rotatecenter.mif`.

Oxs_StageZeeman: The `Oxs_StageZeeman` class provides spatially varying applied fields that are updated once per stage. In its general form, the field at each stage is provided by an Oxs vector field object determined by a user supplied Tcl script. There is also a simplified interface that accepts a list of vector field files, one per stage, that are used to specify the applied field.

The Specify block takes the form

```
Specify Oxs_StageZeeman:name {
    script Tcl_script
    files { list_of_files }
    stage_count number_of_stages
    multiplier multiplier
}
```

The initialization string should specify either `script` or `files`, but not both. If a **script** is specified, then each time a new stage is started in the simulation, a Tcl command is formed by appending to *Tcl_script* the 0-based integer stage number. This command should return a reference to an `Oxs_VectorField` object, as either the instance name of an object defined via a top-level Specify block elsewhere in the MIF file, or as a two item list consisting of the name of an `Oxs_VectorField` class and an appropriate initialization string. In the latter case the `Oxs_VectorField` object will be created as a temporary object via an inlined Specify call.

The following example should help clarify the use of the `script` parameter.

```
proc SlidingField { xcutoff xrel yrel zrel } {
   if {$xrel>$xcutoff} { return [list 0. 0. 0.] }
```

```
   return [list 2e4 0. 0.]
}

proc SlidingFieldSpec { stage } {
  set xcutoff [expr {double($stage)/10.}]
  set spec Oxs_ScriptVectorField
  lappend spec [subst {
      atlas :atlas
      script {SlidingField $xcutoff}
   }]
   return $spec
}

Specify Oxs_StageZeeman {
  script SlidingFieldSpec
  stage_count 11
}
```

The `SlidingFieldSpec` proc generates an initialization string that is used to create an `Oxs_ScriptVectorField` vector field object, which in turn uses the `SlidingField` proc to specify the applied field on a position-by-position basis. The resulting field will be 2×10^4 A/m in the positive x-direction at all points with relative x-coordinate larger than `$stage/10.`, and 0 otherwise. `$stage` is the stage index, which here is one of 0, 1, ..., 10. For example, if `$stage` is 5, then the left half of the sample will see a 2×10^4 A/m field directed to the right, and the right half of the sample will see none. The return value from `SlidingFieldSpec` in this case will be

```
Oxs_ScriptVectorField {
   atlas :atlas
   script {SlidingField 0.5}
}
```

The `:atlas` reference is to an `Oxs_Atlas` object defined elsewhere in the MIF file.

The **stage_count** parameter lets the `Oxs_Driver` know how many stages the `Oxs_StageZeeman` object wants. A value of 0 indicates that the object is prepared for any range of stages. Zero is the default value for `stage_count` when using the *Tcl_script* interface. The `stage_count` value given here must be compatible with the `stage_count` setting in the driver Specify block (page 92).

The example above made use of two scripts, one to specify the `Oxs_VectorField` object, and one used internally by the `Oxs_ScriptVectorField` object. But any `Oxs_VectorField` class may be used, as in the next example.

```
proc FileField { stage } {
```

```
  set filelist { field-a.ohf field-b.ohf field-c.ohf }
  set spec Oxs_FileVectorField
  lappend spec [subst {
      atlas :atlas
      file [lindex $filelist $stage]
   }]
   return $spec
}

Specify Oxs_StageZeeman {
  script FileField
  stage_count 3
}
```

The `FileField` proc yields a specification for an `Oxs_FileVectorField` object that loads one of three files, `field-a.ohf`, `field-b.ohf`, or `field-c.ohf`, depending on the stage number.

Specifying applied fields from a sequence of files is common enough to warrant a simplified interface. This is the purpose of the **files** parameter:

```
Specify Oxs_StageZeeman {
  files { field-a.ohf field-b.ohf field-c.ohf }
}
```

This is essentially equivalent to the preceding example, with two differences. First, `stage_count` is not needed because `Oxs_StageZeeman` knows the length of the list of files. You may specify `stage_count`, but the default value is the length of the `files` list. This is in contrast to the default value of 0 when using the `script` interface. If `stage_count` is set larger than the file list, then the last file is repeated as necessary to reach the specified size.

The second difference is that no `Oxs_Atlas` is specified when using the `files` interface. The `Oxs_FileVectorField` object spatially scales the field read from the file to match a specified volume. Typically a volume is specified by explicit reference to an atlas, but with the `files` interface to `Oxs_StageZeeman` the file fields are implicitly scaled to match the whole of the meshed simulation volume. This is the most common case; to obtain a different spatial scaling use the `script` interface as illustrated above with a different atlas or an explicit x/y/z-range specification.

The *list_of_files* value is interpreted as a *grouped list*. See the notes in Sec. 17.3.3 for details on grouped lists.

The remaining `Oxs_StageZeeman` parameter is **multiplier**. The value of this parameter is applied as a scale factor to the field magnitude on a point-by-point basis. For example, if the field returned by the `Oxs_VectorField` object were in Oe, instead of

the required A/m, then `multiplier` could be set to 79.5775 to perform the conversion. The direction of the applied field can be reversed by supplying a negative `multiplier` value.

In addition to the standard energy and field outputs, the `Oxs_StageZeeman` class provides these four scalar outputs:

- **B max:** Pointwise maximum magnitude of the applied field, in mT. This is a non-negative quantity; B max $= \sqrt{(\text{Bx max})^2 + (\text{By max})^2 + (\text{Bz max})^2}$.
- **Bx max:** Signed value of the x-component of the applied field at the point of maximum applied field magnitude, in mT.
- **By max:** Signed value of the y-component of the applied field at the point of maximum applied field magnitude, in mT.
- **Bz max:** Signed value of the z-component of the applied field at the point of maximum applied field magnitude, in mT.

Examples: `sliding.mif`, `slidingproc.mif`, `rotatestage.mif`, `rotatecenterstage.mif`.

7.3.4 Evolvers

Evolvers are responsible for updating the magnetization configuration from one step to the next. There are two types of evolvers, *time evolvers*, which track Landau-Lifshitz-Gilbert dynamics, and *minimization evolvers*, which locate local minima in the energy surface through direct minimization techniques. Evolvers are controlled by drivers, and must be matched with the appropriate driver type, i.e., time evolvers must be paired with time drivers, and minimization evolvers must be paired with minimization drivers. The drivers hand a magnetization configuration to the evolvers with a request to advance the configuration by one *step* (also called an *iteration*). It is the role of the drivers, not the evolvers, to determine when a simulation stage or run is complete. Specify blocks for evolvers contain parameters to control all aspects of individual stepwise evolution, but stopping criteria are communicated in the Specify block of the driver, not the evolver.

There are currently three time evolvers and one minimization evolver in the standard OOMMF distribution. The time evolvers are `Oxs_EulerEvolve`, `Oxs_RungeKuttaEvolve`, and `Oxs_SpinXferEvolve`. The minimization evolver is `Oxs_CGEvolve`.

Oxs_EulerEvolve: Time evolver implementing a simple first order forward Euler method with step size control on the Landau-Lifshitz ODE [13, 18]:

$$\frac{d\mathbf{M}}{dt} = -|\bar{\gamma}|\,\mathbf{M}\times\mathbf{H}_{\text{eff}} - \frac{|\bar{\gamma}|\alpha}{M_s}\,\mathbf{M}\times(\mathbf{M}\times\mathbf{H}_{\text{eff}})\,, \tag{7.2}$$

where $\mathbf{M}$ is the magnetization, $\mathbf{H}_{\text{eff}}$ is the effective field, $\bar{\gamma}$ is the Landau-Lifshitz gyromagnetic ratio, and α is the damping constant. The Gilbert form

$$\frac{d\mathbf{M}}{dt} = -|\gamma|\,\mathbf{M} \times \mathbf{H}_{\text{eff}} + \frac{\alpha}{M_s}\left(\mathbf{M} \times \frac{d\mathbf{M}}{dt}\right), \tag{7.3}$$

where γ is the Gilbert gyromagnetic ratio, is mathematically equivalent to the Landau-Lifshitz form under the relation $\gamma = (1+\alpha^2)\,\bar{\gamma}$.

The Specify block has the form

```
Specify Oxs_EulerEvolve:name {
    alpha α
    gamma_LL γ̄
    gamma_G γ
    do_precess precess
    min_timestep minimum_stepsize
    max_timestep maximum_stepsize
    fixed_spins {
        atlas_spec
        region1 region2 ...
    }
    start_dm Δm
    error_rate rate
    absolute_step_error abs_error
    relative_step_error rel_error
    step_headroom headroom
}
```

All the entries have default values, but the ones most commonly adjusted are listed first.

The options **alpha**, **gamma_LL** and **gamma_G** are as in the Landau-Lifshitz-Gilbert ODE (7.2), (7.3), where the units on $\bar{\gamma}$ and γ are m/A·s and α is dimensionless. At most one of $\bar{\gamma}$ and γ should be specified. If neither is specified, then the default is $\gamma = 2.211 \times 10^5$. (Because of the absolute value convention adopted on $\bar{\gamma}$ and γ in (7.2), (7.3), the sign given to the value of `gamma_LL` or `gamma_G` in the Specify block is irrelevant.) The default value for α is 0.5, which is large compared to experimental values, but allows simulations to converge to equilibria in a reasonable time. However, for accurate dynamic studies it is important to assign an appropriate value to α.

The **do_precess** value should be either 1 or 0, and determines whether or not the precession term in the Landau-Lifshitz ODE (i.e., the first term on the righthand side in (7.2)) is used. If *precess* is 0, then precession is disabled and the simulation evolves towards equilibrium along a steepest descent path. The default value is 1.

The **min_timestep** and **max_timestep** parameters provide soft limits on the size of steps taken by the evolver. The minimum value may be overridden by the driver if a smaller step is needed to meet time based stopping criteria. The maximum value will be ignored if a step of that size would produce a magnetization state numerically indistinguishable from the preceding state. The units for `min_timestep` and `max_timestep` are seconds. Default values are 0 and 10^{-10} respectively.

The optional **fixed_spins** entry allows the magnetization in selected regions of the simulation to be frozen in its initial configuration. The value portion of the entry should be a list, with the first element of the list being either an inline atlas definition (grouped as a single item), or else the name of a previously defined atlas. The remainder of the list are names of regions in that atlas for which the magnetization is to be fixed, i.e., $\mathbf{M}(t) = \mathbf{M}(0)$ for all time t for all points in the named regions. Fields and energies are computed and reported normally across these regions. Although any atlas may be used, it is frequently convenient to set up an atlas with special regions defined expressly for this purpose.

The stepsize for the first candidate iteration in the problem run is selected so that the maximum change in the normalized (i.e., unit) magnetization $\mathbf{m}$ is the value specified by **start_dm**. The units are degrees, with default value 0.01.

The remaining entries, **error_rate**, **absolute_step_error**, **relative_step_error**, and **step_headroom**, control fine points of stepsize selection, and are intended for advance use only. Given normalized magnetization $\mathbf{m}_i(t)$ at time t and position i, and candidate magnetization $\mathbf{m}_i(t+\Delta t)$ at time $t+\Delta t$, the error at position i is estimated to be

$$\text{Error}_i = |\dot{\mathbf{m}}_i(t+\Delta t) - \dot{\mathbf{m}}_i(t)|\,\Delta t\,/\,2,$$

where the derivative with respect to time, $\dot{\mathbf{m}}$, is computed using the Landau-Lifshitz ODE (7.2). First order methods essentially assume that $\dot{\mathbf{m}}$ is constant on the interval $[t, t+\Delta t]$; the above formula uses the difference in $\dot{\mathbf{m}}$ at the endpoints of the interval to estimate (guess) how untrue that assumption is.

A candidate step is accepted if the maximum error across all positions i is smaller than all three of the error controls `absolute_step_error`, `error_rate` $\times\,\Delta t$, and `relative_step_error` $\times\,|\dot{\mathbf{m}}_{\max}|\Delta t$, where $|\dot{\mathbf{m}}_{\max}|$ is the maximum value of $|\dot{\mathbf{m}}_i|$ across all i at time t. If the step is rejected, then a smaller stepsize is computed that appears to pass the above tests, and a new candidate step is proposed using that smaller stepsize times `step_headroom`. Alternatively, if the step is accepted, then the error information is used to determine the stepsize for the next step, modified in the same manner by `step_headroom`.

The error calculated above is in terms of unit magnetizations, so the natural units are radians or radians/second. Inside the Specify block, however, the `error_rate` and `absolute_step_error` are specified in degrees/nanosecond and degrees, respectively; they are converted appropriately inside the code before use. The `relative_step_error`

is a dimensionless quantity, representing a proportion between 0 and 1. The error check controlled by each of these three quantities may be disabled by setting the quantity value to -1. They are all optional, with default values of -1 for `error_rate`, 0.2 for `absolute_step_error`, and 0.2 for `relative_step_error`.

The `headroom` quantity should lie in the range $(0, 1)$, and controls how conservative the code will be in stepsize selection. If `headroom` is too large, then much computation time will be lost computing candidate steps that fail the error control tests. If `headroom` is small, then most candidate steps will pass the error control tests, but computation time may be wasted calculating more steps than are necessary. The default value for `headroom` is 0.85.

In addition to the above error control tests, a candidate step will also be rejected if the total energy, after adjusting for effects due to any time varying external field, is found to increase. In this case the next candidate stepsize is set to one half the rejected stepsize.

The `Oxs_EulerEvolve` module provides five scalar, one scalar field, and three vector field outputs. The scalar outputs are

- **Max dm/dt:** maximum $|d\mathbf{m}/dt|$, in degrees per nanosecond; $\mathbf{m}$ is the unit magnetization direction.
- **Total energy:** in joules.
- **Delta E:** change in energy between last step and current step, in joules.
- **dE/dt:** derivative of energy with respect to time, in joules per second.
- **Energy calc count:** number of times total energy has been calculated.

The scalar field output is

- **Total energy density:** cellwise total energy density, in $\mathrm{J/m^3}$.

The vector field outputs are

- **Total field:** total effective field $\mathbf{H}$ in A/m.
- **mxH:** torque in A/m; $\mathbf{m}$ is the unit magnetization direction, $\mathbf{H}$ is the total effective field.
- **dm/dt:** derivative of spin $\mathbf{m}$ with respect to time, in radians per second.

Example: `octant.mif`.

Oxs_RungeKuttaEvolve: Time evolver implementing several Runge-Kutta methods for integrating the Landau-Lifshitz-Gilbert ODE (7.2), (7.3), with step size control. In most cases it will greatly outperform the `Oxs_EulerEvolve` class. The Specify block has the form

```
Specify Oxs_RungeKuttaEvolve:name {
    alpha α
    gamma_LL γ̄
    gamma_G γ
    do_precess precess
    allow_signed_gamma signed_gamma
    min_timestep minimum_stepsize
    max_timestep maximum_stepsize
    fixed_spins {
        atlas_spec
        region1 region2 ...
    }
    start_dm Δm
    start_dt start_timestep
    stage_start scontinuity
    error_rate rate
    absolute_step_error abs_error
    relative_step_error rel_error
    energy_precision eprecision
    min_step_headroom min_headroom
    max_step_headroom max_headroom
    reject_goal reject_proportion
    method subtype
}
```

Most of these options appear also in the `Oxs_EulerEvolve` class. The repeats have the same meaning as in that class, and the same default values except for `error_rate` and `relative_step_error`, which for `Oxs_RungeKuttaEvolve` have default values of 1.0 and 0.01, respectively. Additionally, the **alpha**, **gamma_LL** and **gamma_G** options may be initialized using scalar field objects, to allow these material parameters to vary spatially.

The **allow_signed_gamma** parameter is for simulation testing purposes, and is intended for advanced use only. There is some lack of consistency in the literature with respect to the sign of γ. For this reason the Landau-Lifshitz-Gilbert equations are presented above (7.2, 7.3) using the absolute value of γ. This is the interpretation used if `allow_signed_gamma` is 0 (the default). If instead `allow_signed_gamma` is set to 1, then the Landau-Lifshitz-Gilbert equations are interpreted without the absolute values and with a sign change on the γ terms, i.e., the default value for γ in this case is -2.211×10^5 (units are m/A·s). In this setting, if γ is set positive then the spins will precess backwards about the effective field, and the damping term will force the spins **away** from the effective field and increase the total energy. If you are experimenting with $\gamma > 0$, you should either set $\alpha \leq 0$ to force spins back towards the effective field,

or disable the energy precision control (discussed below).

The two controls **min_step_headroom** and **max_step_headroom** (default values 0.33 and 0.95, respectively) replace the single `step_headroom` control option in the `Oxs_EulerEvolve` class. The effective `step_headroom` is automatically adjusted by the evolver between the *min_headroom* and *max_headroom* limits to make the observed reject proportion approach the **reject_goal** (default value 0.05).

The **method** entry selects a particular Runge-Kutta implementation. It should be set to one of *rk2*, *rk4*, *rkf54*, *rkf54m*, or *rkf54s*; the default value is *rkf54*. The *rk2* and *rk4* methods implement canonical second and fourth global order Runge-Kutta methods[29], respectively. For *rk2*, stepsize control is managed by comparing $\dot{\mathbf{m}}$ at the middle and final points of the interval, similar to what is done for stepsize control for the `Oxs_EulerEvolve` class. One step of the *rk2* method involves 2 evaluations of $\dot{\mathbf{m}}$.

In the *rk4* method, two successive steps are taken at half the nominal step size, and the difference between that end point and that obtained with one full size step are compared. The error is estimated at 1/15th the maximum difference between these two states. One step of the *rk4* method involves 11 evaluations of $\dot{\mathbf{m}}$, but the end result is that of the 2 half-sized steps.

The remaining methods, *rkf54*, *rkf54m*, and *rkf54s*, are closely related Runge-Kutta-Fehlberg methods derived by Dormand and Prince[8, 9]. In the nomenclature of these papers, *rkf54* implements RK5(4)7FC, *rkf54m* implements RK5(4)7FM, and *rkf54s* implements RK5(4)7FS. All are 5th global order with an embedded 4th order method for stepsize control. Each step of these methods requires 6 evaluations of $\dot{\mathbf{m}}$ if the step is accepted, 7 if rejected. The difference between the methods involves tradeoffs between stability and error minimization. The RK5(4)7FS method has the best stability, RK5(4)7FM the smallest error, and RK5(4)7FC represents a compromise between the two. The default method used by `Oxs_RungeKuttaEvolve` is RK5(4)7FC.

The remaining undiscussed entry in the `Oxs_RungeKuttaEvolve` Specify block is **energy_precision**. This should be set to an estimate of the expected relative accuracy of the energy calculation. After accounting for any change in the total energy arising from time-varying applied fields, the energy remainder should decrease from one step of the LLG ODE to the next. `Oxs_RungeKuttaEvolve` will reject a step if the energy remainder is found to increase by more than that allowed by *eprecision*. The default value for *eprecision* is 10^{-10}. This control may be disabled by setting *eprecision* to -1.

The `Oxs_RungeKuttaEvolve` module provides the same scalar, scalar field, and vector field outputs as `Oxs_EulerEvolve`. It also provides the following state data accessible through the MIF `GetStateData` command:

```
Mx          My          Mz
dMx/dt      dMy/dt      dMz/dt
Total E     dE/dt       pE/pt
"Timestep lower bound"
```

The full name for each of these items is

`Oxs_RungeKuttaEvolve:<instance_name>:<item_name>`,

where `<instance_name>` is the instance name of the object (typically an empty string or something like "evolver"). For stage start states only the `Mx`, `My`, and `Mz` items are available. These terms, and the corresponding `dMx/dt`, `dMy/dt`, and `dMz/dt`, are component values averaged across the full simulation.

Examples: `sample.mif`, `acsample.mif`, `spinmag.mif`, `spinmag2.mif`, `varalpha.mif`, `yoyo.mif`.

Oxs_SpinXferEvolve: Time evolver that integrates an Landau-Lifshitz-Gilbert ODE augmented with a spin momentum term [35],

$$\frac{d\mathbf{m}}{dt} = -|\gamma|\,\mathbf{m}\times\mathbf{H}_{\text{eff}} + \alpha\left(\mathbf{m}\times\frac{d\mathbf{m}}{dt}\right) + |\gamma|\beta\epsilon\,(\mathbf{m}\times\mathbf{m}_p\times\mathbf{m}) - |\gamma|\beta\epsilon'\,\mathbf{m}\times\mathbf{m}_p \quad (7.4)$$

(compare to (7.3)), where

$$\begin{aligned}
\mathbf{m} &= \text{reduced magnetization, } \mathbf{M}/M_s\\
\gamma &= \text{Gilbert gyromagnetic ratio}\\
\beta &= \left|\frac{\hbar}{\mu_0 e}\right|\frac{J}{tM_s}\\
\mathbf{m}_p &= \text{(unit) electron polarization direction}\\
\epsilon &= \frac{P\Lambda^2}{(\Lambda^2+1)+(\Lambda^2-1)(\mathbf{m}\cdot\mathbf{m}_p)}\\
\epsilon' &= \text{secondary spin transfer term.}
\end{aligned}$$

In the definition of β, e is the electron charge in C, J is current density in A/m^2, t is the free layer thickness in meters, and M_s is the saturation magnetization in A/m.

The various parameters are defined in the Specify block, which is an extension of that for the `Oxs_RungeKuttaEvolve` class:

```
Specify Oxs_SpinXferEvolve:name {
    alpha α
    gamma_LL γ̄
    gamma_G γ
    do_precess precess
    allow_signed_gamma signed_gamma
    min_timestep minimum_stepsize
    max_timestep maximum_stepsize
    fixed_spins {
        atlas_spec
```

region1 region2 ...
}
`start_dm` $\Delta \boldsymbol{m}$
`stage_start` *scontinuity*
`error_rate` *rate*
`absolute_step_error` *abs_error*
`relative_step_error` *rel_error*
`energy_precision` *eprecision*
`min_step_headroom` *min_headroom*
`max_step_headroom` *max_headroom*
`reject_goal` *reject_proportion*
`method` *subtype*
`P` *polarization*
`P_fixed` *p_fixed_layer*
`P_free` *p_free_layer*
`Lambda` Λ
`Lambda_fixed` Λ*_fixed_layer*
`Lambda_free` Λ*_free_layer*
`eps_prime` *ep*
`J` *current_density*
`J_profile` *Jprofile_script*
`J_profile_args` *Jprofile_script_args*
`mp` *p_direction*
`energy_slack` *eslack*
}

The options duplicated in the `Oxs_RungeKuttaEvolve` class Specify block have the same meaning and default values here, with the exception of `error_rate`, which for `Oxs_SpinXferEvolve` has the default value of -1 (i.e., disabled).

The default values for **P** and **Lambda** are 0.4 and 2, respectively. If preferred, values for the fixed and free layers may be instead specified separately, through **P_fixed**, **P_free**, **Lambda_fixed**, and **Lambda_free**. Otherwise P_fixed = P_free = P and Lambda_fixed = Lambda_free = Lambda. Lambda must be larger than or equal to 1; set Lambda=1 to remove the dependence of ϵ on $\mathbf{m} \cdot \mathbf{m}_p$. If you want non-zero ϵ', it is set directly as **eps_prime**.

Current density **J** and unit polarization direction **mp** are required. The units on J are $\mathrm{A/m^2}$. Positive J produces torque that tends to align **m** towards $\mathbf{m}_p$.

Parameters J, mp, P, Lambda, and eps_prime may all be varied pointwise, but are fixed with respect to time. However, J can be multiplied by a time varying "profile," to model current rise times, pulses, etc. Use the **J_profile** and **J_profile_args** options to enable this feature. The *Jprofile_script* should be a Tcl script that returns a single scalar. *Jprofile_script_args* should be a subset of {`stage stage_time total_time`},

to specify arguments appended to *Jprofile_script* on each time step. Default is the entire set, in the order as listed.

The `Oxs_SpinXferEvolve` module provides the same five scalar outputs and three vector outputs as `Oxs_RungeKutta`, plus the scalar output "average J," and the vector field outputs "Spin torque" (which is $|\gamma|\beta\epsilon\,(\mathbf{m}\times\mathbf{m}_p\times\mathbf{m})$) and "J*mp." (Development note: In the case `propagate_mp` is enabled, `mp` is actually $\Delta_x\partial\mathbf{m}/\partial\mathbf{x}$, where $\mathbf{x}$ is the flow direction and Δ_x is the cell dimension in that direction.)

The `Oxs_SpinXferEvolve` class does **not** include any oersted field arising from the current. Of course, arbitrary fields simulating the oersted field may be added separately as Zeeman energy terms. An example of this is contained in the `spinxfer.mif` sample file.

There are no temperature effects in this evolver, i.e., it is a T = 0 K code.

Note also that $\mathbf{m}_p$ is fixed.

For basic usage, the Specify block can be as simple as

```
Specify Oxs_SpinXferEvolve:evolve {
  alpha 0.014
  J 7.5e12
  mp {1 0 0}
  P 0.4
  Lambda 2
}
```

This class is still in early development; at this time the example files are located in `oommf/app/oxs/local` instead of `oommf/app/oxs/examples`.

Examples: `spinxfer.mif`, `spinxfer-miltat.mif`, `spinxfer-onespin.mif`.

Oxs_CGEvolve: The minimization evolver is `Oxs_CGEvolve`, which is an in-development conjugate gradient minimizer with no preconditioning. The Specify block has the form

```
Specify Oxs_CGEvolve:name {
    gradient_reset_angle reset_angle
    gradient_reset_count count
    minimum_bracket_step minbrack
    maximum_bracket_step maxbrack
    line_minimum_angle_precision min_prec_angle
    line_minimum_relwidth relwidth
    energy_precision eprecision
    method cgmethod
    fixed_spins {
        atlas_spec
        region1 region2 ...
```

```
        }
    }
```

All entries have default values.

The evolution to an energy minimum precedes by a sequence of line minimizations. Each line represents a one dimensional affine subspace in the $3N$ dimensional space of possible magnetization configurations, where N is the number of spins in the simulation. Once a minimum has been found along a line, a new direction is chosen that is ideally orthogonal to all preceding directions, but related to the gradient of the energy taken with respect to the magnetization. In practice the line direction sequence cannot be extended indefinitely; the parameters **gradient_reset_angle** and **gradient_reset_count** control the gradient resetting process. The first checks the angle between the new direction and the gradient. If that angle is larger than *reset_angle* (expressed in degrees), then the selected direction is thrown away, and the conjugate-gradient process is re-initialized with the gradient direction as the new first direction. In a similar vein, *count* specifies the maximum number of line directions selected before resetting the process. Because the first line in the sequence is selected along the gradient direction, setting *count* to 1 effectively turns the algorithm into a steepest descent minimization method. The default values for *reset_angle* and *count* are 80 degrees and 50, respectively.

Once a minimization direction has been selected, the first stage of the line minimization is to bracket the minimum energy on that line, i.e., given a start point on the line—the location of the minimum from the previous line minimization—find another point on the line such that the energy minimum lies between those two points. As one moves along the line, the spins in the simulation rotate, with one spin rotating faster than (or at least as fast as) all the others. If the start point was not the result of a successful line minimization from the previous stage, then the first bracket attempt step is sized so that the fastest moving spin rotates through the angle specified by **minimum_bracket_step**. In the more usual case that the start point is a minimum from the previous line minimization stage, the initial bracket attempt step size is set to the distance between the current start point and the start point of the previous line minimization stage.

The energy and gradient of the energy are examined at the candidate bracket point to test if an energy minimum lies in the interval. If not, the interval is extended, based on the size of the first bracket attempt interval and the derivatives of the energy at the interval endpoints. This process is continued until either a minimum is bracketed or the fastest moving spin rotates through the angle specified by **maximum_bracket_step**.

If the bracketing process is successful, then a one dimensional minimization is carried out in the interval, using both energy and energy derivative information. Each step in this process reduces the width of the bracketing interval. This process is continued until the angle between the line direction and the computed energy gradient is within

line_minimum_angle_precision degrees of orthogonal, and the width of the interval relative to the distance of the interval from the start point (i.e., the stop point from the previous line minimization process) is less than **line_minimum_relwidth**. The stop point, i.e., the effective minimum, is taken to be the endpoint of the final interval having smaller energy. The default value for *min_prec_angle* is 1 degree, and the default value for *relwidth* is 1. This latter setting effectively disables the `line_minimum_relwidth` control, which should generally be used only as a secondary control.

If the bracketing process is unsuccessful, i.e., the check for bracketed energy minimum failed at the maximum bracket interval size allowed by `maximum_bracket_step`, then the maximum bracket endpoint is accepted as the next point in the minimization iteration.

Once the line minimum stop point has been selected, the next iteration begins with selection of a new line direction, as described above, except in the case where the stop point was not obtained as an actual minimum, but rather by virtue of satisfying the `maximum_bracket_step` constraint. In that case the orthogonal line sequence is reset, in the same manner as when the `gradient_reset_angle` or `gradient_reset_count` controls are triggered, and the next line direction is taken directly from the energy gradient.

There are several factors to bear in mind when selecting values for the parameters `minimum_bracket_step`, `maximum_bracket_step`, and `line_minimum_relwidth`. If `minimum_bracket_step` is too small, then it may take a great many steps to obtain an interval large enough to bracket the minimum. If `minimum_bracket_step` is too large, then the bracket interval will be unnecessarily generous, and many steps may be required to locate the minimum inside the bracketing interval. However, this value only comes into play when resetting the line minimization direction sequence, so the setting is seldom critical. It is specified in degrees, with default value 0.05.

If `maximum_bracket_step` is too small, then the minima will be mostly not bracketed, and the minimization will degenerate into a type of steepest descent method. On the other hand, if `maximum_bracket_step` is too large, then the line minimizations may draw the magnetization far away from a local energy minimum (i.e., one on the full $3N$ dimensional magnetization space), eventually ending up in a different, more distant minimum. The value for `maximum_bracket_step` is specified in degrees, with default value 10.

The `line_minimum_angle_precision` and `line_minimum_relwidth` values determine the precision of the individual line minimizations, not the total minimization procedure, which is governed by the stopping criteria specified in the driver's Specify block. However, these values are important because the precision of the line minimizations affects the line direction sequence orthogonality. If both are too coarse, then the selected line directions will quickly drift away from mutual orthogonality. Conversely,

setting either too fine will produce additional line minimization steps that do nothing to improve convergence towards the energy minimum in the full $3N$ dimensional magnetization space.

The **energy_precision** parameter estimates the relative precision of the energy computations. This is used to introduce a slack factor into the energy comparisons during the bracketing and line minimization stages, that is, if the computed energy values at two points have relative error difference smaller than *eprecision*, they are treated as having the same energy. The default value for *eprecision* is 1e-10. The true precision will depend primarily on the number of spins in the simulation. It may be necessary for very large simulations to increase the *eprecision* value.

The **method** parameter can be set to either `Fletcher-Reeves` or `Polak-Ribiere` to specify the conjugate gradient direction selection algorithm. The default is Fletcher-Reeves, which has somewhat smaller memory requirements.

The final parameter, **fixed_spins**, performs the same function as it does in the `Oxs_EulerEvolve` class.

The `Oxs_CGEvolve` module provides nine scalar, one scalar field, and two vector field outputs. The scalar outputs are

- **Max mxHxm:** maximum $|\mathbf{m} \times \mathbf{H} \times \mathbf{m}|$, in A/m; $\mathbf{m}$ is the unit magnetization direction.
- **Total energy:** in joules.
- **Delta E:** change in energy between last step and current step, in joules.
- **Energy calc count:** number of times total energy has been calculated.
- **Bracket count:** total number of attempts required to bracket energy minimum during first phase of line minimization procedures.
- **Line min count:** total number of minimization steps during second phase of line minimization procedures (i.e., steps after minimum has been bracketed).
- **Cycle count:** number of line direction selections.
- **Cycle sub count:** number of line direction selections since the last gradient direction reset.
- **Conjugate cycle count:** number of times the conjugate gradient process has been reset to the gradient direction.

The scalar field output is

- **Total energy density:** cellwise total energy density, in $\mathrm{J/m^3}$.

The vector field outputs are

- **H:** total effective field in A/m.

- **mxHxm:** in A/m; **m** is the unit magnetization direction.

Examples: `cgtest.mif`, `stdprob3.mif`, `yoyo.mif`.

7.3.5 Drivers

While evolvers are responsible for moving the simulation forward in individual steps, *drivers* coordinate the action of the evolver on the simulation as a whole, by grouping steps into tasks, stages and runs.

Tasks are small groups of steps that can be completed without adversely affecting user interface responsiveness. Stages are larger units specified by the MIF problem description; in particular, problem parameters are not expected to change in a discontinuous manner inside a stage. The run is the complete sequence of stages, from problem start to finish. The driver detects when stages and runs are finished, using criteria specified in the MIF problem description, and can enforce constraints, such as making sure stage boundaries respect time stopping criteria.

There are two drivers in Oxs, `Oxs_TimeDriver` for controlling LLG time evolvers such as `Oxs_RungeKuttaEvolve`, and `Oxs_MinDriver` for controlling energy minimization evolvers like `Oxs_CGEvolve`.

Oxs_TimeDriver: The Oxs time driver is **Oxs_TimeDriver**. The specify block has the form

```
Specify Oxs_TimeDriver:name {
    evolver evolver_spec
    mesh mesh_spec
    Ms scalar_field_spec
    m0 vector_field_spec
    stopping_dm_dt torque_criteria
    stopping_time time_criteria
    stage_iteration_limit stage_iteration_count
    total_iteration_limit total_iteration_count
    stage_count number_of_stages
    stage_count_check test
    checkpoint_file restart_file_name
    checkpoint_interval checkpoint_minutes
    checkpoint_disposal cleanup_behavior
    start_iteration iteration
    start_stage stage
    start_stage_iteration stage_iteration
    start_stage_start_time stage_time
    start_stage_elapsed_time stage_elapsed_time
    start_last_timestep timestep
```

```
    normalize_aveM_output aveMflag
    report_max_spin_angle report_angle
    report_wall_time report_time
}
```

The first four parameters, **evolver**, **mesh**, **Ms** and **m0** provide references to a time evolver, a mesh, a scalar field and a vector field, respectively. Here `Ms` is the pointwise saturation magnetization in A/m, and `m0` is the initial configuration for the magnetization unit spins, i.e., $|\mathbf{m}| = 1$ at each point. These four parameters are required.

The next group of three parameters control stage stopping criteria. Measured in degrees per nanosecond, the **stopping_dm_dt** value specifies that a stage should be considered complete when the maximum $|d\mathbf{m}/dt|$ across all spins drops below this value. Similarly, the **stopping_time** value specifies the maximum "Simulation time," i.e., the Landau-Lifshitz-Gilbert ODE (7.2), (7.3) time, allowed per stage. For example, if *time_criteria* is 10^{-9}, then no stage will evolve for more than 1 ns. If there were a total of 5 stages in the simulation, then the total simulation time would be not more than 5 ns. The third way to terminate a stage is with a **stage_iteration_limit**. This is a limit on the number of successful evolver steps allowed per stage. A stage is considered complete when any one of these three criteria are met. Each of the criteria may be either a single value, which is applied to every stage, or else a *grouped list* (Sec. 17.3.3) of values. If the simulation has more stages than a criteria list has entries, then the last criteria value is applied to all additional stages. These stopping criteria all provide a default value of 0, meaning no constraint, but usually at least one is specified since otherwise there is no automatic stage termination control. For quasi-static simulations, a `stopping_dm_dt` value in the range of 1.0 to 0.01 is reasonable; the numerical precision of the energy calculations usually makes it not possible to obtain $|d\mathbf{m}/dt|$ much below 0.001 degree per nanosecond.

The **total_iteration_limit**, **stage_count** and **stage_count_check** parameters involve simulation run completion conditions. The default value for the first is 0, interpreted as no limit, but one may limit the total number of steps performed in a simulation by specifying a positive integer value here. The more usual run completion condition is based on the stage count. If a positive integer value is specified for `stage_count`, then the run will be considered complete when the stage count reaches that value. If `stage_count` is not specified, or is given the value 0, then the effective *number_of_stages* value is computed by examining the length of the stopping criteria lists, and also any other `Oxs_Ext` object that has stage length expectations, such as `Oxs_UZeeman`. The longest of these is taken to be the stage limit value. Typically these lengths, along with `stage_count` if specified, will all be the same, and any differences indicate an error in the MIF file. Oxs will automatically test this condition, provided `stage_count_check` is set to 1, which is the default value. Stage length requests of 0 or 1 are ignored in this test, since those lengths are commonly used to represent sequences of arbitrary length. At times a short sequence is intentionally specified that

is meant to be implicitly extended to match the full simulation stage length. In this case, the stage count check can be disabled by setting *test* to 0.

The checkpoint options are used to control the periodic saving of solver state to disk; these saves are used by the **Oxsii** and **Boxsi** restart feature. The default filename is

`basename-CRC_<mifcrc>-Params_<paramcrc>.restart`

where `basename` is the MIF `basename` option active when the driver `Specify` block is processed, *<mifcrc>* is the CRC of the MIF file, and *<paramcrc>* is the CRC of the command line `-parameter` option. The CRCs help protect against simultaneous Oxs runs from overwriting each other's checkpoint files. The default checkpoint filename can be changed with the **checkpoint_file** option; both are overridden by the **Oxsii**/**Boxsi** command line `-restartfilename` option. Standard practice is to give the checkpoint file the `.restart` extension.

A copy of the active MIF file is also made at the time of the first checkpoint save. The MIF backup has the same name as the checkpoint file but with a `-mif` suffix.

The checkpoint directory default is the MIF file directory, but this can changed by setting a `OOMMF_RESTARTFILEDIR` environment variable. Both yield to the **Oxsii**/**Boxsi** `-restartfiledir` command line option, or you can specify the restart file via an absolute path. If `OOMMF_RESTARTFILEDIR` or `-restartfiledir` are given as a relative path, they are interpreted relative to the MIF file directory.

Cleanup of the checkpoint files is determined by the setting of **checkpoint_disposal**, which should be one of *standard* (the default), *done_only*, or *never*. Under the standard setting, the checkpoint files are automatically deleted upon normal program termination, either because the solver reached the end of the problem, or because the user interactively terminated the problem prematurely. If *cleanup_behavior* is set to `done_only`, then the checkpoint files are only deleted if the problem endpoint is reached. If *cleanup_behavior* is `never`, then OOMMF does not delete checkpoint files; the user is responsible for deleting these files as she desires.

The **checkpoint_interval** value is the time in minutes between overwrites of the checkpoint file. No checkpoint file is written until *checkpoint_minutes* have elapsed. Checkpoint writes occur between solver iterations, so the actual interval time may be somewhat longer than the specified time. If *checkpoint_minutes* is 0, then each step is saved. Setting *checkpoint_minutes* to -1 disables checkpointing. The default checkpoint interval is 15 minutes.

The six **start_*** options control the problem run start point. These are intended primarily for automatic use by the restart feature. The default value for each is 0.

The **normalize_aveM_output** option is used to control the scaling and units on the average magnetization components M_x, M_y and M_z sent as DataTable output (this includes output sent to **mmDataTable**, **mmGraph**, and **mmArchive**). If *aveMflag*

is true (1), then the output values are scaled to lie in the range $[-1, 1]$, where the extreme values are obtained only at saturation (i.e., all the spins are aligned). If *aveMflag* is false (0), then the output is in A/m. The default setting is 1.

In the older MIF 2.1 format, the driver Specify block supports three additional values: **basename**, **scalar_output_format**, and **vector_field_output_format**. In the MIF 2.2 format these output controls have been moved into the `SetOptions` block. See the `SetOptions` documentation for details.

`Oxs_TimeDriver` provides 12 scalar outputs and 2 vector field outputs. The scalar outputs are

- **Stage:** current stage number, counting from 0.
- **Stage iteration:** number of successful evolver steps in the current stage.
- **Iteration:** number of successful evolver steps in the current simulation.
- **Simulation time:** Landau-Lifshitz-Gilbert evolution time, in seconds.
- **Last time step:** The size of the preceding time step, in seconds.
- **Mx/mx:** magnetization component in the x direction, averaged across the entire simulation, in A/m (Mx) or normalized units (mx), depending on the setting of the `normalize_aveM_output` option.
- **My/my:** magnetization component in the y direction, averaged across the entire simulation, in A/m (My) or normalized units (my), depending on the setting of the `normalize_aveM_output` option.
- **Mz/mz:** magnetization component in the z direction, averaged across the entire simulation, in A/m (Mz) or normalized units (mz), depending on the setting of the `normalize_aveM_output` option.
- **Max Spin Ang:** maximum angle between neighboring spins having non-zero magnetization M_s, measured in degrees. The definition of "neighbor" depends on the mesh, but for `Oxs_RectangularMesh` the neighborhood of a point consists of 6 points, those nearest forward and backward along each of the 3 coordinate axis directions.
- **Stage Max Spin Ang:** the largest value of "Max Spin Ang" obtained across the current stage, in degrees.
- **Run Max Spin Ang:** the largest value of "Max Spin Ang" obtained across the current run, in degrees.
- **Wall time:** Wall clock time, in seconds.

The three "Max Spin Ang" outputs are disabled by default. In general one should refer instead to the neighboring spin angle outputs provided by the exchange energies. However, for backward compatibility, or for simulations without any exchange energy terms,

the driver spin angle outputs can be enabled by setting the **report_max_spin_angle** option to 1.

The "Wall time" output is also disabled by default. It can be enabled by setting the **report_wall_time** option to 1. It reports the wall clock time, in seconds, since a system-dependent zero-time. This output may be useful for performance comparisions and debugging. (Note: The timestamp for a magnetization state is recorded when output is first requested for that state; the timestamp is not directly tied to the processing of the state.)

The vector field outputs are

- **Magnetization:** magnetization vector **M**, in A/m.
- **Spin:** unit magnetization **m**. This output ignores the `vector_field_output_format` *precision* setting, instead always exporting at full precision.

Examples: `sample.mif`, `pulse.mif`.

Oxs_MinDriver: The Oxs driver for controlling minimization evolvers is **Oxs_MinDriver**. The specify block has the form

```
Specify Oxs_MinDriver:name {
    evolver evolver_spec
    mesh mesh_spec
    Ms scalar_field_spec
    m0 vector_field_spec
    stopping_mxHxm torque_criteria
    stage_iteration_limit stage_iteration_count
    total_iteration_limit total_iteration_count
    stage_count number_of_stages
    stage_count_check test
    checkpoint_file restart_file_name
    checkpoint_interval checkpoint_minutes
    checkpoint_disposal cleanup_behavior
    start_iteration iteration
    start_stage stage
    start_stage_iteration stage_iteration
    start_stage_start_time stage_time
    start_stage_elapsed_time stage_elapsed_time
    start_last_timestep timestep
    normalize_aveM_output aveMflag
    report_max_spin_angle report_angle
    report_wall_time report_time
}
```

These parameters are the same as those described for the `Oxs_TimeDriver` class (page 91), except that **stopping_mxHxm** replaces `stopping_dm_dt`, and there is no analogue to `stopping_time`. The value for `stopping_mxHxm` is in A/m, and may be a *grouped list* (Sec. 17.3.3). Choice depends on the particulars of the simulation, but typical values are in the range 10 to 0.1. Limits in the numerical precision of the energy calculations usually makes it not possible to obtain $|\mathbf{m} \times \mathbf{H} \times \mathbf{m}|$ below about 0.01 A/m. This control can be disabled by setting it to 0.0.

As with `Oxs_TimeDriver`, in the older MIF 2.1 format this Specify block supports three additional values: **basename** to control output filenames, and output format controls **scalar_output_format** and **vector_field_output_format**. In the MIF 2.2 format these output controls have been moved into the `SetOptions` block. See the `SetOptions` documentation for details.

`Oxs_MinDriver` provides 10 scalar outputs and 2 vector field outputs. The scalar outputs are

- **Stage:** current stage number, counting from 0.
- **Stage iteration:** number of successful evolver steps in the current stage.
- **Iteration:** number of successful evolver steps in the current simulation.
- **Mx/mx:** magnetization component in the x direction, averaged across the entire simulation, in A/m (Mx) or normalized units (mx), depending on the setting of the `normalize_aveM_output` option.
- **My/my:** magnetization component in the y direction, averaged across the entire simulation, in A/m (My) or normalized units (my), depending on the setting of the `normalize_aveM_output` option.
- **Mz/mz:** magnetization component in the z direction, averaged across the entire simulation, in A/m (Mz) or normalized units (mz), depending on the setting of the `normalize_aveM_output` option.
- **Max Spin Ang:** maximum angle between neighboring spins having non-zero magnetization M_s, measured in degrees. The definition of "neighbor" depends on the mesh, but for `Oxs_RectangularMesh` the neighborhood of a point consists of 6 points, those nearest forward and backward along each of the 3 coordinate axis directions.
- **Stage Max Spin Ang:** the largest value of "Max Spin Ang" obtained across the current stage, in degrees.
- **Run Max Spin Ang:** the largest value of "Max Spin Ang" obtained across the current run, in degrees.
- **Wall time:** Wall clock time, in seconds.

As is the case for the `Oxs_TimeDriver`, the three "Max Spin Ang" outputs and "Wall time" are disabled by default. They angle outputs are enabled by setting the **report_max_spin_angle** option to 1, and the wall time output is enabled by setting the **report_wall_time** option to 1.

The vector field outputs are

- **Magnetization:** magnetization vector **M**, in A/m.
- **Spin:** unit magnetization **m**. This output ignores the `vector_field_output_format` *precision* setting, instead always exporting at full precision.

Examples: `cgtest.mif`, `stdprob3.mif`.

7.3.6 Field Objects

Field objects return values (either scalar or vector) as a function of position. These are frequently used as embedded objects inside Specify blocks of other `Oxs_Ext` objects to initialize spatially varying quantities, such as material parameters or initial magnetization spin configurations. Units on the returned values will be dependent upon the context in which they are used.

Scalar field objects are documented first. Vector field objects are considered farther below.

Oxs_UniformScalarField: Returns the same constant value regardless of the import position. The Specify block takes one parameter, **value**, which is the returned constant value. This class is frequently embedded inline to specify homogeneous material parameters. For example, inside a driver Specify block we may have

```
Specify Oxs_TimeDriver {
    ...
    Ms { Oxs_UniformScalarField {
       value 8e5
    }}
    ...
}
```

As discussed in the MIF 2 documentation (Sec. 17.3.3, page 242), when embedding `Oxs_UniformScalarField` or `Oxs_UniformVectorField` objects, a notational shorthand is allowed that lists only the value. The previous example is exactly equivalent to

```
Specify Oxs_TimeDriver {
    ...
    Ms 8e5
    ...
}
```

where an implicit `Oxs_UniformScalarField` object is created with `value` set to `8e5`.

Examples: `sample.mif`, `cgtest.mif`.

Oxs_AtlasScalarField: Declares values that are defined across individual regions of an `Oxs_Atlas`. The Specify block looks like

```
Specify Oxs_AtlasScalarField:value {
    atlas atlas_spec
    multiplier mult
    default_value scalar_field_spec
    values {
        region1_label scalar_field_spec1
        region2_label scalar_field_spec2
        ...
    }
}
```

The specified **atlas** is used to map cell locations to regions; the value at the cell location of the scalar field from the corresponding **values** sub-block is assigned to that cell. The **default_value** entry is optional; if specified, and if a cell's region is not included in the `values` sub-block, then the `default_value` scalar field is used. If `default_value` is not specified, then missing regions will raise an error.

The scalar field entries may specify any of the scalar field types described in this (Field Objects) section. As usual, one may provide a single numeric value in any of the `scalar_field_spec` positions; this will be interpreted as requesting a uniform (spatially homogeneous) field with the indicated value.

If the optional **multiplier** value is provided, then each field value is scaled (multiplied) by the value *mult*.

The vector field analogue to this class is `Oxs_AtlasVectorField`, which is described below in the vector fields portion of this section.

Examples: `diskarray.mif`, `ellipsoid.mif`, `grill.mif`, `spinvalve.mif`, `tclshapes.mif`.

Oxs_LinearScalarField: Returns a value that varies linearly with position. The Specify block has the form:

```
Specify Oxs_LinearScalarField:name {
    norm value
    vector { v_x v_y v_z }
    offset off
}
```

If optional value **norm** is specified, then the given **vector** is first scaled to the requested size. The **offset** entry is optional, with default value 0. For any given point (x, y, z), the scalar value returned by this object will be $xv_x + yv_y + zv_z + off$.

Example: `spinvalve-af.mif`.

Oxs_RandomScalarField: Defines a scalar field that varies spatially in a random fashion. The Specify block has the form:

```
Specify Oxs_RandomScalarField:name {
    range_min minvalue
    range_max maxvalue
    cache_grid mesh_spec
}
```

The value at each position is drawn uniformly from the range declared by the two required parameters, **range_min** and **range_max**. There is also an optional parameter, **cache_grid**, that takes a mesh specification describing the grid used for spatial discretization. If **cache_grid** is not specified, then each call to `Oxs_RandomScalarField` generates a different field. If you want to use the same random scalar field in two places (as a base for setting, say anisotropy coefficients and saturation magnetization), then specify **cache_grid** with the appropriate (usually the base problem) mesh.

Examples: `randomshape.mif`, `stdprob1.mif`.

Oxs_ScriptScalarField: Analogous to the parallel `Oxs_ScriptVectorField` class, this class produces a scalar field dependent on a Tcl script and optionally other scalar and vector fields. The Specify block has the form

```
Specify Oxs_ScriptScalarField:name {
    script Tcl_script
    script_args { args_request }
    scalar_fields { scalar_field_spec ... }
    vector_fields { vector_field_spec ... }
    atlas atlas_spec
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
}
```

For each point of interest, the specified **script** is called with the arguments requested by **script_args** appended to the command, as explained in the User Defined Support Procedures of the MIF 2 file format documentation. The value for `script_args` should be a subset of {`rawpt relpt minpt maxpt span scalars vectors`}.

If `rawpt` is requested, then when the Tcl proc is called, at the corresponding spot in the argument list the `x`, `y`, `z` values of point will be placed, in problem coordinates (in meters). The points so passed will usually be node points in the simulation discretization (the mesh), but this does not have to be the case in general. The `relpt`, `minpt`, `maxpt`, and `span` rely on a definition of a *bounding box*, which is an axes parallel parallelepiped. The bounding box must be specified by either referencing an **atlas**, or by explicitly stating the range via the three entries **xrange**, **yrange**, **zrange** (in meters). The `minpt` and `maxpt` arguments list the minimum and maximum values of the bounding box (coordinate by coordinate), while `span` provides the 3-vector resulting from (`maxpt` − `minpt`). The `relpt` selection provides `x_rel`, `y_rel`, `z_rel`, where each element lies in the range $[0, 1]$, indicating a relative position between `minpt` and `maxpt`, coordinate-wise.

Each of the `script_args` discussed so far places exactly 3 arguments onto the Tcl proc argument list. The last two, `scalars` and `vectors`, place arguments depending on the size of the **scalar_fields** and **vector_fields** lists. The `scalar_fields` value is a list of other scalar field objects. Each scalar field is evaluated at the point in question, and the resulting scalar value is placed on the Tcl proc argument list, in order. The `vector_fields` option works similarly, except each vector field generates three points for the Tcl proc argument list, since the output from vector field objects is a three vector. Although the use of these entries appears complicated, this is a quite powerful facility that allows nearly unlimited control for the modification and combination of other field objects. Both `scalar_fields` and `vector_fields` entries are optional.

If `script_args` is not specified, the default value `relpt` is used.

Note that if `script_args` includes `relpt`, `minpt`, `maxpt`, or `span`, then a bounding box must be specified, as discussed above. The following example uses the explicit range method. See the `Oxs_ScriptVectorField` documentation (page 108) for an example using an atlas specification.

```
proc Ellipsoid { xrel yrel zrel } {
    set xrad [expr {$xrel - 0.5}]
    set yrad [expr {$yrel - 0.5}]
    set zrad [expr {$zrel - 0.5}]
    set test [expr {$xrad*$xrad+$yrad*$yrad+$zrad*$zrad}]
    if {$test>0.25} {return 0}
    return 8.6e5
}

Specify Oxs_ScriptScalarField {
    script Ellipsoid
    xrange { 0   1e-6 }
    yrange { 0 250e-9 }
    zrange { 0  50e-9 }
```

```
}
```

This `Oxs_ScriptScalarField` object returns 8.6×10^5 if the import (x,y,z) lies within the ellipsoid inscribed inside the axes parallel parallelepiped defined by (xmin=0, ymin=0, zmin=0) and (xmax=1e-6, ymax=250e-9, zmax=50e-9), and 0 otherwise. See also the discussion of the `ReadFile` MIF extension command in Sec. 17.3.2 for an example using an imported image file for similar purposes.

Below is one more example, illustrating the use of the `vector_fields` option.

```
proc DotProduct { x1 y1 z1 x2 y2 z2 } {
    return [expr {$x1*$x2+$y1*$y2+$z1*$z2}]
}

Specify Oxs_FileVectorField:file1 {
    atlas :atlas
    file  file1.omf
}

Specify Oxs_UniformVectorField:dir111 {
    norm 1
    vector {1 1 1}
}

Specify Oxs_ScriptScalarField:project {
    script DotProduct
    script_args vectors
    vector_fields {:file1 :dir111}
}
```

The scalar field `:project` yields at each point in space the projection of the vector field `:file1` onto the [1,1,1] direction.

Examples: `antidots-filled.mif`, `ellipsoid-fieldproc.mif`, `manyregions-scriptfields.mif`, `manyspheres.mif`, `varalpha.mif`.

Oxs_VecMagScalarField: The `Oxs_VecMagScalarField` class produces a scalar field from a vector field by taking the norm of the vector field on a point-by-point basis, i.e.,

$$\|\boldsymbol{v}\| = \sqrt{v_x^2 + v_y^2 + v_z^2}.$$

The Specify block has the form:

```
Specify Oxs_VecMagScalarField:name {
    field vector_field_spec
    multiplier mult
```

```
    offset off
}
```

The **multiplier** and **offset** entries are applied after the vector norm, i.e., the resulting scalar field is `mult` $* \|\boldsymbol{v}\| +$ `off`. The default values for `mult` and `off` are 1 and 0, respectively.

The functionality of the `Oxs_VecMagScalarField` class may be achieved with the `Oxs_ScriptScalarField` class by using the `vector_fields` option and a Tcl script to compute the vector norm. However, this particular functionality is needed frequently enough that a specialized class is useful. For example, this class can be used in conjunction with a vector field object to set both the saturation magnetization distribution (M_s) and the initial magnetization:

```
Specify Oxs_FileVectorField:file1 {
    atlas :atlas
    file  file1.omf
}

Specify Oxs_TimeDriver {
    basename test
    evolver :evolve
    stopping_dm_dt 0.01
    mesh :mesh
    m0 :file1
    Ms { Oxs_VecMagScalarField {
       field :file1
    }}
}
```

Example: `sample-vecrotate.mif`.

Oxs_ScriptOrientScalarField: Scalar fields provide scalar values as a function of position across three-space. The `Oxs_ScriptOrientScalarField` class is used to compose a transformation on the input position before evaluation by a scalar field. The Specify block has the form:

```
Specify Oxs_ScriptOrientScalarField:name {
    field scalar_field_spec
    script Tcl_script
    script_args { args_request }
    atlas atlas_spec
    xrange { xmin xmax }
    yrange { ymin ymax }
```

zrange { *zmin zmax* }
}

The **field** argument should refer to a scalar field object. The **script** is a Tcl script that should return a position vector that will be sent on the `field` object to ultimately produce a scalar value. The arguments to the `Tcl_script` are determined by **script_args**, which should be a subset of {`relpt rawpt minpt maxpt span`}. If any arguments other than `rawpt` are requested, then the bounding box must be specified by either the **atlas** option, or else through the three **xrange**, **yrange**, **zrange** entries. The default value for `script_args` is `relpt`.

The `Oxs_ScriptOrientScalarField` class can be used to change the "orientation" of a scalar field, as in the following simple example, which reflects the `:file1mag` scalar field across the yz-plane:

```
Specify Oxs_FileVectorField:file1 {
    atlas :atlas
    file  file1.omf
}

Specify Oxs_VecMagScalarField:file1mag {
   field :file1
}

proc Reflect { x y z xmin ymin zmin xmax ymax zmax} {
   return [list [expr {($xmax+$xmin-$x)}] $y $z]
}

Specify Oxs_ScriptOrientScalarField:reflect {
   field :file1mag
   script Reflect
   script_args {rawpt minpt maxpt}
   atlas :atlas
}
```

See also the `Oxs_ScriptOrientVectorField` class (page 111) for analogous operations on vector fields.

Example: `sample-reflect.mif`.

Oxs_AffineOrientScalarField: The `Oxs_AffineOrientScalarField` class is similar to the `Oxs_ScriptOrientScalarField` class, except that the transformation on the import position is by an affine transformation defined in terms of a 3×3 matrix and an offset instead of a Tcl script. Although this functionality can be obtained by an appropriate Tcl script, the `Oxs_AffineOrientScalarField` is easier to use and will

run faster, as the underlying transformation is performed by compiled C++ instead of Tcl script.

The Specify block has the form:

```
Specify Oxs_AffineOrientScalarField:name {
    field scalar_field_spec
    M { matrix_entries ... }
    offset { off_x off_y off_z }
    inverse invert_flag
    inverse_slack slack
}
```

If $F(\boldsymbol{x})$ represents the scalar field specified by the **field** value, then the resulting transformed scalar field is $F(M\boldsymbol{x} + \mathbf{off})$. Here $\mathbf{M}$ is a 3×3 matrix, which may be specified by a list of 1, 3, 6 or 9 entries. If the `matrix_entries` list consists of a single value, then M is taken to be that value times the identity matrix, i.e., M is a homogeneous scaling transformation. If `matrix_entries` consists of 3 values, then M is taken to be the diagonal matrix with those three values along the diagonal. If `matrix_entries` is 6 elements long, then M is assumed to be a symmetric matrix, where the 6 elements specified correspond to M_{11}, M_{12}, M_{13}, M_{22}, M_{23}, and M_{33}. Finally, if `matrix_entries` is 9 elements long, then the elements specify the entire matrix, in the order M_{11}, M_{12}, M_{13}, M_{21}, ..., M_{33}. If M is not specified, then it is taken to be the identity matrix.

The **offset** entry is simply a 3-vector that is added to $M\boldsymbol{x}$. If `offset` is not specified, then it is set to the zero vector.

It is frequently the case that the transformation that one wants to apply is not $M\boldsymbol{x}+\mathbf{off}$, but rather the inverse, i.e., $M^{-1}(\boldsymbol{x} - \mathbf{off})$. Provided M is nonsingular, this can be accomplished by setting the **inverse** option to 1. In this case the matrix $M.M^{-1}$ is compared to the identity matrix, to check the accuracy of the matrix inversion. If any entry in $M.M^{-1}$ differs from I by more than the 8-byte float machine precision (typically 2×10^{-16}) times the value of **inverse_slack**, then an error is raised. The default setting for `invert_flag` is 0, meaning don't invert, and the default setting for `slack` is 128.

Here is an example using `Oxs_AffineOrientScalarField` to rotate a field by 90° counterclockwise about the z-axis. Note that the specified atlas is square in x and y, with the origin of the atlas coordinates in the center of the atlas volume.

```
Specify Oxs_BoxAtlas:atlas {
  xrange {-250e-9 250e-9}
  yrange {-250e-9 250e-9}
  zrange { -15e-9  15e-9}
}
```

```
Specify Oxs_FileVectorField:file1 {
    atlas :atlas
    file  file1.omf
}

Specify Oxs_VecMagScalarField:file1mag {
   field :file1
}

Specify Oxs_AffineOrientScalarField:reflect {
   field :file1mag
   M { 0 1 0
      -1 0 0
       0 0 1 }
}
```

See also the `Oxs_AffineOrientVectorField` class (page 112) for analogous operations on vector fields.

Example: `sample-rotate.mif`.

Oxs_AffineTransformScalarField: Like the `Oxs_AffineOrientScalarField` class, this class composes an affine transform with a separate scalar field, but in this case the affine transform is applied *after* the field evaluation. The Specify block has the form:

```
Specify Oxs_AffineTransformScalarField:name {
    field scalar_field_spec
    multiplier mult
    offset off
    inverse invert_flag
}
```

If $F(\boldsymbol{x})$ represents the scalar field specified by the **field** value, then the resulting scalar field is mult $*$ $F(\boldsymbol{x})$ + off. Since the output from F is a scalar, both **multiplier** and **offset** are scalars. If **inverse** is 1, then the transform is changed to ($F(\boldsymbol{x})$ − off) /mult, provided `mult` is non-zero.

The default values for *mult*, *off*, and *invert_flag* are 1, 0, and 0, respectively. The `field` value is the only required entry.

The functionality provided by `Oxs_AffineTransformScalarField` can also be produced by the `Oxs_ScriptScalarField` class (page 99) with the `scalar_fields` entry, but the `Oxs_AffineTransformScalarField` class is faster and has a simpler interface. See also the `Oxs_AffineTransformVectorField` class (page 112) for analogous operations on vector fields.

Example: `sample-rotate.mif`.

Oxs_ImageScalarField: This class creates a scalar field using an image. The Specify block has the form

```
Specify Oxs_ImageScalarField:name {
    image pic
    invert invert_flag
    multiplier mult
    offset off
    viewplane view
    atlas atlas_spec
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    exterior ext_flag
}
```

The **image** is interpreted as a monochromatic map, yielding a scalar field with black corresponding to zero and white to one if **invert** is 0 (the default), or with black corresponding to 1 and white to 0 if `invert` is 1. Color images are converted to grayscale by simply summing the red, green, and blue components. A **multiplier** option is available to change the range of values from $[0, 1]$ to $[0, \texttt{mult}]$, after which the **offset** value, if any, is added.

The **viewplane** is treated in the same manner as the `Oxs_ImageAtlas` viewplane option, and should likewise take one of the three two-letter codes `xy` (default), `zx` or `yz`. The spatial scale is adjusted to fit the volume specified by either the **atlas** or **xrange/yrange/zrange** selections. If the specified volume does not fill the entire simulation volume, then points outside the specified volume are handled as determined by the **exterior** setting, which should be either a floating point value, or one of the keywords `boundary` or `error`. In the first case, the floating point value is treated as a default value for points outside the image, and should have a value in the range $[0, 1]$. The multiplier and offset adjustments are made to this value in the same way as to points inside the image. If *ext_flag* is `boundary`, then points outside the image are filled with the value of the closest point on the boundary of the image. If *ext* is `error` (the default), then an error is raised if a value is needed for any point outside the image.

Examples: `rotatecenterstage.mif`, `sample-reflect.mif`.

The available vector field objects are:

Oxs_UniformVectorField: Returns the same constant value regardless of the import position. The Specify block takes one required parameter, **vector**, which is a 3-element

list of the vector to return, and one optional parameter, **norm**, which if specified adjusts the size of export vector to the specified magnitude. For example,

```
Specify Oxs_UniformVectorField {
    norm 1
    vector {1 1 1}
}
```

This object returns the unit vector (a, a, a), where $a = 1/\sqrt{3}$, regardless of the import position.

This class is frequently embedded inline to specify spatially uniform quantities. For example, inside a driver Specify block we may have

```
Specify Oxs_TimeDriver {
    ...
    m0 { Oxs_UniformVectorField {
       vector {1 0 0}
    }}
    ...
}
```

As discussed in the MIF 2 documentation (Sec. 17.3.3, page 242), when embedding `Oxs_UniformVectorField` or `Oxs_UniformScalarField` objects, a notational shorthand is allowed that lists only the required value. The previous example is exactly equivalent to

```
Specify Oxs_TimeDriver {
    ...
    m0 {1 0 0}
    ...
}
```

where an implicit `Oxs_UniformVectorField` object is created with the value of `vector` set to `{1 0 0}`.

Examples: `sample.mif`, `cgtest.mif`.

Oxs_AtlasVectorField: Declares vector values that are defined across individual regions of an `Oxs_Atlas`. The Specify block has the form

```
Specify Oxs_AtlasVectorField:name {
    atlas atlas_spec
    norm magval
    multiplier mult
    default_value vector_field_spec
```

```
    values {
        region1_label vector_field_spec1
        region2_label vector_field_spec2
        ...
    }
}
```

Interpretation is analogous to the `Oxs_AtlasScalarField` specify block, except here the output values are 3 dimensional vectors rather than scalars. Thus the values associated with each region are vector fields rather than scalar fields. Any of the vector field types described in this (Field Objects) section may be used. As usual, one may provided a braced list of three numeric values to request a uniform (spatially homogeneous) vector field with the indicated value.

The optional **norm** parameter causes each vector value to be scaled to have magnitude *magval*. The optional **multiplier** value scales the field values. If both `norm` and `multiplier` are specified, then the field vectors are first normalized before being scaled by the multiplier value.

Examples: `diskarray.mif`, `exchspring.mif`, `imageatlas.mif`, `spinvalve.mif`.

Oxs_ScriptVectorField: Conceptually similar to the `Oxs_ScriptScalarField` scalar field object (page 99), except that the script should return a vector (as a 3 element list) rather than a scalar. In addition to parameters accepted by `Oxs_ScriptScalarField`, `Oxs_ScriptVectorField` also accepts an optional parameter **norm**. If specified, the return values from the script are size adjusted to the specified magnitude. If both `norm` and `multiplier` are specified, then the field vectors are first normalized before being scaled by the multiplier value.

The following example produces a vortex-like unit vector field, with an interior core region pointing parallel to the z-axis. Here the scaling region is specified using an `atlas` reference to an object named ":atlas", which is presumed to be defined earlier in the MIF file. See the `Oxs_ScriptScalarField` sample Specify block for an example using the explicit range option.

```
proc Vortex { xrel yrel zrel } {
    set xrad [expr {$xrel-0.5}]
    set yrad [expr {$yrel-0.5}]
    set normsq [expr {$xrad*$xrad+$yrad*$yrad}]
    if {$normsq <= 0.025} {return "0 0 1"}
    return [list [expr {-1*$yrad}] $xrad 0]
}

Specify Oxs_ScriptVectorField {
    script Vortex
```

```
    norm  1
    atlas :atlas
}
```

See also the `Oxs_MaskVectorField` documentation and the discussion of the `ReadFile` MIF extension command in Sec. 17.3.2 for another `Oxs_ScriptVectorField` example.

Examples: `cgtest.mif`, `ellipsoid.mif`, `manyregions-scriptfields.mif`, `sample-vecreflect.mif`, `stdprob3.mif`, `yoyo.mif`.

Oxs_FileVectorField: Provides a file-specified vector field. The Specify block has the form

```
Specify Oxs_FileVectorField:name {
    file filename
    atlas atlas_spec
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    spatial_scaling { xscale yscale zscale }
    spatial_offset { xoff yoff zoff }
    exterior ext_flag
    norm magnitude
    multiplier mult
}
```

Required values in the Specify block are the name of the input vector field file and the desired scaling parameters. The filename is specified via the **file** entry, which names a file containing a vector field in one of the formats recognized by **avf2ovf**. If **atlas** or **xrange/yrange/zrange** are specified, then the file will be scaled and translated as necessary to fit that scaling region, in the same manner as done, for example, by the `Oxs_ScriptScalarField` and `Oxs_ScriptVectorField` classes. Alternatively, one may specify **spatial_scaling** and **spatial_offset** directly. In this case the vector spatial positions are taken as specified in the file, multiplied component-wise by `(xscale,yscale,zscale)`, and then translated by `(xoff,yoff,zoff)`. If you want to use the spatial coordinates as directly specified in the file, use `(1,1,1)` for spatial_scaling and `(0,0,0)` for spatial_offset.

In all cases, once the input field has been scaled and translated, it is then sub-sampled (zeroth-order fit) as necessary to match the simulation mesh.

The **exterior** flag determines the behavior at "exterior points", i.e., locations (if any) in the simulation mesh that lie outside the extent of the scaled and translated vector field. The *ext_flag* should be either a three-vector, or one of the keywords `boundary` or `error`. If a three-vector is given, then that value is supplied at all exterior points.

If *ext_flag* is set to `boundary`, then the value used is the point on the boundary of the input vector field that is closest to the exterior point. The default setting for *ext_flag* is `error`, which raises an error if there are any exterior points.

The magnitude of the field can be modified by the optional **norm** and **multiplier** attributes. If the norm parameter is given, then each vector in the field will be renormalized to the specified magnitude. If the multiplier parameter is given, then each vector in the field will be multiplied by the given scalar value. If the multiplier value is negative, the field direction will be reversed. If both `norm` and `multiplier` are given, then the field vectors are renormalized before being scaled by the multiplier value.

Examples: `stdprob3.mif`, `yoyo.mif`.

Oxs_RandomVectorField: Similar to `Oxs_RandomScalarField`, but defines a vector field rather than a scalar field that varies spatially in a random fashion. The Specify block has the form:

```
Specify Oxs_RandomVectorField:name {
    min_norm minvalue
    max_norm maxvalue
    base_field vector_field_spec
    cache_grid mesh_spec
}
```

The Specify block takes two required parameters, **min_norm** and **max_norm**. The vectors produced will have magnitude between these two specified values. If `min_norm` = `max_norm`, then the samples are uniformly distributed on the sphere of that radius. Otherwise, the samples are uniformly distributed in the hollow spherical volume with inner radius `min_norm` and outer radius `max_norm`.

There are also two optional parameters, **base_field** and **cache_grid**. If the former is given, then the generated random field is added onto the field specified by `vector_field_spec`. If **base_field** is omitted, the result is equivalent to specifying a zero field.

The **cache_grid** option takes a mesh specification that describes the grid used for spatial discretization caching. If **cache_grid** is not specified, then each call to `Oxs_RandomVectorField` generates a different field. If you want to use the same random vector field in two places (as a base for setting, say anisotropy axes and initial magnetization), then specify **cache_grid** with the appropriate (usually the base problem) mesh. Note also that if **cache_grid** is not used, then on multi-threaded runs the random number generator call sequence among the threads is not deterministic, and so results may varying between runs, even for simulations started with the same random number seed. (`RandomSeed`, Sec. 17.3.2.)

Examples: `diskarray.mif`, `sample2.mif`, `randomshape.mif` `stdprob1.mif`.

Oxs_PlaneRandomVectorField: Similar to `Oxs_RandomVectorField`, except that samples are drawn from 2D planes rather than 3-space. The Specify block has the form

```
Specify Oxs_RandomVectorField:name {
    plane_normal vector_field_spec
    min_norm minvalue
    max_norm maxvalue
    base_field vector_field_spec
    cache_grid mesh_spec
}
```

The **min_norm**, **max_norm**, **base_field** and **cache_grid** parameters have the same meaning as for the `Oxs_RandomVectorField` class. The additional parameter, **plane_normal**, specifies a vector field that at each point provides a vector that is orthogonal to the plane from which the random vector at that point is to be drawn. If the vector field is specified explicitly as three real values, then a spatially uniform vector field is produced and all the random vectors will lie in the same plane. More generally, however, the normal vectors (and associated planes) may vary from point to point. As a special case, if a normal vector at a point is the zero vector, then no planar restriction is made and the resulting random vector is drawn uniformly from a hollow ball in three space satisfying the minimum/maximum norm constraints.

Example: `sample2.mif`.

Oxs_ScriptOrientVectorField: Compare to the `Oxs_ScriptOrientScalarField` class (page 102). The Specify block has the form:

```
Specify Oxs_ScriptOrientVectorField:name {
    field vector_field_spec
    script Tcl_script
    script_args { args_request }
    atlas atlas_spec
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
}
```

The interpretation of the specify block and the operation of the Tcl script is exactly the same as for the `Oxs_ScriptOrientScalarField` class, except the input **field** and the resulting field are vector fields instead of scalar fields.

Note that the "orientation" transformation is applied to the import spatial coordinates only, not the output vector. For example, if the `field` value represents a shaped vector field, and the `script` proc is a rotation transformation, then the resulting vector field shape will be rotated as compared to the original vector field, but the output vectors

themselves will still point in their original directions. In such cases one may wish to compose the `Oxs_ScriptOrientVectorField` with a `Oxs_ScriptVectorField` object (page 108) to rotate the output vectors as well. This situation occurs also with the `Oxs_AffineOrientVectorField` class. See the `Oxs_AffineTransformVectorField` class documentation (page 112) for an example illustrating the composition of an object of that class with a `Oxs_AffineOrientVectorField` object.

Example: `sample-vecreflect.mif`.

Oxs_AffineOrientVectorField: Compare to the `Oxs_AffineOrientScalarField` class (page 103). The Specify block has the form:

```
Specify Oxs_AffineOrientVectorField:name {
    field vector_field_spec
    M { matrix_entries ... }
    offset { off_x off_y off_z }
    inverse invert_flag
    inverse_slack slack
}
```

The interpretation of the specify block and the affine transformation is exactly the same as for the `Oxs_AffineOrientScalarField` class, except the input **field** and the resulting field are vector fields instead of scalar fields.

As explained in the `Oxs_ScriptOrientVectorField` documentation, the "orientation" transformation is applied to the import spatial coordinates only, not the output vector. If one wishes to rotate the output vectors, then a `Oxs_AffineTransformVectorField` object may be applied with the opposite rotation. See that section for an example.

Examples: `yoyo.mif`, `sample-vecrotate.mif`.

Oxs_AffineTransformVectorField: This class applies an affine transform to the output of a vector field. It is similar to the `Oxs_AffineTransformScalarField` class (page 105), except that in this case the affine transform is applied to a vector instead of a scalar. The Specify block has the form:

```
Specify Oxs_AffineTransformVectorField:name {
    field vector_field_spec
    M { matrix_entries ... }
    offset { off_x off_y off_z }
    inverse invert_flag
    inverse_slack slack
}
```

Because the output from **field** is a 3-vector, the transform defined by **M** and **offset** requires `M` to be a 3×3 matrix and `offset` to be a 3-vector. Thus, if $\boldsymbol{v}(\boldsymbol{x})$ represents the vector field specified by the **field** value, then the resulting vector field is $M.\boldsymbol{v}(\boldsymbol{x}) + \mathbf{off}$.

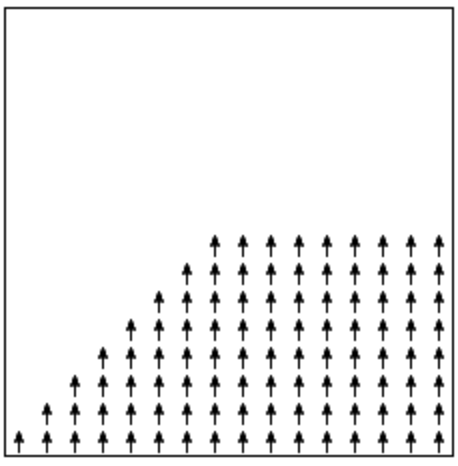
`M` is described by a list of from one to nine entries, in exactly the same manner as for the `Oxs_AffineOrientVectorField` and `Oxs_AffineOrientScalarField` classes (page 105). The interpretation of **offset**, **inverse**, and **inverse_slack** is also the same. In particular, if *invert_flag* is 1, then the resulting vector field is $M^{-1}.(\boldsymbol{v}(\boldsymbol{x}) - \textbf{off})$.

The following example illustrates combining a `Oxs_AffineTransformVectorField` with a `Oxs_AffineOrientVectorField` to completely rotate a vector field.

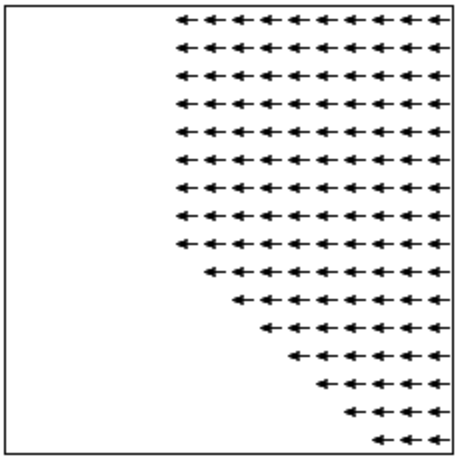
```
Specify Oxs_BoxAtlas:atlas {
  xrange {-80e-9 80e-9}
  yrange {-80e-9 80e-9}
  zrange {0  40e-9}
}

proc Trap { x y z } {
     if {$y<=$x && $y<=0.5} {return [list 0 1 0]}
     return [list 0 0 0]
}

Specify Oxs_ScriptVectorField:trap {
   script Trap
   atlas :atlas
}

Specify Oxs_AffineOrientVectorField:orient {
   field :trap
   M { 0 -1 0
       1  0 0
       0  0 1 }
   offset { -20e-9 0 0 }
   inverse 1
}

Specify Oxs_AffineTransformVectorField:rot {
   field :orient
   M { 0 -1 0
       1  0 0
       0  0 1 }
}

proc Threshold { vx vy vz } {
   set magsq [expr {$vx*$vx+$vy*$vy+$vz*$vz}]
   if {$magsq>0} {return 8e5}
```

```
   return 0.0
}

Specify Oxs_ScriptScalarField:Ms {
  vector_fields :rot
  script Threshold
  script_args vectors
}

Specify Oxs_TimeDriver {
 m0 :rot
 Ms :Ms
 stopping_dm_dt 0.01
 evolver :evolve
 mesh :mesh
}
```

The base field here is given by the `Oxs_ScriptVectorField:trap` object, which produces a vector field having a trapezoidal shape with the non-zero vectors pointing parallel to the y-axis. The `:orient` and `:rot` transformations rotate the shape and the vectors counterclockwise 90°. Additionally, the `offset` option in `:orient` translates the shape 20 nm towards the left. The original and transformed fields are illustrated below.



Original field



Rotated field

Example: `sample-vecrotate.mif`.

Oxs_MaskVectorField: Multiplies a vector field pointwise by a scalar vector field (the mask) to produce a new vector field. The Specify block has the form:

```
Specify Oxs_MaskVectorField:name {
```

```
    mask scalar_field_spec
    field vector_field_spec
}
```

This functionality can be achieved, if in a somewhat more complicated fashion, with the `Oxs_ScriptVectorField` class. For example, given a scalar field `:mask` and a vector field `:vfield`, this example using the `Oxs_MaskVectorField` class

```
Specify Oxs_MaskVectorField {
   mask :mask
   field :vfield
}
```

is equivalent to this example using the `Oxs_ScriptVectorField` class

```
proc MaskField { m vx vy vz } {
   return [list [expr {$m*$vx}] [expr {$m*$vy}] [expr {$m*$vz}]]
}

Specify Oxs_ScriptVectorField {
  script MaskField
  script_args {scalars vectors}
  scalar_fields { :mask }
  vector_fields { :vfield }
}
```

Of course, the `Oxs_ScriptVectorField` approach is easily generalized to much more complicated and arbitrary combinations of scalar and vector fields.

Example: `rotatecenterstage.mif`.

Oxs_ImageVectorField: This class creates a vector field using an image. The Specify block has the form

```
Specify Oxs_ImageVectorField:name {
    image pic
    multiplier mult
    vx_multiplier xmult
    vy_multiplier ymult
    vz_multiplier zmult
    vx_offset xoff
    vy_offset yoff
    vz_offset zoff
    norm norm_magnitude
    viewplane view
```

```
    atlas atlas_spec
    xrange { xmin xmax }
    yrange { ymin ymax }
    zrange { zmin zmax }
    exterior ext_flag
}
```

The **image** is interpreted as a three-color map, yielding a vector field where each (x,y,z) component is determined by the red, green, and blue color components, respectively.. . .

The **viewplane**, **atlas**, **xrange/yrange/zrange**, and **exterior** are treated the same as for the `Oxs_ImageScalarField` class (q.v.)

Examples: `NONE`.

7.3.7 MIF Support Classes

Oxs_LabelValue: A convenience object that holds label + value pairs. `Oxs_LabelValue` objects may be referenced via the standard `attributes` field in other Specify blocks, as in this example:

```
Specify Oxs_LabelValue:probdata {
  alpha 0.5
  start_dm 0.01
}

Specify Oxs_EulerEvolve {
  attributes :probdata
}
```

The Specify block string for `Oxs_LabelValue` objects is an arbitrary Tcl list with an even number of elements. The first element in each pair is interpreted as a label, the second as the value. The `attribute` option causes this list to be dropped verbatim into the surrounding object. This technique is most useful if the label + value pairs in the `Oxs_LabelValue` object are used in multiple Specify blocks, either inside the same MIF file, or across several MIF files into which the `Oxs_LabelValue` block is imported using the `ReadFile` MIF extension command.

Examples: The MIF files `sample-rotate.mif` and `sample-reflect.mif` use the `Oxs_LabelValue` object stored in the `sample-attributes.tcl` file.

Refer to Sec. 17.3 for details on the base MIF 2 format specification.

7.4 Contributed Oxs_Ext Child Classes

The `Oxs_Ext` classes discussed in the previous section were written and are maintained by the OOMMF core development team. Additional `Oxs_Ext` classes are available from third-parties or may be written by the end user (see the *OOMMF Programming Manual*[1]). Available extensions known to the OOMMF developers are listed on the Oxs Extensions Modules[2] page. These extensions are included in OOMMF distributions, current as of the time of release.

Contributed extensions can be found under `oommf/app/oxs/contrib/` in individual directories. These are "installed" by (1) using the **oxspkg install** command**oxspkg install** to copy the source code from the distribution area into `oommf/app/oxs/local/`, and then (2) running **pimake** to compile and link the extension into the Oxs executable.

All of the extensions distributed in an OOMMF release that do not require libraries beyond those needed to run OOMMF are distributed in the installed state. It is up to the end user to install additional libraries as needed if they wish to activate other extensions. See the `requires` subcommand in the **oxspkg** documentation for details.

Note that extensions are distributed as source code; at this time there are no provisions for binary distribution of `Oxs_Ext` modules. This means that to add extensions to your OOMMF installation you need to have a supported C++ compiler, and you must run **pimake** to build the extensions. If you are running OOMMF on Windows with pre-built binaries, then you should first `cd` to the OOMMF root directory and run

```
$ tclsh oommf.tcl pimake distclean
```

to completely delete all pre-built binary files. Afterwards run

```
$ tclsh oommf.tcl pimake
```

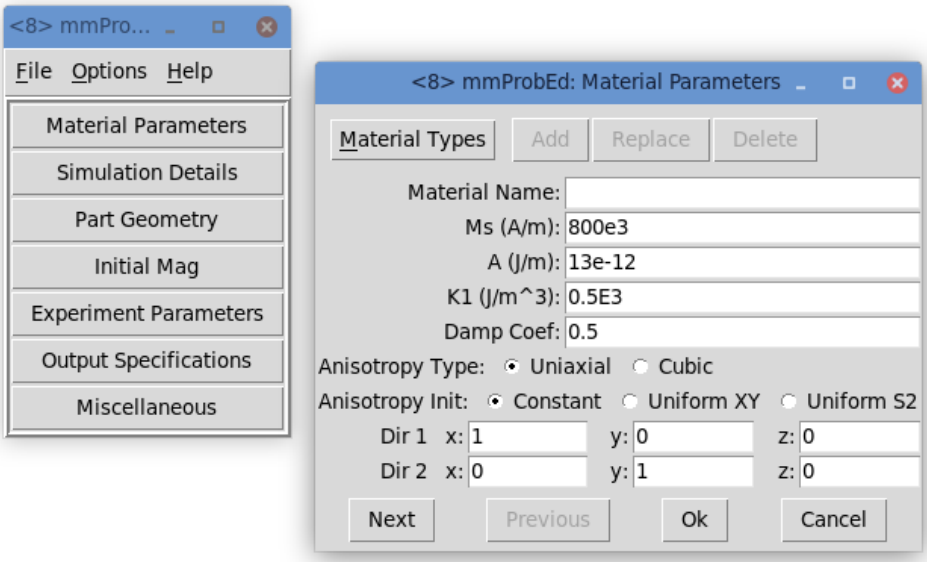
to create a fresh build of OOMMF. This ensures that the new extension binaries are compatible with the rest of the OOMMF infrastructure.

Third party extensions are not maintained or documented by the OOMMF development team. Such extensions are provided "as-is." Some extension authors may provide limited support for their extensions, but be aware that extensions are generally provided as a public service to the community and support is not compulsory.

[1] https://math.nist.gov/oommf/doc/
[2] https://math.nist.gov/oommf/contrib/oxsext/

Chapter 8

Micromagnetic Problem Editor: mmProbEd



Overview

The application **mmProbEd** provides a user interface for creating and editing micromagnetic problem descriptions in the MIF 1.1 and MIF 1.2 formats. **mmProbEd** also acts as a server, supplying problem descriptions to running **mmSolve2D** micromagnetic solvers.

Launching

mmProbEd may be started either by selecting the **mmProbEd** button on **mmLaunch**, or from the command line via

```
tclsh oommf.tcl mmProbEd [standard options] [-net <0|1>]
```

-net <0|1> Disable or enable a server which provides problem descriptions to other applications. By default, the server is enabled. When the server is disabled, **mmProbEd** is only useful for editing problem descriptions and saving them to files.

Inputs

The menu selection **File|Open...** displays a dialog box for selecting a file from which to load a MIF problem description. Several example files are included in the OOMMF release in the directory `oommf/app/mmpe/examples`. At startup, **mmProbEd** loads the problem contained in `oommf/app/mmpe/init.mif` as an initial problem. Note: When loading a file, **mmProbEd** discards comments and moves records it does not understand to the bottom of its output file. Use the **FileSource** application to serve unmodified problem descriptions.

Outputs

The menu selection **File|Save as...** displays a dialog box for selecting/entering a file in which the problem description currently held by **mmProbEd** is to be saved. Because the internal data format use by **mmProbEd** is an unordered array that does not include comments (or unrecognized records), the simple operation of reading in a MIF file and then writing it back out may alter the file.

Each instance of **mmProbEd** contains exactly one problem description at a time. When the option `-net 1` is active (the default), each also services requests from client applications (typically solvers) for the problem description it contains.

Controls

The **Options** menu allows selection of MIF output format; either MIF 1.1 or MIF 1.2 may be selected. This affects both **File|Save as...** file and **mmSolve2D** server output. See the MIF 1.2 format documentation for details on the differences between these formats.

The main panel in the **mmProbEd** window contains buttons corresponding to the sections in a MIF 1.x problem description. Selecting a button brings up another window through which the contents of that section may be edited. The MIF sections and the elements they contain are described in detail in the MIF 1.1 and MIF 1.2 documentation. Only one editing

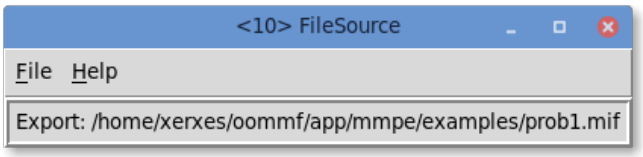
window is displayed at a time. The windows may be navigated in order using their **Next** or **Previous** buttons.

The **Material Parameters** edit window includes a pull-down list of pre-configured material settings. **NOTE:** These values should *not* be taken as standard reference values for these materials. The values are only approximate, and are included for convenience, and as examples for users who wish to supply their own material types with symbolic names. To introduce additional material types, edit the `Material Name`, `Ms`, `A`, `K1`, and `Anisotropy Type` values as desired, and hit the **Add** button. (The `Damp Coef` and `Anistropy Init` settings are not affected by the Material Types selection.) The Material Name entry will appear in red if it does not match any name in the Material Types list, or if the name matches but one or more of the material values differs from the corresponding value as stored in the Material Types list. You can manage the Material Types list with the **Replace** and **Delete** buttons, or by directly editing the file `oommf/app/mmpe/local/materials`; follow the format of other entries in that file. The format is the same as in the default `oommf/app/mmpe/materials` file included with the OOMMF distribution.

The menu selection **File|Exit** terminates the **mmProbEd** application. The menu **Help** provides the usual help facilities.

Chapter 9

Micromagnetic Problem File Source: FileSource



Overview

The application **FileSource** provides the same service as **mmProbEd**, supplying MIF 1.x problem descriptions to running **mmSolve2D** micromagnetic solvers. As the MIF specification evolves, **mmProbEd** may lag behind. There may be new fields in the MIF specification that **mmProbEd** is not capable of editing, or which **mmProbEd** may not pass on to solvers after loading them in from a file. To make use of such fields, a MIF file may need to be edited "by hand" using a general purpose text editor. **FileSource** may then be used to supply the MIF problem description contained in a file to a solver without danger of corrupting its contents.

Launching

FileSource must be launched from the command line. You may specify on the command line the MIF problem description file it should serve to client applications. The command line is

```
tclsh oommf.tcl FileSource [standard options] [filename]
```

Although **FileSource** does not appear on the list of **Programs** that **mmLaunch** offers to launch, running copies do appear on the list of **Running Applications** since they do provide a service registered with the account service directory.

Inputs

FileSource takes its MIF problem description from the file named on the command line, or from a file selected through the **File|Open** dialog box. No checking of the file contents against the MIF specification is performed. The file contents are passed uncritically to any client application requesting a problem description. Those client applications should raise errors when presented with invalid problem descriptions.

Outputs

Each instance of **FileSource** provides the contents of exactly one file at a time. The file name is displayed in the **FileSource** window to help the user associate each instance of **FileSource** with the data file it provides. Each instance of **FileSource** accepts and services requests from client applications (typically solvers) for the contents of the file it exports.

The contents of the file are read at the time of the client request, so if the contents of a file change between the time of the **FileSource** file selection and the arrival of a request from a client, the new contents will be served to the client application.

Controls

The menu selection **File|Exit** terminates the **FileSource** application. The **Help** menu provides the usual help facilities.

Chapter 10

The 2D Micromagnetic Solver

The OOMMF 2D micromagnetic computation engine, mmSolve, is capable of solving problems defined on a two-dimensional grid of square cells with three-dimensional spins. This solver is deprecated as it is less flexible and extensible than the Oxs solver. Users are encouraged to migrate to Oxs.

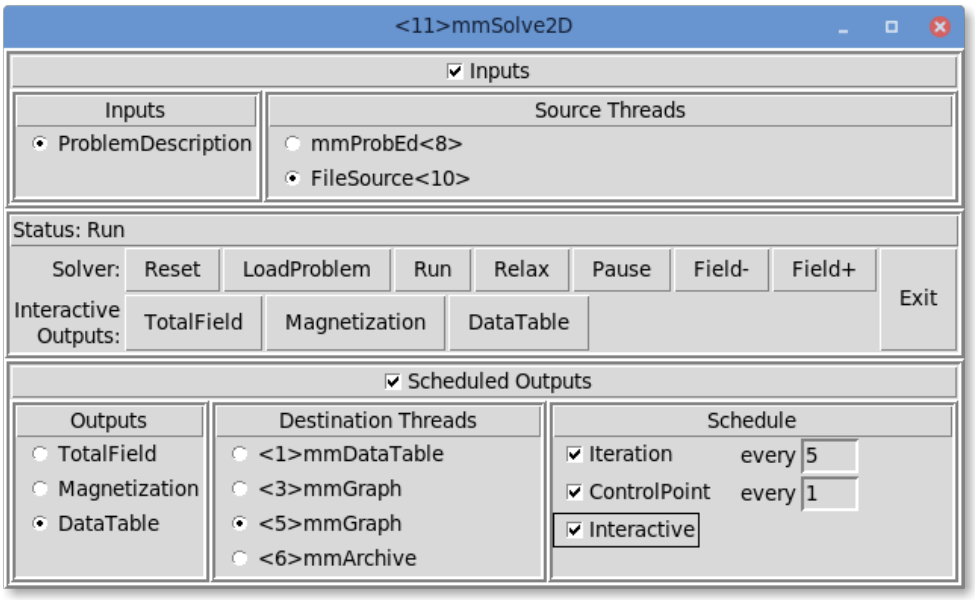
There are two interfaces provided to mmSolve, the interactive **mmSolve2D** interface and the command line driven **batchsolve** interface which can be used in conjunction with the OOMMF Batch System.

Problem definition for mmSolve is accomplished using input files in the MIF 1.1 format. Please note that this format is incompatible with the newer MIF 2.x format used by the Oxs solver. However, the command line utility **mifconvert** can be used to aid conversion from the MIF 1.1 format to MIF 2.1.

mmSolve will also accept files in the MIF 1.2 format format, provided the `CellSize` record meets the restriction that the x- and y-dimensions are the same, and that the z-dimension equals the part thickness.

Note on Tk dependence: If a problem is loaded that uses a bitmap mask file, and if that mask file is not in the PPM P3 (text) format, then **mmSolve2D** will launch any2ppm to convert it into the PPM P3 format. Since **any2ppm** requires Tk, at the time the mask file is read a valid display must be available. See the **any2ppm** documentation for details.

10.1 The 2D Micromagnetic Interactive Solver: mmSolve2D



Overview

The application **mmSolve2D** is a micromagnetic computation engine capable of solving problems defined on two-dimensional square grids of three-dimensional spins. Within the OOMMF architecture, **mmSolve2D** is both a server and a client application. **mmSolve2D** is a client of problem description server applications, data table display and storage applications, and vector field display and storage applications. **mmSolve2D** is the server of a solver control service for which the only client is **mmLaunch**. It is through this service that **mmLaunch** provides a user interface window (shown above) on behalf of **mmSolve2D**.

Launching

mmSolve2D is started from the command line via

```
tclsh oommf.tcl mmSolve2D [standard options] [-restart <0|1>]
```

-restart <0|1> Affects the behavior of the solver when a new problem is loaded. Default value is 0. When launched with `-restart 1`, the solver will look for *`basename`*`.log` and *`basename`*`*.omf` files to restart a previous run from the last saved state (where *`basename`* is the "Base Output Filename" specified in the input MIF 1.1 problem specification file (Sec. 17.1)). If these files cannot be found, then a warning is issued and the solver falls back to the default behavior (`-restart 0`) of starting the problem from scratch. The specified `-restart` setting holds for **all** problems fed to the solver,

not just the first. (There is currently no interactive way to change the value of this switch.)

Once launched, an instance of **mmSolve2D** will appear next to a checkbutton on the Running Applications list in **mmLaunch**; selecting the checkbutton will open the **mmSolve2D** GUI interface. (**mmSolve2D** does not appear on the **mmLaunch** Programs launch list because it has been deprecated in favor of **Oxsii**. However, users may follow the edit instructions in the file `oommf/app/mmlaunch/mmLaunchPrograms.tcl` to reinstate **mmSolve2D** to the launch list.)

The GUI window is divided into panels, providing user interfaces to the **Inputs**, **Outputs**, and **Controls** of **mmSolve2D**.

Inputs

The top panel of the user interface window may be opened and closed by toggling the **Inputs** checkbutton. When open, the **Inputs** panel reveals two subpanels. The left subpanel contains a list of the inputs required by **mmSolve2D**. There is only one item in the list: **ProblemDescription**. When **ProblemDescription** is selected, the right subpanel (labeled **Source Threads**) displays a list of applications that can supply a problem description. The user selects from among the listed applications the one from which **mmSolve2D** should request a problem description.

Outputs

When **mmSolve2D** has outputs available to be controlled, a **Scheduled Outputs** checkbutton appears in the user interface window. Toggling the **Scheduled Outputs** checkbutton causes a bottom panel to open and close in the user interface window. When open, the **Scheduled Outputs** panel contains three subpanels. The **Outputs** subpanel is filled with a list of the types of output **mmSolve2D** can generate while solving the loaded problem. The three elements in this list are **TotalField**, for the output of a vector field representing the total effective field, **Magnetization**, for the output of a vector field representing the current magnetization state of the grid of spins, and **DataTable**, for the output of a table of data values describing other quantities of interest calculated by **mmSolve2D**.

Upon selecting one of the output types from the **Outputs** subpanel, a list of applications appears in the **Destination Threads** subpanel which provide a display and/or storage service for the type of output selected. The user may select from this list those applications to which the selected type of output should be sent.

For each application selected, a final interface is displayed in the **Schedule** subpanel. Through this interface the user may set the schedule according to which the selected type of data is sent to the selected application for display or storage. The schedule is described relative to events in **mmSolve2D**. An **Iteration** event occurs at every step in the solution of the ODE. A **ControlPoint** event occurs whenever the solver determines that a control point specification is met. (Control point specs are discussed in the Experiment parameters

paragraph in the MIF 1.1 documentation, and are triggered by solver equilibrium, simulation time, and iteration count conditions.) An **Interactive** event occurs for a particular output type whenever the corresponding "Interactive Outputs" button is clicked in the **Runtime Control** panel. The **Interactive** schedule gives the user the ability to interactively force data to be delivered to selected display and storage applications. For the **Iteration** and **ControlPoint** events, the granularity of the output delivery schedule is under user control. For example, the user may elect to send vector field data describing the current magnetization state to an **mmDisp** instance for display every 25 iterations of the ODE, rather than every iteration.

The quantities included in **DataTable** output produced by **mmSolve2D** include:

- **Iteration:** The iteration count of the ODE solver.
- **Field Updates:** The number of times the ODE solver has calculated the effective field.
- **Sim Time (ns):** The elapsed simulated time.
- **Time Step (ns):** The interval of simulated time spanned by the last step taken in the ODE solver.
- **Step Size:** The magnitude of the last step taken by the ODE solver as a normalized value. (This is currently the time step in seconds, multiplied by the gyromagnetic ratio times the damping coefficient times M_s.)
- **Bx, By, Bz (mT):** The x, y, and z components of the nominal applied field.
- **B (mT):** The magnitude of the nominal applied field (always non-negative).
- **|m x h|:** The maximum of the point-wise quantity $\|\mathbf{M} \times \mathbf{H}_{\text{eff}}\|/M_s^2$ over all the spins. This "torque" value is used to test convergence to an equilibrium state (and raise control point -torque events).
- **Mx/Ms, My/Ms, Mz/Ms:** The x, y, and z components of the average magnetization of the magnetically active elements of the simulated part.
- **Total Energy (J/m^3):** The total average energy density for the magnetically active elements of the simulated part.
- **Exchange Energy (J/m^3):** The component of the average energy density for the magnetically active elements of the simulated part due to exchange interactions.
- **Anisotropy Energy (J/m^3):** The component of the average energy density for the magnetically active elements of the simulated part due to crystalline and surface anisotropies.

- **Demag Energy (J/m^3):** The component of the average energy density for the magnetically active elements of the simulated part due to self-demagnetizing fields.
- **Zeeman Energy (J/m^3):** The component of average energy density for the magnetically active elements of the simulated part due to interaction with the applied field.
- **Max Angle:** The maximum angle (in degrees) between the magnetization orientation of any pair of neighboring spins in the grid. (The neighborhood of a spin is the same as that defined by the exchange energy calculation.)

In addition, the solver automatically keeps a log file that records the input problem specification and miscellaneous runtime information. The name of this log file is *basename*.log, where *basename* is the "Base Output Filename" specified in the input problem specification. If this file already exists, then new entries are appended to the end of the file.

Controls

The middle section of the user interface window contains a series of buttons providing user control over the solver. After a problem description server application has been selected, the **LoadProblem** button triggers a fetch of a problem description from the selected server. The **LoadProblem** button may be selected at any time to (re-)load a problem description from the currently selected server. After loading a new problem the solver goes automatically into a paused state. (If no problem description server is selected when the **LoadProblem** button is invoked, nothing will happen.) The **Reset** button operates similarly, except that the current problem specifications are used.

Once a problem is loaded, the solver can be put into any of three states: run, relax and pause. Selecting **Relax** puts the solver into the "relax" state, where it runs until a control point is reached, after which the solver pauses. If the **Relax** button is reselected after reaching a control point, then the solver will simply re-pause immediately. The **Field+** or **Field–** button must be invoked to change the applied field state. (Field state schedules are discussed below.) The **Run** selection differs in that when a control point is reached, the solver automatically steps the nominal applied field to the next value, and continues. In "run" mode the solver will continue to process until there are no more applied field states in the problem description. At any time the **Pause** button may be selected to pause the solver. The solver will stay in this state until the user reselects either **Run** or **Relax**. The current state of the solver is indicated in the **Status** line in the center panel of the user interface window.

The problem description (MIF 1.x format) specifies a fixed applied field schedule. This schedule defines an ordered list of applied fields, which the solver in "run" mode steps through in sequence. The **Field–** and **Field+** buttons allow the user to interactively adjust the applied field sequence. Each click on the **Field+** button advances forward one step through the specified schedule, while **Field–** reverses that process. In general, the step direction is *not*

related to the magnitude of the applied field. Also note that hitting these buttons does not generate a "ControlPoint" event. In particular, if you are manually accelerating the progress of the solver through a hysteresis loop, and want to send non-ControlPoint data to a display or archive widget before advancing the field, then you must use the appropriate "Interactive Output" button.

The second row of buttons in the interaction control panel, **TotalField**, **Magnetization** and **DataTable**, allow the user to view the current state of the solver at any time. These buttons cause the solver to send out data of the corresponding type to all applications for which the "Interactive" schedule button for that data type has been selected, as discussed in the Outputs section above.

At the far right of the solver controls is the **Exit** button, which terminates **mmSolve2D**. Simply closing the user interface window does not terminate **mmSolve2D**, but only closes the user interface window. To kill the solver the **Exit** button must be pressed.

Details

Given a problem description, **mmSolve2D** integrates the Landau-Lifshitz equation [13, 18]

$$\frac{d\mathbf{M}}{dt} = -|\bar{\gamma}|\,\mathbf{M}\times\mathbf{H}_{\text{eff}} - \frac{|\bar{\gamma}|\alpha}{M_s}\,\mathbf{M}\times(\mathbf{M}\times\mathbf{H}_{\text{eff}})\,, \tag{10.1}$$

where

$\mathbf{M}$	is the pointwise magnetization (A/m),
$\mathbf{H}_{\text{eff}}$	is the pointwise effective field (A/m),
$\bar{\gamma}$	is the Landau-Lifshitz gyromagnetic ratio (m/(A·s)),
α	is the damping coefficient (dimensionless).

(Compare to (7.2), page 79.)

The effective field is defined as

$$\mathbf{H}_{\text{eff}} = -\mu_0^{-1}\frac{\partial E}{\partial \mathbf{M}}.$$

The average energy density E is a function of $\mathbf{M}$ specified by Brown's equations [5], including anisotropy, exchange, self-magnetostatic (demagnetization) and applied field (Zeeman) terms.

The micromagnetic problem is impressed upon a regular 2D grid of squares, with 3D magnetization spins positioned at the centers of the cells. Note that the constraint that the grid be composed of square elements takes priority over the requested size of the grid. The actual size of the grid used in the computation will be the nearest integral multiple of the grid's cell size to the requested size. It is important when comparing the results from grids with different cell sizes to account for the possible change in size of the overall grid.

The anisotropy and applied field energy terms are calculated assuming constant magnetization in each cell. The exchange energy is calculated using the eight-neighbor bilinear

interpolation described in [6], with Neumann boundary conditions. The more common four-neighbor scheme is available as a compile-time option. Details can be found in the source-code file `oommf/app/mmsolve/magelt.cc`.

The self-magnetostatic field is calculated as the convolution of the magnetization against a kernel that describes the cell to cell magnetostatic interaction. The convolution is evaluated using fast Fourier transform (FFT) techniques. Several kernels are supported; these are selected as part of the problem description in MIF 1.x format. Each kernel represents a different interpretation of the discrete magnetization. The recommended model is `ConstMag`, which assumes the magnetization is constant in each cell, and computes the average demagnetization field through the cell using formulae from [2, 25, 11].

The Landau-Lifshitz ODE (10.1) is integrated using a second order predictor-corrector technique of the Adams type. The right side of (10.1) at the current and previous step is extrapolated forward in a linear fashion, and is integrated across the new time interval to obtain a quadratic prediction for $\mathbf{M}$ at the next time step. At each stage the spins are renormalized to M_s before evaluating the energy and effective fields. The right side of (10.1) is evaluated at the predicted $\mathbf{M}$, which is then combined with the value at the current step to produce a linear interpolation of $d\mathbf{M}/dt$ across the new interval. This is then integrated to obtain the final estimate of $\mathbf{M}$ at the new step. The local (one step) error of this procedure should be $\mathcal{O}(\Delta t^3)$.

The step is accepted if the total energy of the system decreases, and the maximum error between the predicted and final $\mathbf{M}$ is smaller than a nominal value. If the step is rejected, then the step size is reduced and the integration procedure is repeated. If the step is accepted, then the error between the predicted and final $\mathbf{M}$ is used to adjust the size of the next step. No fixed ratio between the previous and current time step is assumed.

A fourth order Runge-Kutta solver is used to prime the predictor-corrector solver, and is used as a backup in case the predictor-corrector fails to find a valid step. The Runge-Kutta solver is not selectable as the primary solver at runtime, but may be so selected at compile time by defining the `RUNGE_KUTTA_ODE` macro. See the file `oommf/app/mmsolve/grid.cc` for all details of the integration procedure.

For a given applied field, the integration continues until a control point (cf. Experiment parameters, Sec. 17.1.5) is reached. A control point event may be raised by the ODE iteration count, elapsed simulation time, or by the maximum value of $\|\mathbf{M}\times\mathbf{H}_{\text{eff}}\|/M_s^2$ dropping below a specified control point -torque value (implying an equilibrium state has been reached).

Depending on the problem size, **mmSolve2D** can require a good deal of working memory. The exact amount depends on a number of factors, but a reasonable estimate is 5 MB + 1500 bytes per cell. For example, a 1 μm $\times$ 1 μm part discretized with 5 nm cells will require approximately 62 MB.

Known Bugs

mmSolve2D requires the damping coefficient to be non-zero. See the MIF 1.1 documentation for details on specifying the damping coefficient.

When multiple copies of **mmLaunch** are used, each can have its own interface to a running copy of **mmSolve2D**. When the interface presented by one copy of **mmLaunch** is used to set the output schedule in **mmSolve2D**, those settings are not reflected in the interfaces presented by other copies of **mmLaunch**. For example, although the first interface sets a schedule that DataTable data is to be sent to an instance of **mmGraph** every third Iteration, there is no indication of that schedule presented to the user in the second interface window. It is unusual to have more than one copy of **mmLaunch** running simultaneously. However, this bug also appears when one copy of **mmLaunch** is used to load a problem and start a solver, and later a second copy of **mmLaunch** is used to monitor the status of that running solver.

10.2 OOMMF 2D Micromagnetic Solver Batch System

The OOMMF Batch System (OBS) provides a scriptable interface to the same micromagnetic solver engine used by **mmSolve2D**, in the form of three Tcl applicatons (**batchmaster**, **batchslave**, and **batchsolve**) that provide support for complex job scheduling. All OBS script files are in the OOMMF distribution directory `app/mmsolve/scripts`.

Unlike much of the OOMMF package, the OBS is meant to be driven primarily from the command line or shell (batch) script. OBS applications are launched from the command line using the bootstrap application.

10.2.1 2D Micromagnetic Solver Batch Interface: batchsolve

Overview

The application **batchsolve** provides a simple command line interface to the OOMMF 2D micromagnetic solver engine.

Launching

The application **batchsolve** is launched by the command line:

```
tclsh oommf.tcl batchsolve [standard options]
   [-end_exit <0|1>] [-end_paused] [-interface <0|1>] \
   [-restart <0|1>] [-start_paused] [file]
```

where

-end_exit <0|1> Whether or not to explicitly call exit at bottom of `batchsolve.tcl`. When launched from the command line, the default is to exit after solving the problem in `file`. When sourced into another script, like `batchslave.tcl`, the default is to wait for the caller script to provide further instructions.

-interface <0|1> Whether to register with the account service directory application, so that **mmLaunch**, can provide an interactive interface. Default = 1 (do register), which will automatically start account service directory and host service directory applications as necessary.

-start_paused Pause solver after loading problem.

-end_paused Pause solver and enter event loop at bottom of `batchsolve.tcl` rather than just falling off the end (the effect of which will depend on whether or not Tk is loaded).

-restart <0|1> Determines solver behavior when a new problem is loaded. If 1, then the solver will look for *basename*`.log` and *basename*`*.omf` files to restart a previous run from the last saved state (where *basename* is the "Base Output Filename" specified in the input problem specification). If these files cannot be found, then a warning is issued and the solver falls back to the default behavior (equivalent to `-restart 0`) of starting the problem from scratch. The specified `-restart` setting holds for **all** problems fed to the solver, not just the first.

file Immediately load and run the specified MIF 1.x file.

The input file `file` should contain a Micromagnetic Input Format 1.x problem description, such as produced by **mmProbEd**. The batch solver searches several directories for this file, including the current working directory, the `data` and `scripts` subdirectories, and parallel directories relative to the directories `app/mmsolve` and `app/mmpe` in the OOMMF distribution. Refer to the `mif_path` variable in `batchsolve.tcl` for the complete list.

If `-interface` is set to 1 (enabled), **batchsolve** registers with the account service directory application, and **mmLaunch** will be able to provide an interactive interface. Using this interface, **batchsolve** may be controlled in a manner similar to **mmSolve2D**. The interface allows you to pause, un-pause, and terminate the current simulation, as well as to attach data display applications to monitor the solver's progress. If more interactive control is needed, **mmSolve2D** should be used.

If `-interface` is 0 (disabled), **batchsolve** does not register, leaving it without an interface, unless it is sourced into another script (e.g., `batchslave.tcl`) that arranges for an interface on the behalf of **batchsolve**.

Use the `-start_paused` switch to monitor the progress of **batchsolve** from the very start of a simulation. With this switch the solver will be paused immediately after loading the specified MIF file, so you can bring up the interactive interface and connect display applications before the simulation begins. Start the simulation by selecting the **Run** command from the interactive interface. This option cannot be used if `-interface` is disabled.

The `-end_paused` switch ensures that the solver does not automatically terminate after completing the specified simulation. This is not generally useful, but may find application when **batchsolve** is called from inside a Tcl-only wrapper script.

Note on Tk dependence: If a problem is loaded that uses a bitmap mask file, and if that mask file is not in the PPM P3 (text) format, then **batchsolve** will launch any2ppm to

convert it into the PPM P3 format. Since **any2ppm** requires Tk, at the time the mask file is read a valid display must be available. See the **any2ppm** documentation for details.

Output

The output may be changed by a Tcl wrapper script, but the default output behavior of **batchsolve** is to write tabular text data and the magnetization state at the control point for each applied field step. The tabular data are appended to the file *basename*`.odt`, where *basename* is the "Base Output Filename" specified in the input MIF 1.x file. See the routine `GetTextData` in `batchsolve.tcl` for details, but at present the output consists of the solver iteration count, nominal applied field **B**, reduced average magnetization **m**, and total energy. This output is in the ODT file format.

The magnetization data are written to a series of OVF (OOMMF Vector Field) files, *basename*`.field`*nnnn*`.omf`, where *nnnn* starts at `0000` and is incremented at each applied field step. (The ASCII text header inside each file records the nominal applied field at that step.) These files are viewable using **mmDisp**.

The solver also automatically appends the input problem specification and miscellaneous runtime information to the log file *basename*`.log`.

Programmer's interface

In addition to directly launching **batchsolve** from the command line, `batchsolve.tcl` may also be sourced into another Tcl script that provides additional control structures. Within the scheduling system of OBS, `batchsolve.tcl` is sourced into **batchslave**, which provides additional control structures that support scheduling control by **batchmaster**. There are several variables and routines inside `batchsolve.tcl` that may be accessed and redefined from such a wrapper script to provide enhanced functionality.

Global variables

mif A Tcl handle to a global `mms_mif` object holding the problem description defined by the input MIF 1.x file.

solver A Tcl handle to the `mms_solver` object.

search_path Directory search path used by the `FindFile` proc (see below).

Refer to the source code and sample scripts for details on manipulation of these variables.

Batchsolve procs

The following Tcl procedures are designed for external use and/or redefinition:

SolverTaskInit Called at the start of each task.

BatchTaskIterationCallback Called after each iteration in the simulation.

BatchTaskRelaxCallback Called at each control point reached in the simulation.

SolverTaskCleanup Called at the conclusion of each task.

FindFile Searches the directories specified by the global variable `search_path` for a specified file. The default `SolverTaskInit` proc uses this routine to locate the requested input MIF file.

`SolverTaskInit` and `SolverTaskCleanup` accept an arbitrary argument list (`args`), which is copied over from the `args` argument to the `BatchTaskRun` and `BatchTaskLaunch` procs in `batchsolve.tcl`. Typically one copies the default procs (as needed) into a task script, and makes appropriate modifications. You may (re-)define these procs either before or after sourcing `batchsolve.tcl`. See Sec. 10.2.2.4 for example scripts.

10.2.2 2D Micromagnetic Solver Batch Scheduling System

Overview

The OBS supports complex scheduling of multiple batch jobs with two applications, **batchmaster** and **batchslave**. The user launches **batchmaster** and provides it with a task script. The task script is a Tcl script that describes the set of tasks for **batchmaster** to accomplish. The work is actually done by instances of **batchslave** that are launched by **batchmaster**. The task script may be modeled after the included `simpletask.tcl` or `multitask.tcl` sample scripts.

The OBS has been designed to control multiple sequential and concurrent micromagnetic simulations, but **batchmaster** and **batchslave** are completely general and may be used to schedule other types of jobs as well.

10.2.2.1 Master Scheduling Control: batchmaster

The application **batchmaster** is launched by the command line:

```
tclsh oommf.tcl batchmaster [standard options] task_script \
      [host [port]]
```

task_script is the user defined task (job) definition Tcl script,

host specifies the network address for the master to use (default is *localhost*),

port is the port address for the master (default is *0*, which selects an arbitrary open port).

When **batchmaster** is run, it sources the task script. Tcl commands in the task script should modify the global object `$TaskInfo` to inform the master what tasks to perform and optionally how to launch slaves to perform those tasks. The easiest way to create a task

script is to modify one of the included example scripts. More detailed instructions are in Batch task scripts.

After sourcing the task script, **batchmaster** launches all the specified slaves, initializes each with a slave initialization script, and then feeds tasks sequentially from the task list to the slaves. When a slave completes a task it reports back to the master and is given the next unclaimed task. If there are no more tasks, the slave is shut down. When all the tasks are complete, the master prints a summary of the tasks and exits.

When the task script requests the launching and controlling of jobs off the local machine, with slaves running on remote machines, then the command line argument `host` **must** be set to the local machine's network name, and the `$TaskInfo` methods `AppendSlave` and `ModifyHostList` will need to be called from inside the task script. Furthermore, OOMMF does not currently supply any methods for launching jobs on remote machines, so a task script which requests the launching of jobs on remote machines requires a working `ssh` command or equivalent. (Details.)

10.2.2.2 Task Control: batchslave

The application **batchslave** may be launched by the command line:

```
tclsh oommf.tcl batchslave [standard options] \
   host port id password [auxscript [arg ...]]
```

host, port Host and port at which to contact the master to serve.

id, password ID and password to send to the master for identification.

auxscript arg ... The name of an optional script to source (which actually performs the task the slave is assigned), and any arguments it needs.

In normal operation, the user does not launch **batchslave**. Instead, instances of **batchslave** are launched by **batchmaster** as instructed by a task script. Although **batchmaster** may launch any slaves requested by its task script, by default it launches instances of **batchslave**.

The function of **batchslave** is to make a connection to a master program, source the `auxscript` and pass it the list of arguments `aux_arg ...`. Then it receives commands from the master, and evaluates them, making use of the facilities provided by `auxscript`. Each command is typically a long-running one, such as solving a complete micromagnetic problem. When each command is complete, the **batchslave** reports back to its master program, asking for the next command. When the master program has no more commands **batchslave** terminates.

Inside **batchmaster**, each instance of **batchslave** is launched by evaluating a Tcl command. This command is called the spawn command, and it may be redefined by the task script in order to completely control which slave applications are launched and how they are launched. When **batchslave** is to be launched, the spawn command might be:

```
exec tclsh oommf.tcl batchslave -tk 0 -- $server(host) $server(port) \
   $slaveid $passwd batchsolve.tcl -restart 1 &
```

The Tcl command `exec` is used to launch subprocesses. When the last argument to `exec` is `&`, the subprocess runs in the background. The rest of the spawn command should look familiar as the command line syntax for launching **batchslave**.

The example spawn command above cannot be completely provided by the task script, however, because parts of it are only known by **batchmaster**. Because of this, the task script should define the spawn command using "percent variables" which are substituted by **batchmaster**. Continuing the example, the task script provides the spawn command:

```
exec %tclsh %oommf batchslave -tk 0 %connect_info \
   batchsolve.tcl -restart 1
```

batchmaster replaces `%tclsh` with the path to `tclsh`, and `%oommf` with the path to the OOMMF bootstrap application. It also replaces `%connect_info` with the five arguments from `--` through `$password` that provide **batchslave** the hostname and port where **batchmaster** is waiting for it to report to, and the ID and password it should pass back. In this example, the task script instructs **batchslave** to source the file `batchsolve.tcl` and pass it the arguments `-restart 1`. Finally, **batchmaster** always appends the argument `&` to the spawn command so that all slave applications are launched in the background.

The communication protocol between **batchmaster** and **batchslave** is evolving and is not described here. Check the source code for the latest details.

10.2.2.3 Batch Task Scripts

The application **batchmaster** creates an instance of a `BatchTaskObj` object with the name `$TaskInfo`. The task script uses method calls to this object to set up tasks to be performed. The only required call is to the `AppendTask` method, e.g.,

```
$TaskInfo AppendTask A "BatchTaskRun taskA.mif"
```

This method expects two arguments, a label for the task (here "A") and a script to accomplish the task. The script will be passed across a network socket from **batchmaster** to a slave application, and then the script will be interpreted by the slave. In particular, keep in mind that the file system seen by the script will be that of the machine on which the slave process is running.

This example uses the default `batchsolve.tcl` procs to run the simulation defined by the `taskA.mif` MIF 1.x file. If you want to make changes to the MIF problem specifications on the fly, you will need to modify the default procs. This is done by creating a slave initialization script, via the call

```
$TaskInfo SetSlaveInitScript { <insert script here> }
```

The slave initialization script does global initializations, and also usually redefines the `SolverTaskInit` proc; optionally the `BatchTaskIterationCallback`, `BatchTaskRelaxCallback`

and SolverTaskCleanup procs may be redefined as well. At the start of each task SolverTaskInit is called by BatchTaskRun (in batchsolve.tcl), after each iteration BatchTaskIterationCallback is executed, at each control point BatchTaskRelaxCallback is run, and at the end of each task SolverTaskCleanup is called. SolverTaskInit and SolverTaskCleanup are passed the arguments that were passed to BatchTaskRun. A simple SolverTaskInit proc could be

```
proc SolverTaskInit { args } {
   global mif basename outtextfile
   set A [lindex $args 0]
   set outbasename "$basename-A$A"
   $mif SetA $A
   $mif SetOutBaseName $outbasename
   set outtextfile [open "$outbasename.odt" "a+"]
   puts $outtextfile [GetTextData header \
         "Run on $basename.mif, with A=[$mif GetA]"]
}
```

This proc receives the exchange constant A for this task on the argument list, and makes use of the global variables mif and basename. (Both should be initialized in the slave initialization script outside the SolverTaskInit proc.) It then stores the requested value of A in the mif object, sets up the base filename to use for output, and opens a text file to which tabular data will be appended. The handle to this text file is stored in the global outtextfile, which is closed by the default SolverTaskCleanup proc. A corresponding task script could be

```
$TaskInfo AppendTask "A=13e-12 J/m" "BatchTaskRun 13e-12"
```

which runs a simulation with A set to 13×10^{-12} J/m. This example is taken from the multitask.tcl sample script. (For commands accepted by mif objects, see the file mmsinit.cc. Another object than can be gainfully manipulated is solver, which is defined in solver.tcl.)

If you want to run more than one task at a time, then the $TaskInfo method AppendSlave will have to be invoked. This takes the form

```
$TaskInfo AppendSlave <spawn count> <spawn command>
```

where <spawn command> is the command to launch the slave process, and <spawn count> is the number of slaves to launch with this command. (Typically <spawn count> should not be larger than the number of processors on the target system.) The default value for this item (which gets overwritten with the first call to $TaskInfo AppendSlave) is

```
1 {Oc_Application Exec batchslave -tk 0 %connect_info batchsolve.tcl}
```

The Tcl command Oc_Application Exec is supplied by OOMMF and provides access to the same application-launching capability that is used by the OOMMF bootstrap application. Using a <spawn command> of Oc_Application Exec instead of exec %tclsh %oommf saves the spawning of an additional process. The default <spawn command> launches the

batchslave application, with connection information provided by **batchmaster**, and using the `auxscript batchsolve.tcl`.

Before evaluating the `<spawn command>`, **batchmaster** applies several percent-style substitutions useful in slave launch scripts: %tclsh, %oommf, %connect_info, %oommf_root, and %%. The first is the Tcl shell to use, the second is an absolute path to the OOMMF bootstrap program on the master machine, the third is connection information needed by the `batchslave` application, the fourth is the path to the OOMMF root directory on the master machine, and the last is interpreted as a single percent. **batchmaster** automatically appends the argument `&` to the `<spawn command>` so that the slave applications are launched in the background.

To launch **batchslave** on a remote host, use `ssh` in the spawn command, e.g.,

```
$TaskInfo AppendSlave 1 {exec ssh foo tclsh oommf/oommf.tcl \
      batchslave -tk 0 %connect_info batchsolve.tcl}
```

This example assumes `tclsh` is in the execution path on the remote machine `foo`, and OOMMF is installed off of your home directory. In addition, you will have to add the machine `foo` to the host connect list with

```
$TaskInfo ModifyHostList +foo
```

and `batchmaster` must be run with the network interface specified as the server host (instead of the default `localhost`), e.g.,

```
tclsh oommf.tcl batchmaster multitask.tcl bar
```

where `bar` is the name of the local machine.

This may seem a bit complicated, but the examples in the next section should make things clearer.

10.2.2.4 Sample task scripts

The first sample task script is a simple example that runs the 3 micromagnetic simulations described by the MIF 1.x files `taskA.mif`, `taskB.mif` and `taskC.mif`. It is launched with the command

```
tclsh oommf.tcl batchmaster simpletask.tcl
```

This example uses the default slave launch script, so a single slave is launched on the current machine, and the 3 simulations will be run sequentially. Also, no slave initialization script is given, so the default procs in `batchsolve.tcl` are used. Output will be magnetization states and tabular data at each control point, stored in files on the local machine with base names as specified in the MIF files.

The second sample task script builds on the previous example by defining `BatchTaskIterationCallbac` and `BatchTaskRelaxCallback` procedures in the slave init script. The first set up to write tabular data every 10 iterations, while the second writes tabular data on each control point

```
# FILE: simpletask.tcl
#
# This is a sample batch task file.  Usage example:
#
#   tclsh oommf.tcl batchmaster simpletask.tcl
#
# Form task list
$TaskInfo AppendTask A "BatchTaskRun taskA.mif"
$TaskInfo AppendTask B "BatchTaskRun taskB.mif"
$TaskInfo AppendTask C "BatchTaskRun taskC.mif"
```

Figure 10.1: Sample task script `simpletask.tcl`. (description)

event. The data is written to the output file specified by the `Base Output Filename` entry in the input MIF files. Note that there is no magnetization vector field output in this example. This task script is launched the same way as the previous example:

```
tclsh oommf.tcl batchmaster octrltask.tcl
```

The third task script is a more complicated example running concurrent processes on two machines. This script should be run with the command

```
tclsh oommf.tcl batchmaster multitask.tcl bar
```

where `bar` is the name of the local machine.

Near the top of the `multitask.tcl` script several Tcl variables (`RMT_MACHINE` through `A_list`) are defined; these are used farther down in the script. The remote machine is specified as `foo`, which is used in the `$TaskInfo AppendSlave` and `$TaskInfo ModifyHostList` commands.

There are two `AppendSlave` commands, one to run two slaves on the local machine, and one to run a single slave on the remote machine (`foo`). The latter changes to a specified working directory before launching the `batchslave` application on the remote machine. (For this to work you must have `ssh` configured properly.)

Below this the slave initialization script is defined. The Tcl `regsub` command is used to place the task script defined value of `BASEMIF` into the init script template. The init script is run on the slave when the slave is first brought up. It first reads the base MIF file into a newly created `mms_mif` instance. (The MIF file needs to be accessible by the slave process, irrespective of which machine it is running on.) Then replacement `SolverTaskInit` and `SolverTaskCleanup` procs are defined. The new `SolverTaskInit` interprets its first argument as a value for the exchange constant `A`. Note that this is different from the default `SolverTaskInit` proc, which interprets its first argument as the name of a MIF 1.x file to load. With this task script, a MIF file is read once when the slave is brought up, and then each task redefines only the value of `A` for the simulation (and corresponding changes to the output filenames and data table header).

Finally, the Tcl loop structure

```
# FILE: octrltask.tcl
#
# This is a sample batch task file, with expanded output control.
# Usage example:
#
#        tclsh oommf.tcl batchmaster octrltask.tcl
#
# "Every" output selection count
set SKIP_COUNT 10

# Initialize solver. This is run at global scope
set init_script {
    # Text output routine
    proc MyTextOutput {} {
        global outtextfile
        puts $outtextfile [GetTextData data]
        flush $outtextfile
    }
    # Change control point output
    proc BatchTaskRelaxCallback {} {
        MyTextOutput
    }
    # Add output on iteration events
    proc BatchTaskIterationCallback {} {
        global solver
        set count [$solver GetODEStepCount]
        if { ($count % __SKIP_COUNT__) == 0 } { MyTextOutput }
    }
}

# Substitute $SKIP_COUNT in for __SKIP_COUNT__ in above "init_script"
regsub -all -- __SKIP_COUNT__ $init_script $SKIP_COUNT init_script
$TaskInfo SetSlaveInitScript $init_script

# Form task list
$TaskInfo AppendTask A "BatchTaskRun taskA.mif"
$TaskInfo AppendTask B "BatchTaskRun taskB.mif"
$TaskInfo AppendTask C "BatchTaskRun taskC.mif"
```

Figure 10.2: Task script with iteration output `octrltask.tcl`. (description)

```
    foreach A $A_list {
        $TaskInfo AppendTask "A=$A" "BatchTaskRun $A"
    }
```

is used to build up a task list consisting of one task for each value of `A` in `A_list` (defined at the top of the task script). For example, the first value of `A` is 10e-13, so the first task will have the label `A=10e-13` and the corresponding script is `BatchTaskRun 10e-13`. The value 10e-13 is passed on by `BatchTaskRun` to the `SolverTaskInit` proc, which has been redefined to process this argument as the value for `A`, as described above.

There are 6 tasks in all, and 3 slave processes, so the first three tasks will run concurrently in the 3 slaves. As each slave finishes it will be given the next task, until all the tasks are complete.

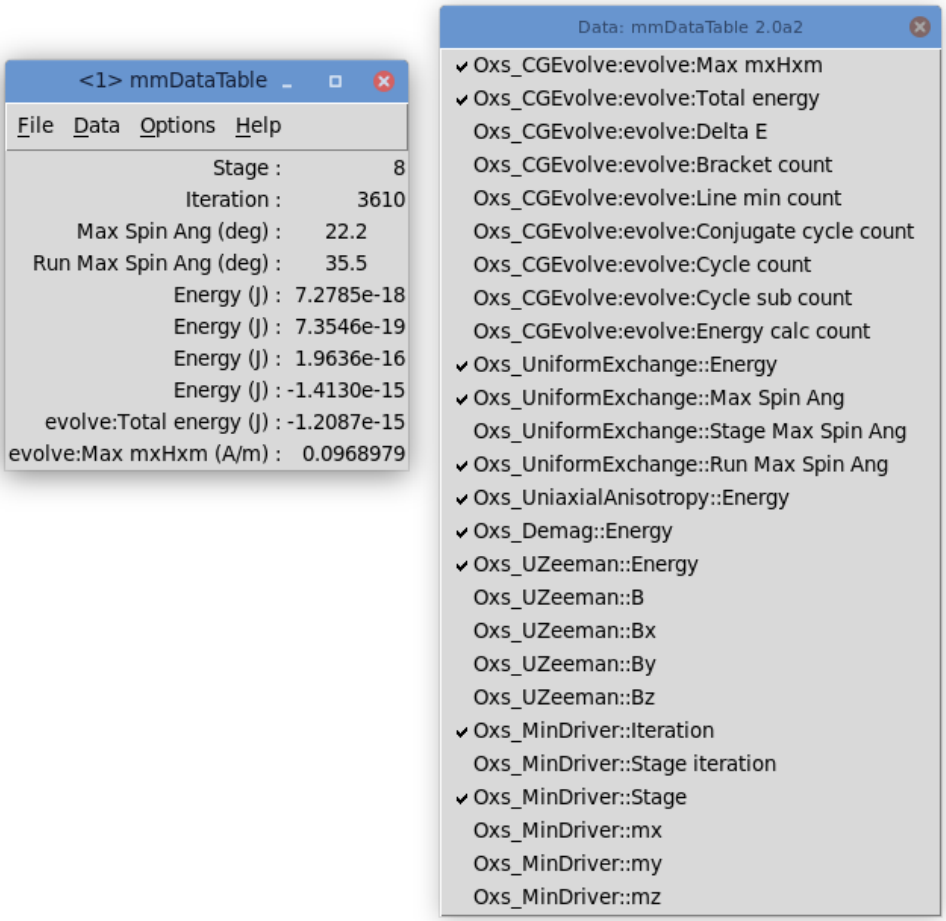
```
# FILE: multitask.tcl
#
# This is a sample batch task file.  Usage example:
#
#   tclsh oommf.tcl batchmaster multitask.tcl hostname [port]
#
# Task script configuration
set RMT_MACHINE   foo
set RMT_TCLSH      tclsh
set RMT_OOMMF      "/path/to/oommf/oommf.tcl"
set RMT_WORK_DIR   "/path/to/oommf/app/mmsolve/data"
set BASEMIF taskA
set A_list { 10e-13 10e-14 10e-15 10e-16 10e-17 10e-18 }

# Slave launch commands
$TaskInfo ModifyHostList +$RMT_MACHINE
$TaskInfo AppendSlave 2 "exec %tclsh %oommf batchslave -tk 0 \
        %connect_info batchsolve.tcl"
$TaskInfo AppendSlave 1 "exec ssh $RMT_MACHINE \
        cd $RMT_WORK_DIR \\\;\
        $RMT_TCLSH $RMT_OOMMF batchslave -tk 0 %connect_info batchsolve.tcl"

# Slave initialization script (with batchsolve.tcl proc
# redefinitions)
set init_script {
    # Initialize solver. This is run at global scope
    set basename __BASEMIF__      ;# Base mif filename (global)
    mms_mif New mif
    $mif Read [FindFile ${basename}.mif]
    # Redefine TaskInit and TaskCleanup proc's
```

```
    proc SolverTaskInit { args } {
        global mif outtextfile basename
        set A [lindex $args 0]
        set outbasename "$basename-A$A"
        $mif SetA $A
        $mif SetOutBaseName $outbasename
        set outtextfile [open "$outbasename.odt" "a+"]
        puts $outtextfile [GetTextData header \
                "Run on $basename.mif, with A=[$mif GetA]"]
        flush $outtextfile
    }
    proc SolverTaskCleanup { args } {
        global outtextfile
        close $outtextfile
    }
}
# Substitute $BASEMIF in for __BASEMIF__ in above script
regsub -all -- __BASEMIF__ $init_script $BASEMIF init_script
$TaskInfo SetSlaveInitScript $init_script

# Create task list
foreach A $A_list {
    $TaskInfo AppendTask "A=$A" "BatchTaskRun $A"
}
```

Figure 10.3: Advanced sample task script `multitask.tcl`. (description)

Chapter 11

Data Table Display: mmDataTable



Overview

The application **mmDataTable** provides a data display service to its client applications. It accepts data from clients which are displayed in a tabular format in a top-level window. Its typical use is to display the evolving values of quantities computed by micromagnetic solver programs.

Launching

mmDataTable may be started either by selecting the **mmDataTable** button on **mmLaunch**, or from the command line via

```
tclsh oommf.tcl mmDataTable [standard options] [-net <0|1>]
```

-net <0|1> Disable or enable a server which allows the data displayed by **mmDataTable** to be updated by another application. By default, the server is enabled. When the server is disabled, **mmProbEd** is only useful if it is embedded in another application.

Inputs

The client application(s) that send data to **mmDataTable** for display control the flow of data. The user, interacting with the **mmDataTable** window, controls how the data is displayed. Upon launch, **mmDataTable** displays only a menubar. Upon user request, a display window below the menubar displays data values.

Each message from a client contains a list of (name, value, units) triples containing data for display. For example, one element in the list might be {`Magnetization 800000 A/m`}. **mmDataTable** stores the latest value it receives for each name. Earlier values are discarded when new data arrives from a client.

Outputs

mmDataTable does not support any data output or storage facilities. To save tabular data, use the **mmGraph** or **mmArchive** applications.

Controls

The **Data** menu holds a list of all the data names for which **mmDataTable** has received data. Initially, **mmDataTable** has received no data from any clients, so this menu is empty. As data arrives from clients, the menu fills with the list of data names. Each data name on the list lies next to a checkbutton. When the checkbutton is toggled from off to on, the corresponding data name and its value and units are displayed at the bottom of the display

window. When the checkbutton is toggled from on to off, the corresponding data name is removed from the display window. In this way, the user selects from all the data received what is to be displayed. Selecting the dashed rule at the top of the **Data** menu detaches it so the user may easily click multiple checkbuttons.

Displayed data values can be individually selected (or deselected) with a left mouse button click on the display entry. Highlighting is used to indicated which data values are currently selected. The **Options** menu also contains commands to select or deselect all displayed values. The selected values can be copied into the cut-and-paste (clipboard) buffer with the `Ctrl+c` key combination, or the **Options|Copy** menu command.

The data value selection mechanism is also used for data value formatting control. The **Options|Format** menu command brings up a **Format** dialog box to change the justification and format specification string. The latter is the conversion string passed to the Tcl `format` command, which uses the C `printf` format codes. If the **Adjust:Selected** radiobutton is active, then the specification will be applied to only the currently selected (highlighted) data values. Alternately, if **Adjust:All** is active, then the specification will be applied to all data values, and will additionally become the default specification.

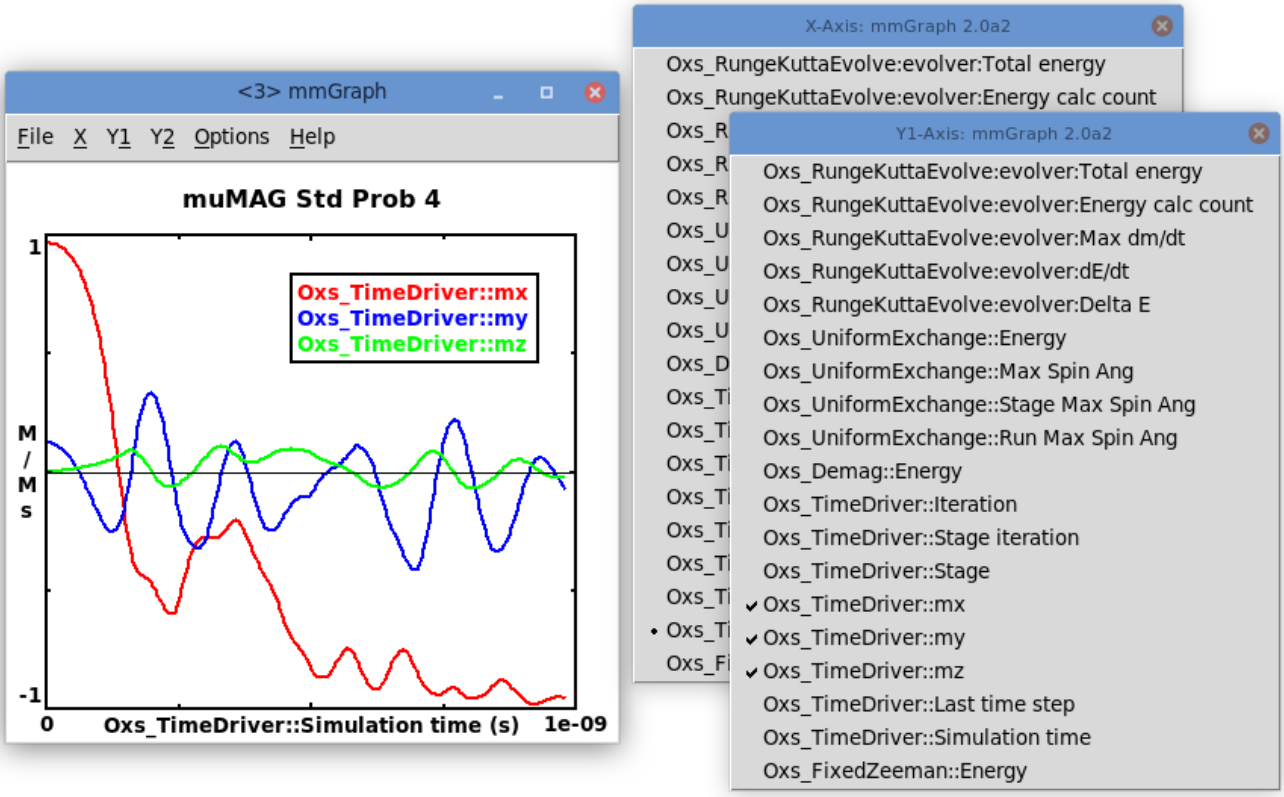
A right mouse button click on a display entry will select that entry, and bring up the **Format** dialog box with the justification and format specifications of the selected entry. These specifications, with any revisions, may then be applied to all of the selected entries.

If a value cannot be displayed with the selected format specification string, e.g., if a "%d" integer format were applied to a string containing a decimal point, then the value will be printed in red in the form as received by **mmDataTable**, without any additional formatting.

The menu selection **File|Reset** reinitializes the **mmDataTable** application to its original state, clearing the display and the **Data** menu. The reset operation is also automatically invoked upon receipt of new data following a data set close message from a solver application. The menu selection **File|Exit** terminates the application. The menu **Help** provides the usual help facilities.

Chapter 12

Data Graph Display: mmGraph



Overview

The application **mmGraph** provides a data display service similar to that of **mmDataTable**. The usual data source is a running solver, but rather than the textual output provided by **mmDataTable**, **mmGraph** produces 2D line plots. **mmGraph** also stores the data

it receives, so it can produce multiple views of the data and can save the data to disk. Postscript output is also supported.

Launching

mmGraph may be started either by selecting the **mmGraph** button on **mmLaunch** or from the command line via

```
tclsh oommf.tcl mmGraph [standard options] [-net <0|1>] [loadfile ...]
```

-net <0|1> Disable or enable a server which allows the data displayed by **mmGraph** to be updated by another application. By default, the server is enabled. When the server is disabled, **mmGraph** may only input data from a file.

loadfile ... Optional list of data (ODT) files to preload.

Inputs

Input to **mmGraph** may come from either a file in the ODT format, or when `-net 1` (the default) is active, from a client application (typically a running solver). The **File|Open...** dialog box is used to select an input file. Receipt of data from client applications is the same as for **mmDataTable**. In either case, input data are appended to any previously held data.

When reading from a file, **mmGraph** will automatically decompress data using the local customization "Nb_InputFilter decompress" option to `Oc_Option`. For details, see the discussion on file translation in the Inputs section of the **mmDisp** documentation.

Curve breaks (i.e., separation of a curve into disjoint segments) are recorded in the data storage buffer via *curve break records.* These records are generated whenever a new data table is detected by **mmGraph**, or when requested by the user using the **mmGraph Options|Break Curves** menu option.

Outputs

Unlike **mmDataTable**, **mmGraph** internally stores the data sent to it. These data may be written to disk via the **File|Save As...** dialog box. If the file specified already exists, then **mmGraph** output is appended to that file. The output is in the tabular ODT format. The data are segmented into separate `Table Start/Table End` blocks across each curve break record.

By default, all data currently held by **mmGraph** is written, but the **Save: Selected Data** option presented in the **File|Save As...** dialog box causes the output to be restricted to those curves currently selected for display. In either case, the graph display limits do *not* affect the output.

The save operation writes records that are held by **mmGraph** at the time the **File|Save As...** dialog box **OK** button is invoked. Additionally, the **Auto Save** option in this dialog box may be used to automatically append to the specified file each new data record as it is received by **mmGraph**. The appended fields will be those chosen at the time of the save operation, i.e., subsequent changing of the curves selected for display does not affect the automatic save operation. The automatic save operation continues until either a new output file is specified, the **Options|Stop Autosave** control is invoked, or **mmGraph** is terminated.

The **File|Print...** dialog is used to produce a Postscript file of the current graph. On Unix systems, the output may be sent directly to a printer by filling the **Print to:** entry with the appropriate pipe command, e.g., `|lpr`. (The exact form is system dependent.)

Controls

Graphs are constructed by selecting any one item off the **X**-axis menu, and any number of items off the **Y1**-axis and **Y2**-axis menus. The y1-axis is marked on the left side of the graph; the y2-axis on the right. These menus may be detached by selecting the dashed rule at the top of the list. Sample results are shown in the figure at the start of this section.

When **mmGraph** is first launched, all the axis menus are empty. They are dynamically built based on the data received by **mmGraph**. By default, the graph limits and labels are automatically set based on the data. The x-axis label is set using the selected item data label and measurement unit (if any). The y-axes labels are the measurement unit of the first corresponding y-axis item selected.

The **Options|Configure...** dialog box allows the user to override default settings. To change the graph title, simply enter the desired title into the **Title** field. To set the axis labels, deselect the **Auto Label** option in this dialog box, and fill in the **X Label**, **Y1 Label** and **Y2 Label** fields as desired. The axis limits can be set in a similar fashion. In addition, if an axis limit is left empty, a default value (based on all selected data) will be used. Select the **Auto Scale** option to have the axis ranges automatically adjust to track incoming data.

Use the **Auto Offset Y1** and **Auto Offset Y2** to automatically translate each curve plotted against the specified axis up or down so that the first point on the curve has a y-value of zero for a linear axis or one for a logarithmic axis. This feature is especially useful for comparing variations between different energy curves, because for these curves one is typically interested in changes is values rather than the absolute energy value itself.

By default the scaling on each axis is linear. The **Log scale axes** check boxes enables logarithmic scaling on the selected axes. If logarithmic scaling is selected then any point with a zero value is dropped and negative values are replaced with their corresponding positive absolute value.

The size of the margin surrounding the plot region is computed automatically. Larger margins may be specified by filling in the appropriate fields in the **Margin Requests** section. Units are pixels. Requested values smaller than the computed (default) values are ignored.

The **Canvas color** selection sets the color of the canvas background. The curve colors and symbol types are automatically assigned, either by curve or segment, as specified by the **Color selection** control. (Data collected across multiple solver runs will consist of one segment for each run.) The **Display** control can be used to enable or disable display of curves and/or symbols.

The initial curve width is determined by the `Ow_GraphWin default_curve_width` setting in the `config/options.tcl` and `config/local/options.tcl` files, following the usual method of local customization. The current curve width can be changed by specifying the desired width in the **Curve Width** entry in the **Options|Configure...** dialog box. The units are pixels. Long curves will be rendered more quickly, especially on Windows, if the curve width is set to 1. The symbol frequency and size (in pixels) may also be controlled. A value of 0 for **Symbol Freq** disables symbol display. The default values for these options can be controlled from the configuration `options.tcl` files.

As mentioned earlier, **mmGraph** stores in memory all data it receives. Over the course of a long run, the amount of data stored can grow to many megabytes. This storage can be limited by specifying a positive (> 0) value for the **Point buffer size** entry in the **Options|Configure...** dialog box. The oldest records are removed as necessary to keep the total number of records stored under the specified limit. A zero value for **Point buffer size** is interpreted as no limit. (The storage size of an individual record depends upon several factors, including the number of items in the record and the version of Tcl being used.) Data erasures may not be immediately reflected in the graph display.

At any time, the point buffer storage may be completely emptied with **Options|Clear Data**, or thinned via **Options|Thin Data**. The **Options|Stop Autosave** selection will turn off the auto save feature, if currently active. The **Options|Break Curves** item inserts a curve break record into the point buffer, causing a break in each curve after the current point. This option may be useful if **mmGraph** is being fed data from multiple sources.

Also on this menu is **Options|Rescale**, which autoscales the graph axis limits from the selected data. This command ignores but does not reset the **Auto Scale** setting in the **Options|Configure...** dialog box. The Rescale command may also be invoked by pressing the `Home` key.

The **Options|Key** selection toggles the key (legend) display on and off. The key may also be repositioned by dragging with the left mouse button. If curves are selected off both the y1 and y2 menus, then a horizontal line in the key separates the two sets of curves, with the labels for the y1 curves on top.

If the **Options|Auto Reset** selection is enabled, then when a new table is detected all previously existing axis menu labels that are not present in the column list of the new data set are deleted, along with their associated data. **mmGraph** will detect a new table when results from a new problem are received, or when data is input from a file. If **Options|Auto Reset** is not selected, then no data or axis menu labels are deleted, and the axes menus will show the union of the old column label list and the new column label list. If the axes menus grow too long, the user may manually apply the **File|Reset** command to clear them.

The last command on the options menu is **Options|Smooth**. If smoothing is disabled,

then the data points are connected by straight line segments. If enabled, then each curve is rendered as a set of parabolic splines, which do not in general pass through the data points. This is implemented using the `-smooth 1` option to the Tcl `canvas create line` command; see that documentation for details.

A few controls are available only using the mouse. If the mouse pointer is positioned over a drawn item in the graph, holding down the `Ctrl` key and the left mouse button will bring up the coordinates of that point, with respect to the y1-axis. Similarly, depressing the `Ctrl` key and the right mouse button, or alternatively holding down the `Ctrl`+`Shift` keys while pressing the left mouse button will bring up the coordinates of the point with respect to the y2-axis. The coordinates displayed are the coordinates of a point on a drawn line, which are not necessarily the coordinates of a plotted data point. (The data points are plotted at the endpoints of each line segment.) The coordinate display is cleared when the mouse button is released while the `Ctrl` key is down.

One vertical and one horizontal rule (line) are also available. Initially, these rules are tucked and hidden against the left and bottom graph axes, respectively. Either may be repositioned by dragging with the left or right mouse button. The coordinates of the cursor are displayed while dragging the rules. The displayed y-coordinate corresponds to the y1-axis if the left mouse button is used, or the y2-axis if the right mouse button or the `Shift` key with the left mouse button are engaged.

The graph extents may be changed by selecting a "zoom box" with the mouse. This is useful for examining a small portion of the graph in more detail. This feature is activated by clicking and dragging the left or right mouse button. A rectangle will be displayed that changes size as the mouse is dragged. If the left mouse button is depressed, then the x-axis and y1-axis are rescaled to just match the extents of the displayed rectangle. If the right mouse button, or alternatively the shift key + left mouse button, is used, then the x-axis and y2-axis are rescaled. An arrow is drawn against the rectangle indicating which y-axis will be rescaled. The rescaling may be canceled by positioning the mouse pointer over the initial point before releasing the mouse button. The zoom box feature is similar to the mouse zoom control in the **mmDisp** application, except that here there is no "un-zooming" mouse control. However, the coordinate limits for each zoom are stored in a list, and the `Esc` and `Shift+Esc` keys may be used to move backwards and forwards, respectively, through this list. The `Enter`/`Return` key will copy the current coordinate limits to the list. A pointer is kept to the selected display configuration state, and new states are added after that point; if a display state is added in the interior of the list (i.e. ahead of the last state), then all configurations following the new entry are deleted. The entire list is cleared any time automatic scaling is invoked.

The `Page Up` and `Page Down` keys may also be used to zoom the display in and out. Use in conjuction with the `Shift` key to jump by larger steps, or with the `Ctrl` key for finer control. The **Options|Rescale** command or the **Options|Configure...** dialog box may also be used to reset the graph extents.

If **mmGraph** is being used to display data from a running solver, and if **Auto Scale** is selected in the **Options|Configure...** dialog box, then the graph extents may be changed

automatically when a new data point is received. This is inconvenient if one is simultaneously using the zoom feature to examine some portion of the graph. In this case, one might prefer to disable the **Auto Scale** feature, and manually pan the display using the keyboard arrow keys. Each key press will translate the display one half frame in the indicated direction. The `Shift` key used in combination with an arrow keys double the pan step size, while the `Ctrl` key halves it.

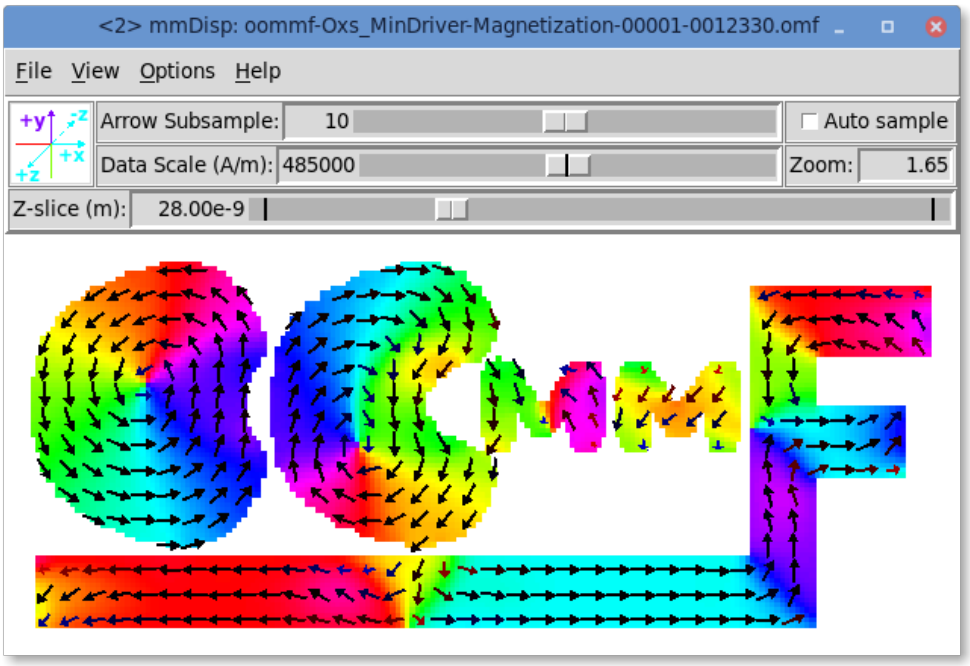
The menu selection **File|Reset** reinitializes the **mmGraph** application to its original state, releasing all data and clearing the axis menus. The menu selection **File|Exit** terminates the application. The menu **Help** provides the usual help facilities.

Details

The axes menus are configured based on incoming data. As a result, these menus are initially empty. If a graph widget is scheduled to receive data only upon control point or stage done events in the solver, it may be a long time after starting a problem in the solver before the graph widget can be configured. Because **mmGraph** keeps all data up to the limit imposed by the `Point buffer size`, data loss is usually not a problem. Of more importance is the fact that automatic data saving can not be set up until the first data point is received. As a workaround, the solver initial state may be sent interactively as a dummy point to initialize the graph widget axes menus. Select the desired quantities off the axes menus, and use the **Options|clear Data** command to remove the dummy point from **mmGraph**'s memory. The **File|Save As...** dialog box may then be used—with the **Auto Save** option enabled—to write out an empty table with proper column header information. Subsequent data will be written to this file as they arrive.

Chapter 13

Vector Field Display: mmDisp



Overview

The application **mmDisp** displays two-dimensional slices of three-dimensional spatial distributions of vector fields. **mmDisp** currently supports display of 1D (i.e., scalar) and 3D vector data. It can load field data from files in a variety of formats, or it can accept data from client applications, such as a running solver. **mmDisp** offers a rich interface for controlling the display of vector field data, and can also save the data to a file or produce PostScript print output.

Launching

mmDisp may be started either by selecting the **mmDisp** button on **mmLaunch**, or from the command line via

```
tclsh oommf.tcl mmDisp [standard options] [-config file] \
   [-net <0|1>] [filename]
```

-config file User configuration file that specifies default display parameters. This file is discussed in detail below.

-net <0|1> Disable or enable a server which allows the data displayed by **mmDisp** to be updated by another application. By default, the server is enabled. When the server is disabled, **mmDisp** may only input data from a file.

If a filename is supplied on the command line, **mmDisp** takes it to be the name of a file containing vector field data for display. That file will be opened on startup.

Inputs

Input to **mmDisp** may come from either a file or from a client application (typically a running solver), in any of the OOMMF vector field formats. Other file formats can also be supported if a translation filter program is available.

Client applications that send data to **mmDisp** control the flow of data. The user, interacting with the **mmDisp** window, determines how the vector field data are displayed.

File input is initiated through the **File|Open...** dialog box. Several example files are included in the OOMMF release in the directory `app/mmdisp/examples`. When the **Browse** button is enabled, the "Open File" dialog box will remain open after loading a file, so that multiple files may be displayed in sequence. The **Auto** configuration box determines whether the vector subsampling, data scale, zoom and slice settings should be determined automatically (based on the data in the file and the current display window size), or whether their values should be held constant while loading the file.

mmDisp permits local customization allowing for automatic translation from other file formats into one of the OOMMF vector field formats that **mmDisp** recognizes. When loading a file, **mmDisp** compares the file name to a list of extensions. An example extension is `.gz`. The assumption is that the extension identifies files containing data in a particular format. For each extension in the list, there is a corresponding translation program. **mmDisp** calls on that program as a filter which takes data in one format from standard input and writes to standard output the same data in one of the formats supported by **mmDisp**. In its default configuration, **mmDisp** recognizes the patterns `.gz`, `.z`, and `.zip`, and invokes the translation program `gzip -dc` to perform the "translation." In this way, support for reading compressed files is "built in" to **mmDisp** on any platform where the **gzip** program is installed.

There are two categories of translations supported: decompression and format conversion. Both are modified by the usual method of local customization. The command governing decompression in the customization file is of the form

```
Oc_Option Add * Nb_InputFilter decompress {{.gz .zip} {gzip -dc}}
```

The final argument in this command is a list with an even number of elements. The first element of each pair is the filename extension. The second element in each pair is the command line for launching the corresponding translation program. To add support for bzip2 compressed files, change this line to

```
Oc_Option Add * Nb_InputFilter decompress \
                {{.gz .zip} {gzip -dc} .bz2 bunzip2}
```

This option also affects other applications such as **mmGraph** that support "on-the-fly" decompression. In all cases the decompression program must accept compressed input on standard input and write the decompressed output to standard output.

There is also input translation support for filters that convert from foreign (i.e., non-OOMMF) file formats. For example, if a program `foo` were known to translate a file format identified by the extension `.bar` into the OVF file format, that program could be made known to **mmDisp** by setting the customization command:

```
Oc_Option Add * Nb_InputFilter ovf {.bar foo}
```

This assumes that the program `foo` accepts input of the form `.bar` on standard input and writes the translated results to standard output.

The **File|Load config...** dialog allows one to load a previously saved display configuration file from disk to reset the current display. See the Configuration Files section below for details.

Outputs

The vector field displayed by **mmDisp** may be saved to disk via the **File|Save As...** dialog box. The output is in the OVF format. The OVF file options may be set by selecting the appropriate radio buttons in the OVF File Options panel. The **Title** and **Desc** fields may be edited before saving. Enabling the **Browse** button allows for saving multiple files without closing the "Save File" dialog box.

The **File|Print...** dialog is used to produce a PostScript file of the current display. On Unix systems, the output may be sent directly to a printer by filling the **Print to:** entry with the appropriate pipe command, e.g., `|lpr`. (The exact form is system dependent.) The other print dialog box options are described in the configuration files section below.

The **File|Write config...** dialog allows one to save to disk a configuration file holding the current display and print settings. Files of this type can be used to control display parameters, at start time via the `-config` option or during run time through the **File|Load**

config... menu option. These files can also be used as input to the avf2ppm and avf2ps command line utilities that convert files from the OVF format into bitmap images, PostScript, or Scalable Vector Graphics. (mmDisp does not provide internal support for direct writing of bitmap or .svg files.) Details of the configuration file are discussed below.

Controls

The menu selection **File|Clear** clears the display window. The menu selection **File|Exit** terminates the **mmDisp** application. The menu **Help** provides the usual help facilities.

The **View** menu provides high-level control over how the vector field is placed in the display window. The menu selection **View|Wrap Display** resizes the display window so that it just contains the entire vector field surrounded by a margin. **View|Fill Display** resizes the vector field until it fills the current size of the display window. If the aspect ratio of the display window does not match the aspect ratio of the vector field, a larger than requested margin appears along one edge to make up the difference. **View|Center Display** translates the vector field to put the center of view at the center of the display window. **View|Rotate ccw** and **View|Rotate cw** rotate the display one quarter turn counter-clockwise and clockwise respectively. If the display size is not locked (see **Options|Lock size** below), then the display window also rotates, so that the portion of the vector field seen and any margins are preserved (unless the display of the control bar forces the display window to be wider). **View|reDraw** allows the user to invoke a redrawing of the display window. The **View|Viewpoint** tearable submenu supports rotation of the vector field out of the plane of the display, so that it may be viewed from along a different axis.

The menu selection **Options|Configure...** brings up a dialog box through which the user may control many features of the vector field display. Vectors in the vector field may be displayed as arrows, pixels, or both. The **Arrow** and **Pixel** buttons in the **Plot Type** column on the left of the dialog box enable each type of display.

Columns 2–4 in the Configure dialog box control the use of color. Both arrows and pixels may be independently colored to indicate some quantity. The **Color Quantity** column controls which scalar quantity the color of the arrow or pixel represents. Available color quantities include vector x, y, and z components, total vector magnitude, slice depth, and angles as measured in-plane about a fixed axis. On regularly gridded data the vector field divergence and curl are also available for display.

The assignment of a color to a quantity value is determined by the **Colormap** selected. Colormaps are labeled by a sequence of colors that are mapped across the range of the selected quantity. For example, if the "Red-Black-Blue" colormap is applied to the **Color Quantity** "z", then vectors pointing into the xy-plane ($z < 0$) are colored red, those lying in the plane ($z = 0$) are colored black, and those pointing out of the plane ($z > 0$) are colored blue. Values between the extremes are displayed with intermediate colors, selected using a discretization determined by the **# of Colors** value. This value governs the use of potentially limited color resources, and can be used to achieve special coloring effects. (Note: The in-plane angle quantities are generally best viewed with a colormap that begins

and ends with the same color, e.g., "Red-Green-Blue-Red.") The **Offset** field and **Reverse** checkbox translate and reverse the colormap, as described in the `arrow,coloroffset` and `arrow,colorreverse` items of the **Configuration Files** section below (page 160).

Below the **Reverse** checkbutton in the pixel plot type row is an **Opaque** checkbutton. If this is selected then arrows underneath the pixel slice (see slice discussion, page 156) will be fully hidden by the pixel element. When the opaque checkbutton is disabled the pixel element is drawn in a translucent manner, so if the arrow display span contains multiple slices then arrows below the pixel element will be partially visible.

When there are many vectors in a vector field, a display of all of them may be more confusing than helpful. The **Subsample** column allows the user to request that only a sampling of vectors from the vector field be displayed. The **Subsample** value is roughly the number of vectors along one spatial dimension of the vector field which map to a single displayed vector (arrow or pixel). Each vector displayed is an actual vector in the vector field—the selection of vectors for display is a sampling process, not an averaging or interpolation process. The subsample rates for arrows and pixels may be set independently. A subsample rate of 0 is interpreted specially to display all data. (This is typically quicker than subsampling at a small rate, e.g., 0.1.)

The length of an arrow represents the magnitude of the vector field. All arrows are drawn with a length between zero and "full-scale." By default, the full-scale arrow length is computed so that it covers the region of the screen that one displayed vector is intended to represent, given the current subsample rate. Following this default, arrows do not significantly overlap each other, yet all non-zero portions of the vector field have a representation in the display. Similarly, pixels are drawn with a default size that fills an area equal to the region of the screen one pixel is intended to represent, given the pixel subsample rate. The **Size** column allows the user to (independently) override the default size of pixels and full-scale arrows. A value of 1 represents the default size. By changing to a larger or smaller **Size** value, the user may request arrows or pixels larger or smaller than the default size.

Below the arrow **Size** box is the **View scale** option. If this is enabled (the default) then arrow scaling is adjusted so that a size setting of 1 results in an in-viewplane vector having length approximately equal to the smaller of the two in-plane view-cell dimensions. (The view-cell is the discretization cell multiplied by the subsample setting.) If **View scale** is disabled then the arrow size is scaled relative to the smallest of all three view-cell dimensions, and is therefore fixed independent of view axis. Disabling both auto subsampling and view scale may make comparisons between different view axis directions easier.

Below the Arrow and Pixel Controls are several additional controls. The **Data Scale** entry affects the data value scaling. As described above, all arrows are displayed with length between zero and full-scale. The full-scale arrow length corresponds to some scalar value of the magnitude of the vector field. The **Data Scale** entry allows the user to set the value at which the drawn arrow length goes full-scale. Any vectors in the vector field with magnitude equal to or greater than the data scale value will be represented by arrows drawn at full scale. Other vectors will be represented by shorter arrows with length determined by a linear scale between zero and the data scale value. Similarly, the data scale value controls the range

of values spanned by the colormap used to color pixels. The usual use of the **Data Scale** entry is to reduce the data scale value so that more detail can be seen in those portions of the vector field which have magnitude less than other parts of the vector field. If the data scale value is increased, then the length of the arrows in the plot is reduced accordingly. If the data scale value is decreased, then the length of the arrows is increased, until they reach full-scale. An arrow representing a vector with magnitude larger than the data scale value may be thought of as being truncated to the data scale value. The initial (default) data scale value is usually the maximum vector magnitude in the field, so at this setting no arrows are truncated. Entering `0` into the data scale box will cause the data scale to be reset to the default value. (For OVF 1.0 files, the default data scale value is set from the `ValueRangeMaxMag` header line. This is typically set to the maximum vector magnitude, but this is not guaranteed.) The data scale control is intended primarily for use with vector fields of varying magnitude (e.g., **H**-fields), but may also be used to adjust the pixel display contrast for any field type.

The **Zoom** entry controls the spatial scaling of the display. The value roughly corresponds to the number of pixels per vector in the fully-sampled vector field. (This value is not affected by the subsampling rate.)

The **Margin** entry specifies the margin size, in pixels, to be maintained around the vector field.

The next row of entry boxes control slice display. Slice selection allows display of that subset of the data that is within a specified distance of a plane running perpendicular to the view axis. The location of that plane with respect to the view axis is specified in the **X-slice center**, **Y-slice center** or **Z-slice center** entry, depending on the current view axis. The thickness of the slice may be varied separately for arrow and pixel displays, as specified in the next two entry boxes. The slice span boxes interpret specially the following values: 0 resets the slice thickness to the default value, which is usually the thickness of a single cell. Any negative value sets the slice thickness to be the full thickness of the mesh. Values for all of the slice control entries are specified in the fundamental mesh spatial unit, for example, meters. (Refer to the vector field file format documentation for more on mesh spatial units.)

Below the slice contols are controls to specify whether or not a bounding polygon is displayed, and the background color for the display window.

No changes made by the user in the **Options|Configure...** dialog box affect the display window until either the **Apply** or **OK** button is selected. If the **OK** button is selected, the dialog box is also dismissed. The **Close** button dismisses the dialog without changing the display window.

The following item on the **Options** menu is **Options|Copy configuration to clipboard**, which copies the current display and print configuration onto the system clipboard. This is plain text data as described in the configuration files section. It can be copied into an editor for modification before saving to disk, or one can use the **Options|Paste configuration from clipboard** control to copy directly from the clipboard into another instance of **mmDisp**, which changes the configuration of the second **mmDisp** to match the first. Use the keyboard shortcuts `Ctrl+Shift+C` and `Ctrl+Shift+V`, respectively, to quickly invoke these operations.

The next item under the **Options** menu is a checkbutton that toggles the display of a control bar. The control bar offers alternative interfaces to some of the operations available from the **Options**|**Configure...** dialog box and the **View** menu. On the left end of the control bar is a display of the coordinate axes. These axes rotate along with the vector field in the display window to identify the coordinate system of the display, and are color coded to agree with the pixel (if active) or arrow coloring. A click of the left mouse button on the coordinate axes causes a counter-clockwise rotation. A click of the right mouse button on the coordinate axes causes a clockwise rotation.

To the right of the coordinate axes are two rows of controls. The top row allows the user to control the subsample rate and size of displayed arrows. The subsample rate may be modified either by direct entry of a new rate, or by manipulation of the slider. The second row controls the current data scale value and zoom (spatial magnification). A vertical bar in the slider area marks the default data scale value. Specifying `0` for the data scale value will reset the data scale to the default value.

Similar to **mmGraph**, the mouse can be used to extract field values and positions from the display window. Depressing the `Shift` key and the left mouse button will open a pop-up showing the field value at the mouse pointer position. Holding down the `Ctrl`+`Shift` keys while pressing the left mouse button will show both the position and value. The pop-up disappears when the mouse button is released, unless the `Shift` key is released before the mouse.

The spatial magnification may be changed either by typing a value in the Zoom box of the control bar, or by using the mouse inside the display window. A click and drag with the left mouse button displays a red rectangle that changes size as the mouse is dragged. When the left mouse button is released, the vector field is rescaled so that the portion of the display window within the red rectangle expands until it reaches the edges of the display window. Both dimensions are scaled by the same amount so there is no aspect distortion of the vector field. Small red arrows on the sides of the red rectangle indicate which dimension will expand to meet the display window boundaries upon release of the left mouse button. After the rescaling, the red rectangle remains in the display window briefly, surrounding the same region of the vector field, but at the new scale.

A click and drag with the right mouse button displays a blue rectangle that changes size as the mouse is dragged. When the right mouse button is released, the vector field is rescaled so that all of the vector field currently visible in the display window fits the size of the blue rectangle. Both dimensions are scaled by the same amount so there is no aspect distortion of the vector field. Small blue arrows on the sides of the blue rectangle indicate the dimension in which the vector field will shrink to exactly transform the display window size to the blue rectangle size. After the rescaling, the blue rectangle remains in the display window briefly, surrounding the same region of the vector field, now centered in the display window, and at the new scale.

When the zoom value is large enough that a portion of the vector field lies outside the display window, scrollbars appear that may be used to translate the vector field so that different portions are visible in the display window. On systems that have a middle mouse

button, clicking the middle button on a point in the display window translates the vector field so that the selected point is centered within the display window.

mmDisp remembers the previous zoom value and data scale values. To revert to the previous settings, the user may hit the `Esc` key. This is a limited "Undo" feature.

Below the data scale and zoom controls in the control bar is the slice center selection control. This will be labeled **Z-slice**, **X-slice**, or **Y-slice**, depending on which view axis is selected. The thickness of the slice can be set from the **Options|Configure...** dialog box.

The final item under the **Options** menu is the **Options|Lock size** checkbutton. By default, when the display is rotated in-plane, the width and height of the viewport are interchanged so that the same portion of the vector field remains displayed. Selecting the **Options|Lock size** checkbutton disables this behavior, and also other viewport changing operations (e.g., display wrap).

Several keyboard shortcuts are available as alternatives to menu- or mouse-based operations. (These are in addition to the usual keyboard access to the menu.) The effect of a key combination depends on which subwindow of **mmDisp** is active. The `Tab` key may be used to change the active subwindow. The `Shift+Tab` key combination also changes the active subwindow, in reverse order.

When the active subwindow is the display window, the following key combinations are active:

- `Ctrl+o` – same as menu selection **File|Open...**
- `Ctrl+s` – same as menu selection **File|Save as...**
- `Ctrl+p` – same as menu selection **File|Print...**
- `Ctrl+c` – same as menu selection **Options|Configure...**
- `Ctrl+v` – launches viewpoint selection menu, **View|Viewpoint**
- `Ctrl+w` – same as menu selection **View|Wrap Display**
- `Ctrl+f` – same as menu selection **View|Fill Display**
- `Home` – First fill, then wrap the display.
- `Ctrl+space` – same as menu selection **View|Center Display**
- `Ctrl+r` – same as menu selection **View|Rotate ccw**
- `Ctrl+Shift+r` – same as menu selection **View|Rotate cw**
- `Insert` – decrease arrow subsample by 1
- `Del` – increase arrow subsample by 1
- `Shift+Insert` – decrease arrow subsample by factor of 2

- `Shift+Del` – increase arrow subsample by factor of 2
- `Page Up` – increase the zoom value by a factor of 1.149
- `Page Down` – decrease the zoom value by a factor of 1.149
- `Shift+Page Up` – increase the zoom value by factor of 2
- `Shift+Page Down` – decrease the zoom value by factor of 2
- `Esc` – revert to previous data scale and zoom values

When the active subwindow is the control bar's coordinate axes display, the following key combinations are active:

- `Left arrow` – same as menu selection **View|Rotate ccw**
- `Right arrow` – same as menu selection **View|Rotate cw**

When the active subwindow is any of the control bar's value entry windows – arrow subsample, size, data scale or zoom, the following key combinations are active:

- `Esc` – undo uncommitted value (displayed in red)
- `Return` – commit entered value

When the active subwindow is in any of the control bar's sliders—arrow subsample, data scale or slice—the following key combinations are active:

- `Left arrow` – slide left (decrease value)
- `Right arrow` – slide right (increase value)
- `Left mouse button` – small step towards cursor
- `Shift + Left mouse button` – large step towards cursor
- `Esc` – undo uncommitted value (displayed in red)
- `Return` – commit current value

When any of the separate dialog windows are displayed (e.g., the **File|Open...** or **Options|Configure...** dialogs), the shortcut `Ctrl+.` (control-period) will raise and transfer keyboard focus back to the root **mmDisp** window.

Configuration Files

The various initial display parameters (e.g., window size, orientation, colormap) are set by configuration files. The default configuration file

```
oommf/app/mmdisp/scripts/mmdisp.config
```

is read first, followed by the local customization file,

```
oommf/app/mmdisp/scripts/local/mmdisp.config
```

if it exists. Lastly, any files passed as `-config` options on the command line are input. The files must be valid Tcl scripts, the main purpose of which is to set elements of the `plot_config` and `print_config` arrays, as illustrated in the default configuration file. (See the Tcl documentation for details of the `array set` command.)

There are several places in the configuration file where colors are specified. Colors may be represented using the symbolic names in `oommf/config/colors.config`, in any of the Tk hexadecimal formats, e.g., `#RRGGBB`, or as a shade of gray using the format "grayD" (or "greyD"), where D is a decimal integer from 0-100, inclusive. Examples in the latter two formats are `#FFFF00` for yellow, `gray0` for black, and `gray100` or `#FFFFFF` for white.

Refer to the default configuration file as we discuss each element of the `plot_config` array:

arrow,status Set to 1 to display arrows, 0 to not draw arrows.

arrow,autosample If 1, then ignore the value of `arrow,subsample` and automatically determine a "reasonable" subsampling rate for the arrows. Set to 0 to turn off this feature.

arrow,subsample If `arrow,autosample` is 0, then subsample the input vectors at this rate when drawing arrows. A value of 0 for `arrow,subsample` is interpreted specially to display all data.

arrow,colormap Select the colormap to use when drawing arrows. Should be one of the strings specified in the `Colormap` section of the **Options|Configure...** dialog.

arrow,colorcount Number of discretization levels to use from the colormap. A value of zero will color all arrows with the first color in the colormap.

arrow,quantity Scalar quantity the arrow color is to represent. Supported values include `x`, `y`, `z`, `xy-angle`, `xz-angle`, `yz-angle`, and `slice`. The **Options|Configure...** dialog presents the complete list of allowed quantities, which may be image dependent.

arrow,coloroffset This and the analogous `pixel,coloroffset` options provide a translation in the selected `colormap`. For the `xy-angle`, `xz-angle` and `yz-angle` color quantities, this displays as a rotation of the colormap, e.g., setting coloroffset to 0.333 effectively changes the `Red-Green-Blue-Red` colormap into `Green-Blue-Red-Green`. For

the other color quantities it shifts the display band, saturating at one end. For example, setting coloroffset to 0.5 changes the `Blue-White-Red` colormap into `White-Red-Red`. For non-angle color quantities coloroffset works jointly with `misc,datascale` to determine color assignment. If `coloroffset` is zero then the colormap extremes correspond to $\pm$`misc,datascale`. To shift the colors to a specific range, say to have the pixel color display range from a to b with $b > a$, then use

$$\begin{aligned} \texttt{misc,datascale} &= (b-a)/2 \\ \texttt{pixel,coloroffset} &= -(b+a)/\left[(b-a)/2\right] \end{aligned}$$

arrow,colorreverse The `colorreverse` value should be 1 or 0, signifying to reverse or not reverse, respectively. If reverse is selected, then the colormap ordering is inverted, changing for example `Blue-White-Red` into `Red-White-Blue`. This corresponds to the `Reverse` control in **Options|Configure...**. If both color reverse and offset adjustment are requested, then color reverse is applied first.

arrow,size Size of the arrows relative to the default size (represented as 1.0).

arrow,viewscale Enables automatic scaling of arrows determined by the in-viewplane cell dimensions.

pixel,... Most of the pixel configuration elements have analogous arrow configuration elements, and are interpreted in the same manner. The exception is the `pixel,opaque` element, which is discussed next. Note too that the auto subsampling rate for pixels is considerably denser than for arrows.

pixel,opaque This setting should be a real number between 0 and 1. If opaque is 1, then the pixel is drawn in a solid manner, concealing any arrows which are drawn under it. Otherwise, the pixel is drawn in a translucent manner so objects beneath it can still be discerned, with smaller values increasing the transparency. In the **mmDisp** Configure dialog box, selecting or deselecting the **Opaque** option toggles the `pixel,opaque` configuration value between 1 and 0.25, respectively.

misc,background Specify the background color.

misc,drawboundary If 1, then draw the bounding polygon, if any, as specified in the input vector field file.

misc,boundarycolor String specifying the bounding polygon color, if drawn.

misc,boundarywidth Width of the bounding polygon, in pixels.

misc,margin The size of the border margin, in pixels.

misc,defaultwindowwidth, misc,defaultwindowheight Width and height of initial display viewport, in pixels.

misc,width, misc,height Width and height of displayed area. This will be less than the viewport dimensions if scrollbars are present. These values are ignored during **mmDisp** initialization, but are written out by the **File|Write config...** command as a convenience for the avf2ppm command line utility.

misc,rotation Counterclockwise rotation in degrees; either 0, 90, 180 or 270.

misc,zoom Scaling factor for the display. This is the same value as shown in the "zoom" box in the **mmDisp** control bar, and corresponds roughly to the number of pixels per vector in the (original, fully-sampled) vector field. If set to zero, then the scaling is set so the image, including margins, just fits inside the viewport dimensions.

misc,datascale Scale for arrow size and colormap ranges; equivalent to the `Data Scale` control. In general, this should be a positive real value, but a zero or empty value will set the scaling to its default setting.

misc,centerpt If specified, the value should be a three item list of real numbers specifying the center of the display, `{x y z}`, in file mesh units (e.g., meters).

misc,relcenterpt If specified, the value should be a three item list of real numbers in the range $[0, 1]$ specifying the center of the display in relative coordinates. If both `misc,relcenterpt` and `misc,centerpt` are defined, then `misc,centerpt` takes precedence.

viewaxis Select the view axis, which should be one of `+z`, `-z`, `+y`, `-y`, `+x`, or `-x`. This option is equivalent to the **View|Viewpoint** menu control.

viewaxis,xarrowspan, viewaxis,yarrowspan, viewaxis,zarrowspan Specifies the thickness of the arrow display slice, for the corresponding view. For example, if the view axis is `+z` or `-z`, then only **viewaxis,zarrowspan** is active. The value for each element should be either a real value or an empty string. If the value is zero or an empty string, then the thickness is set to the default value, which is typically the thickness of a single cell. If the value is positive, then it specifies the slice range in file mesh units, e.g., in meters. Lastly, if the value is negative, then the slice is set to the entire thickness of the mesh in that view direction.

viewaxis,xpixelspan, viewaxis,ypixelspan, viewaxis,zpixelspan Identical interpretation and behavior as the corresponding arrow span elements, but with regards to pixel display.

The `print_config` array controls printing defaults, as displayed in the **File|Print...** dialog box:

orient Paper orientation, either landscape or portrait.

paper Paper type: letter, legal, A4 or A3.

hpos, vpos The horizontal and vertical positioning on the printed page. Valid values for `hpos` are left, center, or right, and for `vpos` are top, center, or bottom.

units Units that the margin and print area dimensions are measured in; either in or cm.

tmargin, lmargin Top and left margin size, measured in the selected units.

pwidth, pheight Output print area dimensions, width and height, measured in the selected units. The output will be scaled to meet the more restrictive dimension. In particular, the x/y-scaling ratio remains 1:1.

croptoview Boolean value, either 0 or 1. If set to the default value 1 (true), then the output is cropped to display only that portion of the vector field that is visible in the display window. If 0, then the display is ignored and the entire vector field is printed, scaled to fit in the dimensions specified by `pwidth` and `pheight`. In this case the `plot_config(misc,zoom)` is used to control arrow and pixel autosampling behavior (if enabled), but is otherwise ignored.

If any of the above elements are set in multiple configuration files, then the last value read takes precedence.

Details

The selection of vectors for display determined by the **Subsample** value differs depending on whether or not the data lie on a regular grid. If so, **Subsample** takes integer values and determines the ratio of data points to displayed points. For example, a value of 5 means that every fifth vector on the grid is displayed. This means that the number of vectors displayed is 25 times fewer than the number of vectors on the grid.

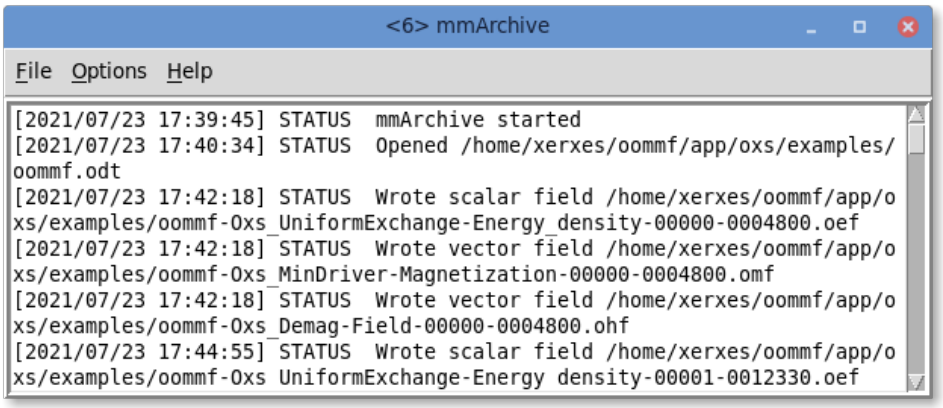
For an irregular grid of vectors, an average cell size is computed, and the **Subsample** takes values in units of 0.1 times the average cell size. A square grid of that size is overlaid on the irregular grid. For each cell in the square grid, the data vector from the irregular grid closest to the center of the square grid cell is selected for display. The vector is displayed at its true location in the irregular grid, not at the center of the square grid cell. As the subsample rate changes, the set of displayed vectors also changes, which can in some circumstances substantially change the appearance of the displayed vector field.

```
array set plot_config {
  arrow,status       1               misc,background           white
  arrow,autosample   1               misc,drawboundary         1
  arrow,subsample    0               misc,boundarycolor        black
  arrow,colormap     Red-Black-Blue  misc,boundarywidth        1
  arrow,colorcount   256             misc,margin               10
  arrow,quantity     z               misc,defaultwindowwidth   640
  arrow,colorreverse 0               misc,defaultwindowheight  480
  arrow,coloroffset  0               misc,width                0
  arrow,size         1               misc,height               0
  arrow,viewscale    1               misc,rotation             0
                                     misc,zoom                 0
  pixel,status       0               misc,datascale            0
  pixel,autosample   1               misc,relcenterpt     {0.5 0.5 0.5}
  pixel,subsample    0
  pixel,colormap     Blue-White-Red  viewaxis                  +z
  pixel,colorcount   256             viewaxis,xarrowspan       {}
  pixel,quantity     x               viewaxis,xpixelspan       {}
  pixel,colorreverse 0               viewaxis,yarrowspan       {}
  pixel,coloroffset  0               viewaxis,ypixelspan       {}
  pixel,size         1               viewaxis,zarrowspan       {}
  pixel,opaque       1               viewaxis,zpixelspan       {}
}
array set print_config {
    orient   landscape               tmargin   1.0
    paper    letter                  lmargin   1.0
    hpos     center                  pwidth    6.0
    vpos     center                  pheight   6.0
    units    in                      croptoview 1
}
```

Figure 13.1: Contents of default configuration file `mmdisp.config`. (description)

Chapter 14

Data Archive: mmArchive



Overview

The application **mmArchive** provides automated vector field and data table storage services. Although **mmDisp** and **mmGraph** are able to save such data under the direction of the user, there are situations where it is more convenient to write data to disk without interactive control.

mmArchive does not present a user interface window of its own, but like the **Oxs solvers** relies on **mmLaunch** to provide an interface on its behalf. Because **mmArchive** does not require a window, it is possible on Unix systems to bring down the X (window) server and still keep **mmArchive** running in the background.

Launching

mmArchive may be started either by selecting the **mmArchive** button on **mmLaunch** by **Oxsii/Boxsi** via a `Destination` command in a MIF 2 file, or from the command line via

```
tclsh oommf.tcl mmArchive [standard options] [-detach]
```

-detach Causes **mmArchive** to close its stdio channels during startup. This may allow a parent process to exit without waiting on **mmArchive**.

When the **mmArchive** button of **mmLaunch** is invoked, **mmArchive** is launched with the `-tk 0` option. This allows **mmArchive** to continue running if the X window server is killed. The `-tk 1` option is useful only for enabling the `-console` option for debugging.

As noted above, **mmArchive** depends upon **mmLaunch** to provide an interface. The entry for an instance of **mmArchive** in the **Running Applications** column of any running copy of **mmLaunch** has a checkbutton next to it. This button toggles the presence of a user interface window through which the user may control that instance of **mmArchive**.

Inputs

mmArchive accepts vector field and data table style input from client applications (typically running solvers) on its network (socket) interface.

Outputs

The client applications that send data to **mmArchive** control the flow of data. **mmArchive** copies the data it receives into files specified by the client. There is no interactive control to select the names of these output files. A simple status line shows the most recent vector file save, or data table file open/close event.

For data table output, if the output file already exists then the new data is appended to the end of the file. The data records for each session are sandwiched between "Table Start" and "Table End" records. See the ODT format documentation for explanation of the data table file structure. It is the responsibility of the user to ensure that multiple data streams are not directed to the same data table file at the same time.

For vector field output, if the output file already exists then the old data is deleted and replaced with the current data. See the OVF documentation for information about the vector field output format.

Controls

The display area inside the **mmArchive** window displays a log of **mmArchive** activity. The menu selection **File|Close interface** closes the **mmArchive** window without terminating **mmArchive**. Use the **File|Exit mmArchive** option or the window close button to terminate **mmArchive**.

If the **Options|Wrap lines** option is selected, then each log entry is line wrapped. Otherwise, each entry is on one line, and a horizontal slider is provided at the bottom of the display window to scroll through line. The **Options|Clear buffer** command clears the log display area. This clears the buffer in that **mmArchive** display window only. If a new

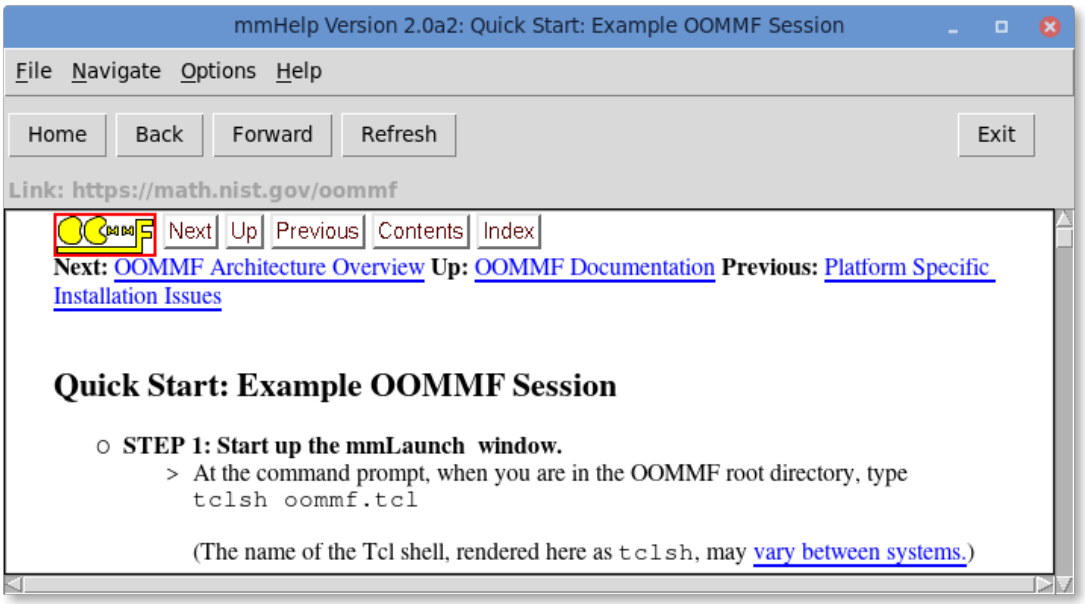
display window is opened for the same **mmArchive** instance, the new display will show the entire log backing store. The last two items on the **Options** menu, **Enlarge font** and **Reduce font**, adjust the size of the font used in the log display area.

Known Bugs

mmArchive appends data table output to the file specified by the source client application (e.g., a running solver). If, at the same time, more than one source specifies the same file, or if the same source sends data table output to more than one instance of **mmArchive**, then concurrent writes to the same file may corrupt the data in that file. It is the responsibility of the user to ensure this does not happen; there is at present no file locking mechanism in OOMMF to protect against this situation.

Chapter 15

Documentation Viewer: mmHelp



Overview

The application **mmHelp** manages the display and navigation of hypertext (HTML) help files. **mmHelp** displays only a simplified form of hypertext required to display the OOMMF help pages. It is not able to display many of the advanced features provided by modern World Wide Web browsers. OOMMF software can be customized to use another program to display the HTML help files.

Launching

mmHelp may be launched from the command line via

```
tclsh oommf.tcl mmHelp [standard options] [URL]
```

The command line argument `URL` is the URL of the first page (home page) to be displayed. If no URL is specified, **mmHelp** displays the Table of Contents of the *OOMMF User's Guide* by default.

Controls

Each page of hypertext is displayed in the main **mmHelp** window. Words which are underlined and colored blue are hyperlinks which **mmHelp** knows how to follow. Words which are underlined and colored red are hyperlinks which **mmHelp** does not know how to follow. Moving the mouse over a hyperlink displays the target URL of the hyperlink in the **Link:** line above the display window. Clicking on a blue hyperlink will follow the hyperlink and display a new page of hypertext.

mmHelp keeps a list of the viewed pages in order of view. Using the **Back** and **Forward** buttons, the user may move backward and forward through this list of pages. The **Home** button causes the first page to be displayed, allowing the user to start again from the beginning. These three buttons have corresponding entries in the **Navigate** menu.

Use the menu selection **File|Open** to directly select a file from the file system to be displayed by **mmHelp**.

The menu selection **File|Refresh**, or the **Refresh** button causes **mmHelp** to reload and redisplay the current page. This may be useful if the display becomes corrupted, or for repeatedly loading a hypertext file which is being edited.

When **mmHelp** encounters hypertext elements it does not recognize, it will attempt to work around the problem. However, in some cases it will not be able to make sense of the hypertext, and will display an error message. Documentation authors should take care to use only the hypertext elements supported by **mmHelp** in their documentation files. Users should never see such an error message.

mmHelp displays error messages in one of two ways: within the display window, or in a separate window. Errors reported in the display window replace the display of the page of hypertext. They usually indicate that the hypertext page could not be retrieved, or that its contents are not hypertext. File permission errors are also reported in this way.

Errors reported in a separate window are usually due to a formatting error within the page of hypertext. Selecting the **Continue** button of the error window instructs **mmHelp** to attempt to resume display of the hypertext page beyond the error. Selecting **Abort** abandons further display.

The menu selection **Options|Font scale...** brings up a dialog box through which the user may select the scale of the fonts to use in the display window, relative to their initial size.

The menu selection **File|Exit** or the **Exit** button terminates the **mmHelp** application. The menu **Help** provides the usual help facilities.

Known Bugs

mmHelp is pretty slow. You may be happier using local customization methods to replace it with another more powerful HTML browser. Also, we have noticed that the underscore character in the italic font is not displayed (or is displayed as a space) at some font sizes on some platforms.

Chapter 16

Command Line Utilities

This section documents a few utilities distributed with OOMMF that are run from the command line (Unix shell or Windows DOS prompt). They are typically used in pre- or post-processing of data associated with a micromagnetic simulation.

16.1 Bitmap File Format Conversion: any2ppm

The **any2ppm** program converts bitmap files from the Portable Pixmap (PPM), Windows BMP, and GIF formats into the Portable Pixmap P3 (text) or P6 (binary) formats, or the uncompressed 24 bits-per-pixel BMP binary format. With Tcl/Tk 8.6 or later the *Portable Network Graphics* (PNG) format is also supported. Additional formats may be available if the Tcl/Tk *Img*[1] package is installed on your system. (Note: OOMMF support for BMP requires Tk 8.0 or later.)

Launching

The **any2ppm** launch command is:

```
tclsh oommf.tcl any2ppm [standard options] [-f] [-format fmt] \
   [-noinfo] [-o outfile] [infile ...]
```

where

-f Force output. If the `-o` option is not specified, then the output filename is automatically generated by stripping the extension, if any, off of each input filename, and appending a format-specific extension (e.g., `.ppm`). If `-f` is specified, that generated filename is used for the output filename. If `-f` is not specified, then a check is made to see if the generated filename already exists. If so, then an additional "-000" or "-001" ... suffix is appended to create an unused filename. If the input is coming from stdin, i.e., there is no input filename, then the default output is to stdout.

[1] https://wiki.tcl-lang.org/page/Img

-format fmt Output file format. The default is `PPM` or `P3` which is the Portable Pixmap P3 (text) format; use `P6` to get the binary PPM P6 output. Setting `fmt` to `BMP` will produce files in the uncompressed Windows BMP 24 bits-per-pixel format. Under Tcl/Tk 8.6 and later the *Portable Network Graphics* format can be selected by setting `fmt` to `PNG`. If the Tcl/Tk Img package is installed, then additional formats, such as PNG (for pre-Tcl/Tk 8.6), JPEG and TIFF, will be available. The default output file extension depends on the format selected, e.g., `.ppm` for PPM files and `.bmp` for BMP files.

-noinfo Suppress writing of progress information to stderr.

-o outfile Write output to `outfile`; use "-" to pipe output to stdout. Note that if `outfile` is specified, then all output will go to this one file; in this case it is unlikely that one wants to specify more than one input file.

infile ... List of input files to process. If none, or if an infile is the empty string, then read from stdin.

Note: If the output is to stdout, and the selected output format is anything other than `PPM`, then the output is first written to a temporary file before being copied to stdout. Under normal operation the temporary file will be automatically deleted, but this is not guaranteed if the program terminates abnormally.

Tk Requirement: any2ppm uses the Tk `image` command in its processing. This requires that Tk be properly initialized, which in particular means that a valid display must be available. This is not a problem on Windows, where a desktop is always present, but on Unix this means that an X server must be running. The *Xvfb*[2] virtual framebuffer can be used if desired. (Xvfb is an X server distributed with X11R6 that requires no display hardware or physical input devices.)

16.2 Making Data Tables from Vector Fields: avf2odt

The **avf2odt** program converts *rectangularly* meshed vector field files in any of the recognized formats into the ODT 1.0 data table format. (Irregular meshes are not supported by this command. Note that any OVF file using the "irregular" meshtype is considered to be using an irregular mesh, even if the mesh nodes do in fact lie on a rectangular grid.)

Launching

The **avf2odt** launch command is:

```
tclsh oommf.tcl avf2odt [standard options] \
   [-average <space|plane|line|point|ball>] [-axis <x|y|z>] \
```

[2] https://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml

```
[-ball_radius brad] [-defaultpos <0|1>] [-defaultvals <0|1>] \
[-extravals flag] [-filesort method] [-headers <full|collapse|none>] \
[-index label units valexpr] [-ipat pattern] [-normalize <0|1>] \
[-numfmt fmt] [-onefile outfile] [-opatexp regexp] [-opatsub sub] \
[-region xmin ymin zmin xmax ymax zmax] \
[-rregion rxmin rymin rzmin rxmax rymax rzmax] [-truncate <0|1>] \
[-v level] [-valfunc label units fcnexpr] \
[-wgtfunc fcnexpr exclude_zeros] [infile ...]
```

where

-average <space|plane|line|point|ball> Specify type of averaging. Selection of `Space` averaging results in the output of one data line (per input file) consisting of the average v_x, v_y and v_z field values in the selected region (see `-region` option below). For example, in magnetization files, v_x, v_y and v_z correspond to M_x, M_y and M_z. If `plane` or `line` is selected, then the output data table consists of multiple lines with 4 or 5 columns per line, respectively. The last 3 columns in both cases are the v_x, v_y and v_z averaged over the specified axes-parallel affine subspace (i.e., plane or line). In the `plane` case, the first column specifies the averaging plane offset along the coordinate axis normal to the plane (see `-axis` option below). In the `line` case, the first 2 columns specify the offset of the averaging line in the coordinate plane perpendicular to the line. If the averaging type is `point`, then no averaging is done, and the output consists of lines of 6 column data, one line for each point in the selected region, where the first 3 columns are the point coordinates, and the last 3 are the v_x, v_y and v_z values at the point. If the type is `ball`, then one line is output for each sample point for which a ball of radius brad (see `-ball_radius` option) centered about that point lies entirely inside the selected region. The output values consist of 6 columns: the ball center point location and the v_x, v_y and v_z values averaged across the ball. As a special case, if the spatial extent of the selected region is two-dimensional (e.g., all the sample locations have the same z-coordinate), then the averaging region is taken to be a disk instead of a ball. Similarly, if the spatial extent of the selected region is one-dimensional, then the averaging region is reduced to a one-dimensional line segment. (Note: The output columns described above may be suppressed by the `-defaultpos` and `-defaultvals` options. Additional columns may be introduced by the `-index` and `-valfunc` options.) The default averaging type is `space`.

-axis <x|y|z> Required for the `-average plane` and `-average line` averaging types, where it selects which subset of affine subspaces the averaging will be performed over. In the `plane` case, the `-axis` represents the normal direction to the planes, while for `line` it is the direction parallel to the lines. This parameter is ignored if `-average` is neither `plane` nor `line`.

-ball_radius brad This option is required if `-average` is `ball`, in which case brad specifies the radius of the averaging ball in problem units (e.g., meters). If `-average` is not

`ball`, then this option is ignored.

-defaultpos <0|1> By default, the output data columns are as described in the description of the `-average` option above. However, `-defaultpos 0` may be used to omit the columns indicating the averaging position.

-defaultvals <0|1> By default, the output data columns are as described in the description of the `-average` option above. However, `-defaultvals 0` may be used to omit the columns containing the averaged v_x, v_y and v_z values. In particular, this may be useful in conjunction with the `-valfunc` option.

-extravals <0|1> Specify `-extravals 1` to augment the output with columns for the average L^1 norm $\sum(|v_x| + |v_y| + |v_z|)/N$, the normalized L^2 norm $\sqrt{\sum v^2/N}$, the minimum component absolute value, and the maximum component absolute value.

-filesort method Specifies the sorting order to apply to the input file list. This order is important when using the `-onefile` option, since it controls the order in which the rows from the various input files are concatenated. Method should be either the keyword "none", or else a valid option string for the Tcl command `lsort`, e.g., "-ascii -decreasing". Note that the `lsort` sort options all begin with a hyphen, "-", and that if you want to use multiple options they must be grouped as one element to `filesort` (by, for example, placing quotes around the list). The default value is "-dictionary" if the `-ipat` option is specified, or "none" otherwise.

-headers <full|collapse|none> Determines the style of headers written to the output ODT file(s). The full style (default) provides the standard headers, as described in the ODT documentation. Specifying "none" produces raw data lines without any headers. The collapse style is used with multiple input files and the `-onefile` output option to concatenate output with no ODT header information between the segments.

-index label units valexpr Adds an input file based index column to the output, where label is the column header, units is a string displayed as the column units header, and valexpr is a Tcl `expr` expression that may include the special variables `$i`, `$f1`, `$f2`, ..., `$d1`, `$d2`, ...; here `$i` is the 0-based index of the file in the list of input files, `$f1` is the first number appearing in the input filename, `$f2` is the second number appearing in the input filename, `$d1` is the first number appearing in the "Desc" fields in the header of the input file, etc. For example, if there are two input files named `foo-100.ovf` and `foo-101.ovf`, then setting valexpr to `abs($f1)+1` would yield a column with the value 101 for all lines coming from `foo-100.ovf`, and the value 102 for all lines coming from `foo-101.ovf`. (We use the Tcl `expr` function `abs` because the leading hyphen in `foo-100.ovf` gets interpreted as a minus sign, so `$f1` is extracted as -100.) On Unix systems, the valexpr string should be surrounding by single quotes in order to forestall interpolation of the special variables by the shell. On Windows, the valexpr string should be surrounded by double quotes as usual to protect embedded

spaces. Multiple instances of the `-index` option on the command line will result in multiple columns in the output file, in the order specified. The index columns, if any, will be the first columns in the output file.

-ipat pattern Specify input files using a pattern with "glob-style" wildcards. Especially useful in DOS. Files must meet the `infile` requirements (see below).

-normalize <0|1> If 1, then the default averaged output values v_x, v_y and v_z are divided by the maximum magnitude that would occur if all the vectors in the averaging manifold are aligned. (In particular, the maximum magnitude of the output vector is 1.) This option should be used carefully because the normalization is done independently for each output row. For `-normalize 0` (the default), averaged output values are in file units.

-numfmt fmt C-style output format for numeric data in the body of the output table. Default value is "`%- #20.15g`".

-onefile outfile Generally a **avf2odt** writes its output to a collection of files with names generated using the `-opatexp` and `-opatsub` specifications. This option overrides that behavior and sends all output to one place. If outfile is "-", then the output is sent to standard output, otherwise outfile is the name of the output file.

-opatexp regexp Specify the "regular expression" applied to input filenames to determine portion to be replaced in generation of output filenames. The default regular expression is: `(\.[^.]?[^.]?[^.]?$|$)`

-opatsub sub The string with which to replace the portion of input filenames matched by the `-opatexp regexp` during output filename generation. The default is `.odt`.

-region xmin ymin zmin xmax ymax zmax Axes-parallel rectangular box denoting region in the vector field file over which data is to be collected. The locations are in problem units (typically meters). A single hyphen, "-", may be specified for any of the box corner coordinates, in which case the corresponding extremal value from the input file is used. Optional; the default, `-region - - - - - -`, selects the entire input file.

-rregion rxmin rymin rzmin rxmax rymax rzmax This option is the same as `-region`, except that the locations are specified in relative units, between 0 and 1.

-truncate <0|1> When opening an existing file for output, the new output can either be appended to the file (`-truncate 0`), or else the existing data can be discarded (`-truncate 1`). The default is `-truncate 0`.

-v level Verbosity (informational message) level, with 0 generating only error messages, and larger numbers generating additional information. The `level` value is an integer, defaulting to 1.

-valfunc label units fcnexpr Similar to the `-index` option, `-valfunc` adds an additional column to the output with label and units as the column header, and fcnexpr is a Tcl `expr` expression that may include special variables. Here, however, the allowed special variables are `$x`, `$y`, `$z`, `$r`, `$vx`, `$vy`, `$vz`, `$vmag`, where `$x`, `$y`, `$z`, and `$r` are sample location and magnitude, respectively ($r = \sqrt{x^2 + y^2 + z^2}$), and `$vx`, `$vy`, `$vz` and `$vmag` are vector component values and magnitude. The output is the value of fcnexpr averaged across the manifold selected by the `-average` option. A couple of examples are

```
-valfunc Ms   A/m '$vmag'
-valfunc M110 A/m '($vx+$vy)/sqrt(2.)'
```

As with the valexpr string for `-index`, the fcnexpr string should be surrounding by single quotes on Unix in order to forestall interpolation of the special variables by the shell. On Windows, the fcnexpr string should be surrounded by double quotes as usual to protect embedded spaces. The output value is not affected by the `-normalize` option. Multiple instances of the `-valfunc` option on the command line will result in multiple columns in the output file, in the order specified. These additional columns will be append to the right of all other columns in the output file.

-wgtfunc fcnexpr exclude_zeros The `-wgtfunc` option provides a pointwise weighting to the file field values. The fcnexpr is a Tcl `expr` expression of the same form as used in the `valfunc` option, and supports the same special variables. At each spatial location the expression is evaluated (the weight) and is multiplied against the field value there. If exclude_zeros is non-zero then points with zero weight are excluded from the average. Multiple instances of this option are stacked; in this case a point is excluded from the average if any of the instances with exclude_zeros set true have weight = 0. For the `plane`, `line`, and `ball` manifold types, if exclude_zeros>1 then an additional column is included in the output reporting the number of points in the averaging manifold.

One use of the `-wgtfunc` option with exclude_zeros > 0 is to average across irregular subsets of the full volume. (As compared to the `-region` and `-rregion` options which create only axis-parallel rectangular subregions.) For example, to take averages across a full-height cylindrical volume of radius 15 nm centered about $(x, y) = (400\times10^{-9}, 200\times10^{-9})$, use

```
-wgtfunc 'pow($x-400e-9,2)+pow($y-200e-9,2)<pow(15e-9,2) ? 1 : 0' 1
```

(On Windows replace the single quotes with double quotes.) As another example, given a magnetization file (`.omf`) for a magnetic part with an irregular boundary, find the magnetization averaged across the magnetically active portion (i.e., the part with $\mathbf{M} \neq 0$) with

```
-wgtfunc '$vmag == 0 ? 0 : 1' 1
```

(If exclude_zeros here were 0 instead of 1, then the average would be across the full volume, yielding a smaller result because of the averaged-in zeros.)

infile ... Input file list. Files must be one of the recognized formats, OVF 1.0 or VIO, in a rectangular mesh subformat.

The file specification options require some explanation. Input files may be specified either by an explicit list (`infile ...`), or by giving a wildcard pattern, e.g., `-ipat *.omf`, which is expanded in the usual way by **avf2odt** (using the Tcl command `glob`). Unix shells (sh, csh, etc.) automatically expand wildcards before handing control over to the invoked application, so the `-ipat` option is not usually needed—although it is useful in case of a "command-line too long" error. DOS does not do this expansion, so you must use `-ipat` to get wildcard expansion in Windows. The resulting file list is sorted based on the `-filesort` specification as described above.

If `-onefile` is not requested, then as each input file is processed, a name for the corresponding output file is produced from the input filename by rules determined by handing the `-opatexp` and `-opatsub` expressions to the Tcl `regsub` command. Refer to the Tcl `regsub` documentation for details, but essentially whatever portion of the input filename is matched by the `-opatexp` expression is removed and replaced by the `-opatsub` string. The default `-opatexp` expression matches against any filename extension of up to 3 characters, and the default `-opatsub` string replaces this with the extension `.odt`.

16.3 Vector Field File Format Conversion: avf2ovf

The **avf2ovf** program converts vector field files from any of the recognized formats into the OOMMF OVF or the Python NumPy NPY format.

Launching

The **avf2ovf** launch command is:

```
tclsh oommf.tcl avf2ovf [standard options] \
   [-clip xmin ymin zmin xmax ymax zmax] [-dataformat <text|b4|b8>] \
   [-fileformat <ovf|npy> version] [-flip flipstr] [-grid <rect|irreg>] \
   [-info] [-keepbb] [-mag] [-pertran xoff yoff zoff] [-q] \
   [-pickcols cols] [-resample xstep ystep zstep order] [-resetbasept] \
   [-rpertran rxoff ryoff rzoff]  [-shiftbasept xbase ybase zbase] \
   [-subsample period] [infile [outfile]]
```

where

-clip xmin ymin zmin xmax ymax zmax The 6 arguments specify the vertices of a bounding clip box. Only mesh points inside the clip box are brought over into the output file. Any of the arguments may be set to "-" to use the corresponding value from the input file, i.e., to not clip along that box face.

-dataformat <text|b4|b8> Specify output data format, either ASCII text (`text`), 4-byte binary (`b4`), or 8-byte binary (`b8`). For OOMMF OVF output files, the default is text (note that the OVF format has an ASCII text header in all cases). For Python NumPy NPY output files the default is 8-byte binary. For OOMMF OVF version 2 output, the text option can additionally include a C-style printf format string, e.g., `-dataformat "text %16.12e"` (note the quotes to keep this a single argument to `-dataformat`).

-fileformat <ovf|npy> version Specify the output file format and version, either OOMMF OVF version 1 (default) or 2, or the Python NumPy array file format version 1.

-flip flipstr Provides an axis coordinate transformation. Flipstr has the form A:B:C, where A, B, C is a permutation of x, y, z, with an optional minus sign on each entry. The first component A denotes the axis to which x is mapped, B where y is mapped, and C where z is mapped. The default is the identity map, `x:y:z`. To rotate 90° about the z-axis, use "-flip y:-x:z", which sends x to the $+y$ axis, y to the -x axis, and leaves z unchanged.

-grid <rect|irreg> Specify output grid structure. The default is `rect`, which will output a regular rectangular grid if the input is recognized as a regular rectangular grid. The option "-grid irreg" forces irregular mesh style output.

-info Instead of converting the file, print information about the file, such as size, range, and descriptive text from the file header.

-keepbb If the `-clip` option is used, then normally the spatial extent, i.e., the boundary, of the output is clipped to the specified clip box. If `-keepbb` (keep bounding box) is given, then the spatial extent of the output file is taken directly from the input file. Clipping is still applied to the data vectors; `-keepbb` affects only the action of the clip box on the boundary.

-mag Write out a scalar valued field instead of a vector value field, where the scalar values are the magnitude $|v(r)|$ of the vector values at each point r. This option is only supported for OOMMF OVF version 2 output.

-pertran xoff yoff zoff Translates field with respect to location coordiates, by amount $(xoff, yoff, zoff)$, in a periodic fashion. For example, if $(xoff, yoff, zoff)$ is (50e-9, 0, 0), then a vector v at position (rx, ry, rz) in the original file is positioned instead at $(rx +$ 50e-9, $ry, rz)$ in the output file. If the spatial extent of the x coordinate in the input file runs from $xmin$ to $xmax$, and if $rx +$ 50e-9 is larger than $xmax$, then v will be placed

at $rx + 50\text{e-}9 - xmax + xmin$ instead. Translations are rounded to the nearest full step; aside from any clipping, the output file has the exact same spatial extent and sample locations as the original file. If both translation and clipping are requested, then the clipping is applied after the translation.

-q Quiet operation — don't print informational messages.

-pickcols cols Select components (columns) from infile to export to outfile. The parameter *cols* is a comma-separated list of 0-based indices into infile. The index order governs the order in outfile. The default is to export all columns in their original order. The value dimension of the output file is the length of *cols*. Note that OVF 1 format is restricted to value dimension 3, so OVF 1 output with *cols* lists of length other than 3 will be subject to either value truncation or zero padding.

-resample xstep ystep zstep <0|1|3> Resample grid using specified step sizes, in problem units (i.e., meters). Each step size must exactly divide the grid extent in the corresponding direction, after any clipping. (That is, the export mesh consists of full cells only.) The last argument specifies the polynomial interpolation order: 0 for nearest value, 1 for trilinear interpolation, or 3 for fitting with tricubic Catmull-Rom splines. This control is only available for input files having a rectangular grid structure. Default is no resampling.

-resetbasept Adjusts the mesh base point (x/y/zbase) and reorders the mesh data so that all node steps (x/y/zstepsize) are non-negative. This changes the mesh internal data structure but does not affect the mesh location or extent. Rectangular grids only.

-rpertran rxoff ryoff rzoff Similar to -pertran, except the offsets $(rxoff, ryoff, rzoff)$ are interpreted as offsets in the range $[0, 1]$ taken relative to the spatial extents of the x, y, and z coordinates. For example, if $xmax - xmin = 500\text{e-}9$, then an *rxoff* value of 0.1 is equivalent to an *xoff* value of 50e-9.

-shiftbasept xbase ybase zbase Shifts the mesh base point to the specified position, effectively translating the full mesh. Rectangular grids only. If multiple options are selected, then **-shiftbasept** is applied last.

-subsample period Reduce point density in output by subsampling input with specified period along x, y, and z axes. For example, if period is 2, then the output will have only 1/8th as many points as the input. This control is only available for input files having a rectangular grid structure. Default value is 1, i.e., no subsampling.

infile Name of input file to process. Must be one of the recognized formats, OVF 0.0, OVF 1.0, OVF 2.0, or VIO. If no file is specified, reads from stdin.

outfile Name of output file. If no file is specified, writes to stdout.

There are also two recognized but deprecated options, **-format** and **-ovf**. The former is replaced by **-dataformat** and the latter superceded by **-fileformat**.

The `-clip` option is useful when one needs to do analysis on a small piece of a large simulation. The `-info` option is helpful here to discover the extents of the original mesh. The `-clip` option can also be used with `-resample` to enlarge the mesh.

The `-flip` option can be used to align different simulations to the same orientation. It can also be used to change a file into its mirror image; for example, "-flip -x:y:z" reflects the mesh through the yz-plane.

If multiple operations are specified, then the operation order is clip, resample, subsample, flip, and translate.

The `-dataformat text` and `-grid irreg` options are handy for preparing files for import into non-OOMMF applications, since all non-data lines are readily identified by a leading "`#`," and each data line is a 6-tuple consisting of the node location and vector value. Pay attention, however, to the scaling of the vector value as specified by "`# valueunit`" and "`# valuemultiplier`" header lines (OVF version 1 only).

The **avf2ovf** command processes only one file at a time. To convert a large collection of files use the looping facilities in your command shell. For example, if you are running a Bourne shell derivative such `bash` (common on Linux) or `zsh` (macOS), you can convert all `.omf` files in the current directory to NumPy NPY format with the command

```
for i in *.omf ; do \
  tclsh oommf.tcl avf2ovf -fileformat npy 1 $i ${i%.omf}.npy ; \
done
```

Here the shell variable `i` is set in turn to the name of each `.omf` file, and fed as the `infile` to `avf2ovf`. For the `outfile`, the syntax `${i%.omf}.npy` replaces the `.omf` extension with `.npy`.

The equivalent on Windows is

```
for %i in (*.omf) do tclsh oommf.tcl avf2ovf -fileformat npy 1 %i %~ni.npy
```

In this case `%~ni.npy` replaces the filename extension.

For file format details, see the OVF file description.

Known Bugs

If the input file contains an explicit boundary polygon (cf. the `boundary` entry in the Segment Header block subsection of the OVF file description) then the output file will also contain an explicit boundary polygon. If clipping is active, then the output boundary polygon is formed by moving the vertices from the input boundary polygon as necessary to bring them into the clipping region. This is arguably not correct, in particular for boundary edges that don't run parallel to a coordinate axis.

16.4 Making Bitmaps from Vector Fields: avf2ppm

The **avf2ppm** utility converts a collection of vector field files (e.g., `.omf`, `.ovf`) into color bitmaps suitable for inclusion into documents or collating into movies. The command line arguments control filename and format selection, while plain-text configuration files, modeled after the **mmDisp** configuration dialog box, specify display parameters.

Launching

The **avf2ppm** launch command is:

```
tclsh oommf.tcl avf2ppm [standard options] [-config file] [-f] \
   [-filter program] [-format <P3|P6|B24|PNG>] [-ipat pattern] \
   [-opatexp regexp] [-opatsub sub] [-v level] [infile ...]
```

where

-config file User configuration file that specifies image display parameters. This file is discussed in detail below.

-f Force overwriting of existing (output) files. By default, if **avf2ppm** tries to create a file, say `foo.ppm`, that already exists, it generates instead a new name of the form `foo.ppm-000`, or `foo.ppm-001`, ..., or `foo.ppm-999`, that doesn't exist and writes to that instead. The `-f` flag disallows alternate filename generation, and overwrites `foo.ppm` instead.

-filter program Post-processing application to run on each **avf2ppm** output file. May be a pipeline of several programs.

-format <P3|P6|B24|PNG> Specify the output image file format. Currently supported formats are the true color *Portable Pixmap* (PPM) formats P3 (ASCII text) and P6 (binary), the uncompressed Windows BMP 24 bits-per-pixel format, and the compressed *Portable Network Graphics* (PNG) format. Conversion to the PNG format requires either Tk 8.6+, or earlier Tk plus the *Img*[3] package. The default format is P6.

-ipat pattern Specify input files using a pattern with "glob-style" wildcards. Mostly useful in DOS.

-opatexp regexp Specify the "regular expression" applied to input filenames to determine portion to be replaced in generation of output filenames. The default regular expression is: `(\.[^.]?[^.]?[^.]?$|$)`

[3] https://wiki.tcl-lang.org/page/Img

-opatsub sub The string with which to replace the portion of input filenames matched by the `-opatexp regexp` during output filename generation. The default is `.ppm` for type P3 and P6, `.bmp` for B24, and `.png` for PNG file output.

-v level Verbosity (informational message) level, with 0 generating only error messages, and larger numbers generating additional information. The `level` value is an integer, defaulting to 1.

infile ... List of input files to process.

The file specification options require some explanation. Input files may be specified either by an explicit list (`infile ...`), or by giving a wildcard pattern, e.g., `-ipat *.omf`, which is expanded in the usual way by **avf2ppm** (using the Tcl command `glob`). Unix shells (sh, csh, etc.) automatically expand wildcards before handing control over to the invoked application, so the `-ipat` option is not needed (although it is useful in case of a "command-line too long" error). DOS does not do this expansion, so you must use `-ipat` to get wildcard expansion in Windows.

As each input file is processed, a name for the output file is produced from the input filename by rules determined by handing the `-opatexp` and `-opatsub` expressions to the Tcl `regsub` command. Refer to the Tcl `regsub` documentation for details, but essentially whatever portion of the input filename is matched by the `-opatexp` expression is removed and replaced by the `-opatsub` string. The default `-opatexp` expression matches against any filename extension of up to 3 characters, and the default `-opatsub` string replaces this with the extension `.ppm` or `.bmp`.

If you have command line image processing "filter" programs, e.g., **ppmtogif** (part of the NetPBM package), then you can use the `-filter` option to pipe the output of **avf2ppm** through that filter before it is written to the output file specified by the `-opat*` expressions. If the processing changes the format of the file, (e.g., **ppmtogif** converts from PPM to GIF), then you will likely want to specify a `-opatsub` different from the default.

Here is an example that processes all files with the `.omf` extension, sending the output through **ppmtogif** before saving the results in files with the extension `.gif`:

```
tclsh oommf.tcl avf2ppm -ipat *.omf -opatsub .gif -filter ppmtogif
```

(On Unix, either drop the `-ipat` flag, or use quotes to protect the input file specification string from expansion by the shell, as in `-ipat '*.omf'`.) You may also pipe together multiple filters, e.g., `-filter "ppmquant 256 | ppmtogif"`.

Configuration files

The details of the conversion process are specified by plain-text configuration files, in the same format as the **mmDisp** configuration file page 160).

Each of the configurable parameters is an element in an array named `plot_config`. The default values for this array are read first from the main configuration file

```
oommf/app/mmdisp/scripts/avf2ppm.config
```

followed by the local customization file

```
oommf/app/mmdisp/scripts/local/avf2ppm.config
```

if it exists. Lastly, any files passed as `-config` options on the command line are input. Each of these parameters is interpreted as explained in the **mmDisp** documentation, except that **avf2ppm** ignores the `misc,defaultwindowwidth` and `misc,defaultwindowheight` parameters, and the following additional parameters are available:

arrow,antialias If 1, then each pixel along the edge of an arrow is drawn not with the color of the arrow, but with a mixture of the arrow color and the background color. This makes arrow boundaries appear less jagged, but increases computation time. Also, the colors used in the anti-aliased pixels are not drawn from the arrow or pixel colormap discretizations, so color allocation in the output bitmap may increase dramatically.

arrow,outlinewidth Width of a colored outline around each arrow; this can improve visibility of an arrow when it is overlayed against a background with color similar to that of the arrow. Default value is zero, meaning no outline. A value of 1 produces an outline with a recommended width, and other positive values are scaled relative to this.

arrow,outlinecolor If `arrow,outlinewidth` is positive, then this is the color of the arrow outline.

misc,boundarypos Placement of the bounding polygon, either `back` or `front`, i.e., either behind or in front of the rendered arrows and pixel elements.

misc,matwidth Specifies the width, in pixels, of a mat (frame) around the outer edge of the image. The mat is drawn in front of all other objects. To disable, set matwidth to 0.

misc,matcolor Color of the mat.

misc,width, misc,height Maximum width and height of the output bitmap, in pixels. If `misc,croptomargin` is enabled, then one or both of these dimensions may be shortened.

misc,croptomargin If disabled (0), then any leftover space in the bitmap (of dimensions `misc,width` by `misc,height`) after packing the image are filled with the background color. If enabled (1), then the bitmap is cropped at the margins specified by `misc,margin`. **NOTE:** Some movie formats require that bitmap dimensions be multiples of 8 or 16. For such purposes, you should disable `misc,croptomargin` and specify appropriate dimensions directly with `misc,width` and `misc,height`.

misc,crop Deprecated option replaced by `misc,croptomargin`.

The default configuration file (page 185) can be used as a starting point for user configuration files. You may also use configuration files produced by the **File|Write config...** command in **mmDisp**, although any of the above **avf2ppm**-specific parameters that you wish to use will have to be added manually, using a plain text editor. You may omit any entries that you do not want to change from the default. You may "layer" configuration files by specifying multiple user configuration files on the command line. These are processed from left to right, with the last value set for each entry taking precedence.

The `misc,datascale` and `arrow,coloroffset`, `pixel,coloroffset` configuration options can be used to control the mapping of values to the selected colormap. See the **mmDisp** documentation for details.

The `oommf/app/mmdisp/` directory contains the files `colorbar.config`, `colorbar.ovf` and `colorwheel.config`, `colorwheel.ovf` that may be used to create colorbar and colorwheel images that can be embedded in figures for publications and presentations. Refer to the notes included at the bottom of `colorbar.config` for details.

16.5 Making PostScript and Scalable Vector Graphics from Vector Fields: avf2ps

The **avf2ps** utility creates a collection of color Encapsulated PostScript (`.eps`) or Scalable Vector Graphics (`.svg`) files from a collection of vector field files (e.g., `.omf`, `.ovf`). Third party tools can be used to convert `.eps` and `.svg` files into other formats such as the Portable Document Format that can be embedded into larger documents. An advantage of `.eps` and `.svg` files over bitmap images such as `.png` or `.tiff` is that arrows in the former scale without pixelization to any magnification. Operation of the **avf2ps** command is modeled after the **avf2ppm** command and the print dialog box in **mmDisp**.

Launching

The **avf2ps** launch command is:

```
tclsh oommf.tcl avf2ps [standard options] [-config file] [-f] \
   [-filter program] [-format fmt] [-ipat pattern] [-opatexp regexp] \
   [-opatsub sub] [-v level] [infile ...]
```

where

-config file User configuration file that specifies image display parameters. This file is discussed in detail below.

-f Force overwriting of existing (output) files. By default, if **avf2ps** tries to create a file, say `foo.ps`, that already exists, it generates instead a new name of the form `foo.ps-000`, or `foo.ps-001`, ..., or `foo.ps-999`, that doesn't exist and writes to that instead. The `-f` flag disallows alternate filename generation, and overwrites `foo.ps` instead.

```
array set plot_config {
    arrow,status       1               misc,background    #FFFFFF
    arrow,antialias    1               misc,drawboundary  1
    arrow,outlinewidth 0.0             misc,boundarywidth 1
    arrow,outlinecolor #000000         misc,boundarycolor #000000
    arrow,colormap     Red-Black-Blue  misc,boundarypos   front
    arrow,colorcount   100             misc,matwidth      0
    arrow,quantity     z               misc,matcolor      #FFFFFF
    arrow,coloroffset  0               misc,margin        10
    arrow,colorreverse 0               misc,width         640
    arrow,autosample   1               misc,height        480
    arrow,subsample    10              misc,croptomargin  1
    arrow,size         1               misc,zoom          0
    arrow,viewscale    1               misc,rotation      0
                                       misc,datascale     0
    pixel,status       1               misc,relcenterpt   {0.5 0.5 0.5}
    pixel,colormap     Teal-White-Red
    pixel,colorcount   100             viewaxis            +z
    pixel,opaque       1               viewaxis,xarrowspan {}
    pixel,quantity     x               viewaxis,xpixelspan {}
    pixel,coloroffset  0               viewaxis,yarrowspan {}
    pixel,colorreverse 0               viewaxis,ypixelspan {}
    pixel,autosample   1               viewaxis,zarrowspan {}
    pixel,subsample    0               viewaxis,zpixelspan {}
    pixel,size         1
}
```

Figure 16.1: Contents of default configuration file `avf2ppm.config`. (description)

-filter program Post-processing application to run on each **avf2ps** output file. May be a pipeline of several programs.

-format fmt Select output format, either `EPS` for Encapsulated PostScript (default) or `SVG` for Scalable Vector Graphics.

-ipat pattern Specify input files using a pattern with "glob-style" wildcards. Mostly useful in DOS.

-opatexp regexp Specify the "regular expression" applied to input filenames to determine portion to be replaced in generation of output filenames. The default regular expression is: `(\.[^.]?[^.]?[^.]?$|$)`

-opatsub sub The string with which to replace the portion of input filenames matched by the `-opatexp regexp` during output filename generation. The default is `.eps` for Encapsulated PostScript output and `.svg` for Scalable Vector Graphics.

-v level Verbosity (informational message) level, with 0 generating only error messages, and larger numbers generating additional information. The `level` value is an integer, defaulting to 1.

infile ... List of input files to process.

The file specification options, `-ipat`, `-opatexp`, and `-opatsub`, are interpreted in the same manner as for the **avf2ppm** application.

If you have command line PostScript processing "filter" programs, e.g., **ghostscript**, then you can use the `-filter` option to pipe the output of **avf2ps** through that filter before it is written to the output file specified by the `-opat*` expressions. If the processing changes the format of the file, (e.g., from PostScript to PDF), then you will likely want to specify a `-opatsub` different from the default.

Here is an example that processes all files with the `.ovf` extension, sending the output through **ps2pdf** (part of the ghostscript package) before saving the results in files with the extension `.pdf`:

```
tclsh oommf.tcl avf2ps -ipat *.ovf -opatsub .pdf -filter "ps2pdf - -"
```

On Unix, either drop the `-ipat` flag, or use quotes to protect the input file specification string from expansion by the shell, as in `-ipat '*.ovf'`.

Configuration files

The details of the conversion process are specified by plain-text configuration files, in the same format as the **mmDisp** configuration file page 160).

The arrays `plot_config` and `print_config` hold the configurable parameters. The default values for these arrays are read first from the main configuration file

`oommf/app/mmdisp/scripts/avf2ps.config`

followed by the local customization file

`oommf/app/mmdisp/scripts/local/avf2ps.config`

if it exists. Lastly, any files passed as `-config` options on the command line are input. Each of these parameters is interpreted as explained in the **mmDisp** documentation, except that **avf2ps** ignores the `misc,defaultwindowwidth` and `misc,defaultwindowheight` parameters, and the following additional parameters are available:

arrow,length Full length of arrow, from tail to tip. All of the arrow dimensions, including this one, are measured relative to the sample cell size. The default value for `arrow,length` is 0.9, which being less than 1.0 ensures that arrows in neighboring cells don't overlap.

arrow,width Width of the arrow shaft.

arrow,tipwidth Width of the arrow at its tip

arrow,headwidth Width of the arrowhead at its widest point.

arrow,headilength Distance from the tip of the arrow back to the point where the arrowhead detaches from the shaft.

arrow,headolength Distance along the shaft from the tip of the arrow back to the widest point of the arrowhead.

arrow,outlinewidth Width of a colored outline around each arrow; this can improve visibility of an arrow when it is overlayed against a background with color similar to that of the arrow. Default value is zero, meaning no outline. Reasonable non-zero values are generally some fraction of the arrow shaft width.

arrow,outlinecolor If `arrow,outlinewidth` is positive, then this is the color of the arrow outline.

misc,boundarypos Placement of the bounding polygon, either `back` or `front`, i.e., either behind or in front of the rendered arrows and pixel elements.

misc,matwidth Specifies the width, in pixels, of a mat (frame) around the outer edge of the image. The mat is drawn inwards from the margin, in front of all other objects. To disable, set matwidth to 0.

misc,matcolor Color of the mat.

misc,width, misc,height Maximum width and height of the output, in pixels. One or both of these dimensions may be shortened if `misc,croptomargin` is enabled.

misc,croptomargin If disabled (0), then any leftover space in the output (of dimensions `misc,width` by `misc,height`) after packing the image are filled with the background color. If enabled (1), then the output is cropped at the margins specified by `misc,margin`. **NOTES:** If the arrow or pixel glyphs are oversized (`size` bigger than 1), then they may extend beyond the margin. In this case the glyphs will be clipped by `misc,croptomargin`. If this is not desired, then either increase the margin size to compensate, or else disable `misc,croptomargin` and use third party tools such as **epstool**,

```
epstool --copy --bbox input.eps output.eps
```

or **epstopdf** + **pdfcrop** to obtain tight cropping.

misc,crop Deprecated option replaced by `misc,croptomargin`.

The default configuration file (page 189) can be used as a starting point for user configuration files. You may also use configuration files produced by the **File|Write config...** command in **mmDisp**, although any of the above **avf2ps**-specific parameters that you wish to use will have to be added manually, using a plain text editor. You may omit any entries that you do not want to change from the default. You may "layer" configuration files by specifying multiple user configuration files on the command line. These are processed from left to right, with the last value set for each entry taking precedence.

The `misc,datascale` and `arrow,coloroffset`, `pixel,coloroffset` configuration options can be used to control the mapping of values to the selected colormap. See the **mmDisp** documentation for details.

16.6 Making Scalable Vector Graphics from Vector Fields: avf2svg

The **avf2svg** command is a synonym for "`avf2ps -format svg`". See the **avf2ps** documentation for details.

16.7 Vector Field File Difference: avfdiff

The **avfdiff** program computes differences between vector field files in any of the recognized formats. The input data must lie on rectangular meshes with identical dimensions.

Launching

The **avfdiff** launch command is:

```
array set plot_config {
  arrow,status       1                  misc,background     #FFFFFF
  arrow,colormap     Red-Black-Blue     misc,drawboundary   1
  arrow,colorcount   100                misc,boundarywidth  1
  arrow,quantity     z                  misc,boundarycolor  #000000
  arrow,coloroffset  0                  misc,boundarypos    front
  arrow,colorreverse 0                  misc,matwidth       0
  arrow,autosample   1                  misc,matcolor       #FFFFFF
  arrow,subsample    10                 misc,margin         10
  arrow,size         1                  misc,width          640
  arrow,length       0.9                misc,height         480
  arrow,width        0.075              misc,croptomargin   1
  arrow,tipwidth     0.01875            misc,zoom           0
  arrow,headwidth    0.4                misc,rotation       0
  arrow,headilength  0.4                misc,datascale      0
  arrow,headolength  0.5                misc,relcenterpt {0.5 0.5 0.5}
  arrow,outlinewidth 0.0
  arrow,outlinecolor #000000            viewaxis            +z
                                        viewaxis,xarrowspan {}
  pixel,status       1                  viewaxis,xpixelspan {}
  pixel,colormap     Teal-White-Red     viewaxis,yarrowspan {}
  pixel,colorcount   225                viewaxis,ypixelspan {}
  pixel,opaque       1                  viewaxis,zarrowspan {}
  pixel,quantity     x                  viewaxis,zpixelspan {}
  pixel,coloroffset  0.
  pixel,colorreverse 0
  pixel,colorinvert  0
  pixel,autosample   1
  pixel,subsample    0
  pixel,size         1
}
array set print_config {
  orient   landscape                    tmargin   1.0
  paper    letter                       lmargin   1.0
  hpos     center                       pwidth    6.0
  vpos     center                       pheight   6.0
  units    in                           croptoview 1
}
```

Figure 16.2: Contents of default configuration file `avf2ps.config`. (description)

```
tclsh oommf.tcl avfdiff [standard options] [-cross] [-filesort method] \
   [-info] [-numfmt fmt] [-odt label units valexpr] \
   [-resample fileselect interp_order] file-0 file-1 [... file-n]
```

where

-cross Compute the pointwise vector cross product of each `file-k` against `file-0` instead of subtraction.

-filesort method Specifies the sorting order to apply to the target file list, `file-1` through `file-n`. The order is important when using the `-odt` option, because it controls the order of the rows in the output. Parameter `method` should be a valid option string for the Tcl command `lsort`, e.g., "-ascii -decreasing". Note that the `lsort` sort options all begin with a hyphen, "-", and that if you want to use multiple options they must be grouped as one element to `-filesort` (by, for example, placing quotes around the list). If this option is not specified then the order is as presented on the command line (or as produced by wildcard expansion).

-info Prints statistics on file differences. If this option is selected then no output files are created.

-numfmt fmt Parameter `fmt` specifies a C-style output format for numeric data if `-info` or `-odt` is selected. Default value is "`%- #20.15g`".

-odt label units valexpr Computes the file differences, but instead of writing difference files to disk this option writes OOMMF Data Table (ODT) output to stdout. The ODT output consists of eight columns. The first column is an index column identifying the target file (`file-1` through `file-n`). The `label` parameter is a string specifying the label for this column, and likewise the `units` parameter is a string specifying the units for the column. The third parameter, `valexpr`, is any valid Tcl `expr` expression that may include the special variables `$i`, `$f1`, `$f2`, ..., `$d1`, `$d2`, ...; here `$i` is the 0-based index of the file in the target file list (`file-1` is index 0, `file-2` is index 1, etc.), `$f1` is the first number appearing in the target filename, `$f2` is the second number appearing in the target filename, `$d1` is the first number appearing in the "Desc" fields in the header of the target file, etc. This control is analogous to the `-index` option to `avf2odt`. The next three columns are the sum of each of the vector components in the difference. The last four columns are the averaged L^1 norm, the normalized L^2 norm, minimum component absolute value, and maximum component absolute value of the difference; these columns correspond to those produced by the `-extravals` option to `avf2odt`.

-resample <0|n> <0|1|3> Resample either the base file (`file-0`) to match the resolutions of the target files (`file-1` through `file-n`), or resample each target file to match the resolution of the base file. Set `fileselect` to 0 for the former, to n for the latter. The second argument specifies the polynomial interpolation order: 0 for nearest value,

1 for trilinear interpolation, or 3 for fitting with tricubic Catmull-Rom splines. Default is no resampling.

file-0 Name of input file to subtract from other files. Must be either an OVF 1.0 file in the rectangular mesh subformat, or an VIO file. Required.

file-1 Name of first input file from which `file-0` is to be subtracted. Must also be either an OVF 1.0 file in the rectangular mesh subformat, or an VIO file, and must have the same dimensions as `file-0`. Required.

... file-n Optional additional files from which `file-0` is to be subtracted, with the same requirements as `file-1`.

If neither `-info` nor `-odt` are specified, then for each target file `file-1` through `file-n` a separate output file is generated, in the OVF 1.0 format. Each output file has a name based on the name of corresponding input file, with a `-diff` suffix. If a file with the same name already exists, it will be overwritten.

For output file format details, see the OVF file description.

16.8 Field from Magnetization (3D): computefield

Inspired by the 2D-solver based **mag2hfield**, the **computefield** application uses the 3D Oxs solver engine to compute demagnetization and other fields from a magnetization snapshot. Various options allow fields to be computed outside the original simulation volume (say the stray field above a thin film sample) or to isolate the field arising from one portion of the sample (for example from one layer in a multilayer structure).

Launching

The **computefield** launch command is:

```
tclsh oommf.tcl computefield [standard options] \
   [-cwd directory] [-meshOvf meshovffile] [-Msfile Msinputfile] \
   [-Msmask maskexpr] [-Msspec spec] [-outdemagenergy demagenergyfile] \
   [-outM magfile] [-outmif miffile] [-pbc axes] [-runboxsi <0|1>] \
   [-runopts boxsiopts] [-simulationbox xmin ymin zmin xmax ymax zmax] \
   [-useroutputs "name1 file1 ..."] \
   [-userscriptfile energyscriptfile] <m0_infile> [outdemagfile]
```

where

-cwd directory Make directory the active working directory.

-meshOvf meshovffile Normally the mesh cell sizes and simulation dimensions are inferred from the `m0_infile`, but the `-meshOvf` option overrides the default with values from `meshovffile`. This is useful primarily in settings where the initial magnetization is prescribed generically using a coarser mesh than the one intended for the simulation—for example, the left half of the simulation is to have magnetization pointing up, the right half pointing down, defined via a mesh with only two (large) cells. In this case a separate OVF file may be used as an argument to both `-Msfile` and `-meshOvf` to define the part geometry and meshing.

-Msfile Msinputfile Set pointwise saturation magnetization values from `Msinputfile`, overriding M_s values from `m0_infile`. Any points outside the span of `Msinputfile` have M_s set to 0.

-Msmask maskexpr At each point in the simulation volume, the Tcl expr-expression `maskexpr` is evaluated to rescale (multiply) the otherwise set saturation magnetization M_s. The expr-expression may include the special variables `$x`, `$y`, `$z`, `$rx`, `$ry`, `$rz`, `$Ms`, `$mx`, `$my`, and `$mz`. The first three are the raw pointwise location in meters, the second three are the relative location (each in the range [0,1]), `$Ms` is the saturation magnetization at the point, and `($mx,$my,$mz)` is the reduced (unit normalized) magnetization. Analogous to the `-wgtfunc` option of **avf2odt**, `maskexpr` is typically a 0/1 valued function used to mask off a portion of the simulation part. See the first example below.

-Msspec spec Override the M_s values from `m0_infile` with the (quoted) `spec` string in the form of an inline `Specify` block. Note: If `spec` is a single numeric value representing a uniform saturation magnetization (in A/m), then that value is applied but restricted to the simulation volume described by `m0_infile` or `meshovffile`, with M_s set to 0 A/m in any extended volume requested by the `-simulationbox` option.

At most one of `-Msfile` and `-Msspec` may be used.

-outdemagenergy demagenergyfile Write the demagnetization energy density scalar field (.oef) to `demagenergyfile`.

-outM magfile Saves a copy of the magnetization as used in the computation, incorporating `m0_infile`, `meshovffile`, and `Ms/file/spec/mask` options. This can be helpful for checking that input parameters, in particular `-Msmask`, are behaving as expected.

-outmif miffile The **computefield** application creates a temporary MIF used as input for running **Boxsi**. This MIF file is deleted after **Boxsi** exits, but the `-outmif` option will save a copy in `miffile`.

-pbc axes Creates a periodic mesh for the simulation. The `axes` should be a string of one or more of the letters '`x`', '`y`', and '`z`', denoting the periodic direction(s), as used by the `periodic` option to the `Oxs_PeriodicRectangularMesh Specify` block (Sec. 7.3.2).

-runboxsi <0|1> By default, **computefield** runs Boxsi on its constructed MIF file to compute the requested output fields. This behavior is short circuited with `-runboxsi 0`, which can be used in conjunction with `-outmif` to allow additional user edits to the MIF file preceding a manual **Boxsi** run.

-runopts boxsiopts Command line options passed to the automatic **Boxsi** run. The `boxsiopts` string must be quoted as a single argument to **computefield**.

-simulationbox xmin ymin zmin xmax ymax zmax Use the specified values for the simulation volume, overriding the extents in the input `m0_infile` or `meshovffile` files. The x/y/z/min/max values can be listed as either six individual values or quoted as a single element to `computefield`. Any entry specified as a single hyphen ('-') will inherit the corresponding value from `m0_infile`/`meshovffile`. (The `-info` option to the command line utility `avf2ovf` can be used to view OVF file extents.)

The `simulationbox` option is typically used to compute stray fields outside the original simulation volume. If the original volume extends outside the requested volume, then magnetization outside the requested volume is ignored and a warning is printed.

-useroutputs "name1 file1 ..." A list, quoted as a single element on the command line, of alternating outputs and filenames, referring to `Oxs_Energy` objects specified in the separate `-userscriptfile`. This command line option augments and amends any `user_outputs` setting in the script file.

-userscriptfile energyscriptfile The only outputs **computefield** directly supports on the command line are demagnetization field, demagnetization energy density, and magnetization. However, the user can create an `energyscriptfile`, which is a separate Tcl script with additional `Oxs_Energy` objects to be included in the run. (See second example below.) The name of each desired output and the associated filename should be appended to the "`user_outputs`" Tcl list, either directly inside the script file, or via the `-useroutputs` command line option.

m0_infile Input magnetization (.omf) file. Required. This file sets the magnetization direction at each point, i.e., the MIF `m0` value in the `Oxs_TimeDriver` and `Oxs_MinDriver Specify` blocks. The pointwise saturation magnetization (MIF `Ms` option) and simulation volume extents (MIF atlas object) are also derived from this file unless overridden by other options.

outdemagfile Demagnetization output file name. Optional. If not specified then the demagnetization field is not output.

Examples

Consider a multilayer thin film having total thickness 30 nm, with the top layer 10 nm thick. A simulation is run and the magnetization at some interesting point is saved to the file

foo.omf. For efficiency, the simulation volume was set to contain only the film itself, but for analysis we want to know the stray field out to 150 nm above the top of the film. Moreover, we would like to know the portion of the stray field coming from just the top 10 nm layer. The command

```
tclsh oommf.tcl computefield -simulationbox - - - - - 180e-9 \
  -Msmask '20e-9<$z && $z<30e-9 ? 1 : 0' foo.omf foo-toplayer.ohf
```

will compute this and save the results in foo-toplayer.ohf. (On the Windows command line, remove or replace the backslash line continuation character with a caret, ^, and replace the single quotes with double quotes.)

As another example, suppose bar.omf holds Oxs_MinDriver::Spin normalized magnetization output for a sample with uniform saturation magnetization of 1400 kA/m. If we want to know the magnetocrystalline anisotropy field at that magnetization state, we can construct a separate file, say barK.tcl, that contains the anisotropy block from the original simulation:

```
Specify Oxs_UniaxialAnisotropy:K {
   K1 520e3
   axis {0 0 1}
}
lappend user_outputs Oxs_UniaxialAnisotropy:K:Field barK.ohf
```

The last line requests the field output be written to the file barK.ohf. We will feed this file to **computefield** using the -userscriptfile option. After closing barK.tcl we decide we would also like to have the anisotropy energy density, so we'll use the -useroutput option to request that, and for good measure request the demagnetization energy density with -outdemagenergy as well. The accompanying **computefield** command is

```
tclsh oommf.tcl computefield -Msspec 1.4e6 -userscriptfile barK.tcl \
   -useroutputs "Oxs_UniaxialAnisotropy:K:Energy\ density barK.oef" \
   -outdemagenergy bar-demag.oef bar.omf
```

Note that the demagnetization field is not requested and so won't be saved. (If that field is wanted, just add the filename after bar.omf.)

16.9 Cyclic Redundancy Check: crc32

The **crc32** application computes 32-bit cyclic redundancy checksums (CRC-32) on files.

Launching

The **crc32** command line is:

```
tclsh oommf.tcl crc32 [standard options] [-binary|-text] \
   [-decimal|-hex] [-v level] [file ...]
```

where

-binary|-text Select binary (default) or text input mode.

-decimal|-hex Output CRC value in decimal (default) or hexadecimal format.

-v level Verbosity (informational message) level, with 0 generating only error messages and minimal CRC output, and larger numbers generating additional information. The `level` value is an integer, defaulting to 1.

file ... List of files to process. If no files are listed, then input is read from stdin.

For each file in the input file list, the CRC-32 is computed and output. By default, the computation is on the raw byte stream (binary mode). However, if text mode is selected, then text mode translations, e.g., carriage return + newline → newline conversion, is performed before the CRC-32 computation. Text mode translations usually have no effect on Unix systems. For additional information on text mode, see the Tcl documentation for `fconfigure`, specifically "-translation auto."

If the verbosity level is 1 or greater, then the length of the byte stream as processed by the CRC-32 computation is also reported.

16.10 Killing OOMMF Processes: killoommf

The **killoommf** application terminates running OOMMF processes.

Launching

The **killoommf** command line is:

```
tclsh oommf.tcl killoommf [standard options] [-account name] \
   [-hostport port] [-pid] [-q] [-show] [-shownames] [-test] \
   [-timeout secs] oid [...]
```

where

-account name Specify the account name. The default is the same used by **mmLaunch**: the current user login name, except on Windows 9X, where the dummy account ID "oommf" may be used instead.

-hostport port Use the host server listening on `port`. Default is set by the `Net_Host port` setting in `oommf/config/options.tcl`, or by the environment variable `OOMMF_HOSTPORT` (which, if set, overrides the former). The standard setting is 15136.

-pid Select processes by system pid rather than OOMMF oid.

-q Quiet; don't print informational messages.

-show Don't kill anything, just print matching targets.

-shownames Don't kill anything, just print nicknames of matching targets, where nicknames are as set by OOMMF command line utility `nickname` or the MIF 2.1 `Destination` command (Sec. 17.3.2).

-test Don't kill anything, just test that targets are responding.

-timeout secs Maximum time to wait for response from servers, in seconds. Default is five seconds.

oid ... List of one or more oids (OOMMF ID's), application names, nicknames, or the keyword "all". Glob-style wildcards may also be used. This field is required (there are no default kill targets). If the `-pid` option is specified then numbers are interpreted as referring to system process ID's rather than OOMMF ID's.

The **killoommf** command affects processes that listen to OOMMF message traffic. These are the same applications that are listed in the **Running Applications** list of **mmLaunch**. The command

```
tclsh oommf.tcl killoommf all
```

is equivalent to the "`File|Exit All OOMMF`" menu option in **mmLaunch**.

An OOMMF application that does not respond to **killoommf** can be killed using the operating system pid (process identification) and the operating system facilities for terminating processes (e.g., `kill` on Unix, or the `Windows Task Manager` on Windows). The mapping from OOMMF oids to pids can be obtained from either the OOMMF command line program **pidinfo** or **killoommf** with the `-show` option.

16.11 Last Oxsii/Boxsi Run: lastjob

The **lastjob** command reads through Oxs log files and identifies the last simulation run, sorted by job start time. From information in the log file, **lastjob** constructs a command equivalent to that used to launch the last simulation, and then will either print or run that command. If the last simulation is not recorded as complete in the log file, and a restart is requested, then the simulation will be restarted with the `-restart 1` command line option. If a restart (checkpoint) file exists for the simulation, then the command will restart the simulation at the checkpoint state. If a restart file cannot be found, then the job restart will fail. (By default, **Oxsii** and **Boxsi** write checkpoint files (Sec. 7.3.5, page 93) to disk every fifteen minutes. If a simulation is aborted, for example by a system crash, then the checkpoint file can be used to restart the simulation.) Refer also to the **logreview** command line tool.

Launching

The **lastjob** launch command is:

```
tclsh oommf.tcl lastjob [-logfile logname] [-unfinished] [-v] <show|restart> \
   <oxsii|boxsi> [hostname] [username]
```

where

-logfile logname The name of the file to look in to determine the last job. Optional. The default is to look in the OOMMF root directory for either `oxsii.log` or `boxsi.log`, corresponding to whether **Oxsii** or **Boxsi** jobs are selected. Use "`-`" as the logname to read from stdin. In particular, log files can be filtered first through **logreview** for additional job selection control.

-unfinished Restrict search to unfinished jobs. Optional.

-v Request verbose output. Optional.

show|restart Selects whether to simply show the command or to attempt a restart. Required.

oxsii|boxsi Selects **Oxsii** or **Boxsi** jobs. Required.

hostname The name of the host machine to look for jobs for. This is optional, with the default being the name of the current machine. This option is useful if the log file is on a shared drive used by multiple hosts. This field is interpreted as a regular expression, so for example "`.*`" can be used to find the last job for all hosts.

username The name of the user to look for jobs for. This is optional, with the default being the name of the current user. This option is useful if the same log file is shared by multiple users. This field is interpreted as a regular expression, so for example "`.*`" can be used to find the last job by any user.

Note: If your command shell expands wildcards, as is common on Unix systems, then you may need to escape or quote regular expressions to protect them from expansion by the shell.

16.12 The OOMMF Host Server: launchhost

Under normal circumstances, the OOMMF host server (also known as the host service directory) is automatically launched in the background as needed by client applications. However, it can be useful, primarily in batch compute environments, to launch the host server explicitly in order to control the host server port address.

Launching

The **launchhost** command line is:

```
tclsh oommf.tcl launchhost [standard options] port
```

where

port Requested port number for host server to listen on. For non-privileged users, this usually has to be larger than 1024, or the special value 0 which signals the host server to open on a random, unused port. On success, **launchhost** writes the host port number actually used to stdout.

As described in the OOMMF architecture documentation, the host server (host service directory) plays a vital role in allowing various OOMMF applications to communicate with one another. To work, the host server port number must be known to all OOMMF applications. Typically this port number is determined by the `Net_Host port` setting in the file `oommf/config/options.tcl`, although this setting may be overridden by the environment variable `OOMMF_HOSTPORT`.

In batch-mode settings, however, it can occur that one wants to run multiple concurrent but independent OOMMF sessions on a single machine. One way to accomplish this is to set the environment variable `OOMMF_HOSTPORT` to distinct values in each session. A difficulty here is the bookkeeping necessary to ensure that each session really gets a distinct value. Using **launchhost** with `port` set to zero provides a straightforward solution to this problem. For example, consider the Bourne shell script:

```
#!/bin/sh
OOMMF_HOSTPORT=`tclsh oommf.tcl launchhost 0`
export OOMMF_HOSTPORT
tclsh oommf.tcl mmArchive
tclsh oommf.tcl boxsi sample.mif
tclsh oommf.tcl killoommf all
```

The second line (`OOMMF_HOSTPORT=...`) launches the host server on a random available port; the port selected is printed to stdout by **launchhost** and is then used to set the environment variable `OOMMF_HOSTPORT`. Note in particular the backticks around the **launchhost** command, which invoke command execution. For modern Bourne compatible shells such as **bash**, **dash**, **ksh**, and **zsh**, an alternative to backticks is

```
OOMMF_HOSTPORT=$(tclsh oommf.tcl launchhost 0)
```

which is rather more readable. For **csh** and derivatives, replace the two `OOMMF_HOSTPORT` lines with the single line

```
setenv OOMMF_HOSTPORT `tclsh oommf.tcl launchhost 0`
```

The subsequent commands launch an instance of **mmArchive** in the background and run **Boxsi** on the problem described by `sample.mif`. (By default, **Boxsi** runs in the foreground.) When **Boxsi** returns, the **killoommf** command is used to terminate all OOMMF processes in this session. (Alternatively, the **Boxsi** command option `-kill` may be used to the same effect as **killoommf**.)

If problems occur, a useful diagnostic is to insert the line

```
echo OOMMF_HOSTPORT: $OOMMF_HOSTPORT
```

immediately before the "`tclsh oommf.tcl mmArchive`" line.

16.13 Oxs Log File Review: logreview

The OOMMF applications **Oxsii** and **Boxsi** record status and error messages from job runs into log files (default `oommf/oxsii.log` and `oommf/boxsi.log`, respectively). These are plain text files, but they can grow rather long and messages from multiple **Oxsii** or **Boxsi** instances running simultaneously can interweave making the log difficult to interpret. The **logreview** application can make the logs easier to understand by extracting subsets of log records based on various criteria. (See too the command line tool **lastjob**, which also accesses Oxs log files.)

Launching

The **logreview** command line is:

```
tclsh oommf.tcl logreview [standard options] <logfile> \
  [-after timestamp] [-before timestamp] [-collate] [-firstrun] \
  [-glob globstr] [-grep regexpr] [-info] [-lastrun] [-machine name] \
  [-pid pid] [-runselect first last] [-user username]
```

where

logfile Specifies the input log file(s). This should be either `boxsi`, `oxsii`, `boxsi+oxsii`, or `-file` *`filename`* to specify a particular file. Multiple `-file` options may be specified. The default **Boxsi** and **Oxsii** log files, `boxsi.log` and `oxsii.log`, are located in the OOMMF root directory. (Note: The default log files were formerly named `oxsii.errors` and `boxsi.errors`.) Use "`-file -`" to read from stdin.

-after timestamp Select only log entries on or after `timestamp`, where `timestamp` is in a format recognized by the Tcl command `clock scan` in "Free Form Scan" mode.

-before timestamp Select only log entries before `timestamp`, where `timestamp` is in a format recognized by the Tcl command `clock scan` in "Free Form Scan" mode.

-collate Reorder log entries to group individual job run log records together. Job run log entries typically begin with a `Start:` record and finish with an `End` record, but if multiple instances of **Oxsii** or **Boxsi** are running simultaneously then log messages from the different instances can become intertwined. The `-collate` option retains the ordering of the first record in each job.

-firstrun Prints all records for the first run in the log. Shorthand for `-runselect 0 0`.

-glob globstr Restrict to log entries matching the glob-style pattern `globstr`, as used by the Tcl `string match` command. This is a full entry, multi-line match, so `globstr` will typically start and end with the asterisk '`*`' wildcard character. Backslash substitution is performed on `globstr`, so '`\n`' may be used to match against line breaks.

-grep regexpr Restrict to log entries matching the regular-expression pattern `regexpr`, as used by the Tcl `regexp` command. If the '`^`' (string start) and '`$`' (string end) anchors are not specified, then a match to any part of the log entry is accepted. Use the newline character '`\n`' to tie a pattern to the start of a line inside a log entry.

-info Print informational details about the log selection instead of the normal result. If `-collate` is specified then this output will include the number of job runs.

-lastrun Prints all records for the last run in the log. Shorthand for `-runselect end end`.

-machine name Restrict to log entries recorded from the specified machine.

-pid pid Restrict to log entries from processes with system process id `pid`. The `pid` parameter may be a quoted list of multiple PIDs to select entries from multiple processes. (For active OOMMF applications, the OOMMF command line program **pidinfo** can be used to map OOMMF IDs to system PIDs.) Operating systems recycle PIDs, and they are assigned separately on each machine, so it may be necessary to use the `-after`, `-before`, and `-machine` options to separate duplicate PID assignments.

-runselect first last Prints all records for jobs indexed `first` through `last`, inclusive, where `first` and `last` are 0-based indices as interpreted by the Tcl `lindex` command. In particular, `end` may be used to denote the last run. This option implies `-collate`. It is applied before `-glob` and `-grep` but after the other filtering options.

-user username Restrict to log entries from user `username`.

The result is printed to stdout. If `-info` is not requested, then the output will be a collection of records in the usual **Oxsii**/**Boxsi** plain text log format. Be aware that `Start:` records can be trimmed off by `-after` requests, and `End` records can be trimmed by `-before` requests, or missing altogether if a job aborts due to an error.

Examples

The following prints all **Oxsii** `Start:` and `End` records from yesterday afternoon, collated by job:

```
tclsh oommf.tcl logreview oxsii -collate -grep "\nStart:|\nEnd" \
   -after "12:00 yesterday" -before "17:00 yesterday"
```

The next example uses the `-info` and `-collate` options together to get a count of jobs that ran during some time window:

```
tclsh oommf.tcl logreview boxsi -info -collate \
   -after "10 days ago" -before "3 days ago"
```

To see the records for the last three runs in the **Boxsi** log:

```
tclsh oommf.tcl logreview boxsi -runselect end-2 end
```

Over time the default log files can become quite large. To trim records preceding some date, one can do

```
tclsh oommf.tcl logreview oxsii -after 1-Apr-2023 > trimmedlog.txt
```

and then move `trimmedlog.txt` to `oommf/oxsii.log`. Several of the ISO 8601 point-in-time specifications are also supported. For example,

```
tclsh oommf.tcl logreview -file trimmedlog.txt -before 20230430T235959
```

16.14 H Fields from Magnetization (2D): mag2hfield

The **mag2hfield** utility takes a MIF 1.1 micromagnetic problem specification file (`.mif`) and a magnetization file (`.omf`) and uses the mmSolve2D computation engine to calculate the resulting component (self-magnetostatic, exchange, crystalline anisotropy, Zeeman) and total energy and/or **H** fields. The main use of this utility to study the fields in a simulation using magnetization files generated by an earlier **mmSolve2D** run. (Deprecated; for most applications use the 3D Oxs-based **computefield** utility instead.)

Launching

The **mag2hfield** launch command is:

```
tclsh oommf.tcl mag2hfield [standard options]
   [-component [all,][anisotropy,][demag,][exchange,][total,][zeeman] \
   [-data [energy,][field]] [-energyfmt fmt] [-fieldstep #] \
   mif_file omf_file [omf_file2 ...]
```

where

-component [**all,**][**anisotropy,**][**demag,**][**exchange,**][**total,**][**zeeman**] Specify all desired energy/field components. Optional; default is `total`, which is the sum of the crystalline anisotropy, demagnetization (self-magnetostatic), exchange, and Zeeman (applied field) terms.

-data [**energy,**][**field**] Calculate energies, **H** fields, or both. Energy values are printed to stdout, **H** fields are written to files as described below. Optional; the default is `energy,field`.

-energyfmt fmt Output C printf-style format string for energy data. Optional. The default format string is `"%s"`.

-fieldstep # Applied field step index, following the schedule specified in the input MIF file (0 denotes the initial field). Optional; default is 0.

mif_file MIF micromagnetic problem specification file (.mif). Required.

omf_file Magnetization state file. This can be in any of the formats accepted by the `avfFile` record of the input MIF file. Required.

omf_file2 ... Optional additional magnetization state files.

The **H** field output file format is determined by the `Total Field Output Format` record of the input MIF 1.1 file (Sec. 17.1). The output file names are constructed using the form *basename*`-hanisotropy.ohf`, *basename*`-hzeeman.ohf`, etc., where *basename* is the input `.omf` magnetization file name, stripped of any trailing `.omf` or `.ovf` extension.

16.15 MIF Format Conversion: mifconvert

The **mifconvert** utility converts any of the MIF formats into the MIF 2.1 format used by the Oxs 3D solvers. It can also convert between the MIF 1.1 and MIF 1.2 formats generated by micromagnetic problem editor, mmProbEd.

As a migration aid, **mifconvert** will convert most files from the obsolete MIF 2.0 format used by OOMMF 1.2a2 into the newer MIF 2.1 format.

Launching

The **mifconvert** launch command is:

```
tclsh oommf.tcl mifconvert [-f|--force] [--format fmt]
   [-h|--help] [--nostagecheck] [-q|--quiet] [--unsafe]
   [-v|--version] input_file output_file
```

where

-f or --force Force overwrite of output file. Optional.

--format fmt Specify output format, where `fmt` is one of 1.1, 1.2, or 2.1. The 1.1 and 1.2 formats are available only if the input file is also in the 1.x format. Conversion from the 2.1 format to the 1.x formats is not supported. Optional; default setting is 2.1.

-h or --help Print help information and stop.

--nostagecheck Sets the `stage_count_check` parameter in the output Oxs_Driver Specify block to 0; this disables stage count consistency checks inside the Oxs solver. Optional. This option is only active when the output MIF format is 2.1.

-q or --quiet Suppress normal informational and warning messages. Optional.

--unsafe Runs embedded Tcl scripts, if any, in unsafe interpreter. Optional.

-v or --version Print version string and stop.

input_file Name of the import micromagnetic problem specification file, in MIF 1.1, MIF 1.2, or MIF 2.0 format. Use "-" to read from stdin. Required.

output_file Name of the export micromagnetic problem specification file. Use "-" to write to stdout. Required.

16.16 PyMIF to MIF/Tcl Format Conversion: mifpy2tcl

The **mifpy2tcl** utility converts PyMIF 1.0 files into MIF/Tcl 2.2 format.

Launching

The **mifpy2tcl** launch command is:

```
tclsh oommf.tcl mifpy2tcl
   [-h|--help] [-infile INFILE] [-outfile OUTFILE]
   [-parameters name value ...] [-v|--version]
```

where

-h or --help Print help information and stop.

-infile Read PyMIF input from INFILE. Default is stdin.

-outfile Write MIF/Tcl output to OUTFILE. Default is stdout.

-parameters One or more optional name + value pairs, passed to the PyMIF `Parameter` command.

-v or --version Print version string and stop.

This utility is used on the fly by the Oxs solvers when processing PyMIF input files, but is also provided for standalone use. It runs `oommf/app/oxs/mifpy2tcl.py` inside a Python 3 executable to process the PyMIF input file. Treat PyMIF files with the same care as you would other Python scripts.

16.17 Process Nicknames: nickname

The **nickname** command associates nicknames to running instances of OOMMF applications. These names are used by the MIF 2.x `Destination` command (Sec. 17.3.2).

Launching

The **nickname** command line is:

```
tclsh oommf.tcl nickname [standard options] [-account name] \
   [-hostport port] [-pid] [-timeout secs] oid nickname [nickname2 ...]
```

where

-account name Specify the account name. The default is the same used by **mmLaunch**: the current user login name, except on Windows 9X, where the dummy account ID "oommf" may be used instead.

-hostport port Use the host server listening on `port`. Default is set by the `Net_Host port` setting in `oommf/config/options.tcl`, or by the environment variable `OOMMF_HOSTPORT` (which, if set, overrides the former). The standard setting is 15136.

-pid Specify application instance to nickname by system PID (process identifier) rather than OID (OOMMF identifier).

-timeout secs Maximum time to wait for response from servers, in seconds. Default is five seconds.

oid The OOMMF ID of the running application instance to nickname, unless the `-pid` option is specified, in which case the system PID is specified instead.

nickname One or more nicknames to associate with the specified application instance. Each nickname must include at least one non-numeric character.

This command is used to associate nicknames with running instances of OOMMF applications. The MIF 2 Destination command can then use the nickname to link Oxs output to a given OOMMF application instance at problem load time. Nicknames for GUI applications can be viewed in the application About dialog box, or can be seen for any application via the -names option to the command line application **pidinfo**.

Note that nicknames can also be associated with OOMMF applications when they are started via the standard -nickname command line option, or by using the application:nickname syntax for applications launched by the MIF Destination command.

16.18 ODT Derived Quantity Calculator: odtcalc

The **odtcalc** utility reads an ODT file on stdin that contains one or more tables, and prints to stdout an ODT file consisting of the same tables augmented by additional columns as controlled by command line arguments. This utility enables the calculation and recording of new data table columns that can be computed from existing columns.

Launching

The **odtcalc** launch command is:

```
tclsh oommf.tcl odtcalc [standard options] [var unit expr ...] \
    <infile >outfile
```

where

var unit expr ... Each triplet of command line arguments determines the calculation to make for the production of a new column in the output data table. Each var value becomes the new Columns: entry in the data table header, labeling the new column of data. Each unit value becomes the new Units: entry in the data table header, reporting the measurement unit for the new column of data. Each expr value is a Tcl expression to be evaluated to compute each new data value to be stored in the new column of data. See below for more details.

<infile odtcalc reads its input from stdin. Use the redirection operator "<" to read input from a file.

>outfile odtcalc writes its output to stdout. Use the redirection operator ">" to send output to a file.

The computation of a new data value for each row of each new column of data is performed by passing the corresponding expr command line argument to Tcl's **expr** command. The standard collection of operators and functions are available, and also the variables $pi and $mu0 representing the commonly used values π and μ_0. The value of other columns in the same row may be accessed by use of the column label as a variable name. For example, the

value of the `Iteration` column can be used in `expr` by including the variable substitution `$Iteration`. When column labels include colons, the `expr` has the option of using just the portion of the column label after the last colon as the variable name. For example, the value of the `Oxs_UZeeman::Bx` column can be used in `expr` by including the variable substitution `$Bx`. When multiple triples specifying new data columns are provided, the values of earlier new columns may be used to compute the values of later new columns. The order of command line arguments controls the order of the new columns that are added to the right side of the data table.

Example

Suppose `ring.odt` contains hysteresis loop data from an **Oxsii** simulation where the field was applied in the xy-plane at an angle of 30° from the x-axis. The data file holds field and average magnetization axis component values B_x, B_y, m_x, and m_y. We want field and magnetization data projected onto the applied field axis. We can create those values using **odtcalc** like so:

```
tclsh oommf.tcl odtcalc B mT '$Bx*0.8660254037844365+$By*0.5' \
   m '$mx*0.8660254037844365+$my*0.5' '' \
   < ring.odt > ring-augmented.odt
```

Here $\cos(30°) = 0.8660254037844365$ and $\sin(30°) = 0.5$. The `ring-augmented.odt` file will have all the data in the original `ring.odt` file, plus two additional columns, labeled B with units of mT and m with empty units. (Note: On Windows replace the single quotes in the above command with double quotes. Also, the Windows command line uses the caret character ^ for line continuation instead of the backslash \.)

To extract just the B and m columns and prepare for import into a spreadsheet program supporting CSV (comma separated value) format, post-process with **odtcols**:

```
tclsh oommf.tcl odtcols -t csv B m < ring-augmented.odt > ring-export.dat
```

16.19 ODT Table Concatenation: odtcat

The **odtcat** utility reads an ODT file on stdin that contains one or more tables, and concatenates them together into a single table, creating a new ODT file consisting of a single table. When successive tables are joined, the tail of the first is truncated as necessary so that the specified control column is monotonic across the seam.

This tool is useful for fixing up ODT output from a simulation that was interrupted and restarted from checkpoint data one or more times.

Launching

The **odtcat** launch command is:

```
tclsh oommf.tcl odtcat [standard options] [-b overlap_lines] \
   [-c control_column] [-o order] [-q] <infile >outfile
```

where

-b overlap_lines Overlap window size. This is the maximum number of lines to retain when looking for overlap between two adjacent tables. This is also the upper limit on the number of lines that may be removed when two tables are joined. The default value is 100.

-c control_column Specifies control column, either by number or glob-string. Default is the glob string {`Oxs_TimeDriver:*:Simulation time`} `Oxs_MinDriver:*:Iteration`.

-o order Order selection: one of `increase`, `decrease`, `auto` (default), or `none`.

-q Quiet; don't write informational messages to stderr.

<infile odtcat reads its input from stdin. Use the redirection operator "<" to read input from a file.

>outfile odtcat writes its output to stdout. Use the redirection operator ">" to send output to a file.

The first table header is examined and compared against the control column specification to identify the control column. If multiple columns match the control column specification, then an error is reported and the process exits. The OOMMF command line utility **odtcols** with the **-s** command line switch can be used to view column headers before running **odtcat**.

Each table in the input stream is assumed to have the same layout as the first; header information between tables is summarily eliminated. As each table is encountered, a check is made that the new table has the same number of columns as the first. If not, an error is reported and processing is halted.

When subsequent table headers are encountered, the values in the control column in the tail of the preceding table and the head of the succeeding table are compared. The order selection is used to determine the position of the start of the latter table inside the tail of the former. If the data are not compatible with the specified ordering, then an error is reported an the program aborts. If identical values are discovered, then the matching lines in the earlier table are excluded from the output stream.

If the **-q** flag is not specified, then after processing is complete a report is written to stderr detailing the number of tables merged and the number of data lines eliminated.

16.20 ODT Column Extraction: odtcols

The **odtcols** utility extracts column subsets from ODT data table files.

Launching

The **odtcols** launch command is:

```
tclsh oommf.tcl odtcols [standard options] [-f format] \
   [-m missing] [-q] [-s] [-S] [-t output_type] \
   [-table select] [-no-table deselect] [-w colwidth] \
   [col ...] <infile >outfile
```

where

-f format C printf-style format string for each output item. Optional. The default format string is `"%$15s"`. Multiple `-f` options may be interspersed with column selections, in which case each format applies to subsequently selected columns.

-m missing String used on output to designate a missing value. Default is the two character open-close curly brace pair, {}, as specified by the ODT file format.

-q Silences some meaningless error messages, such as "broken pipe" when using the Unix head or tail utilities.

-s Produces a file summary instead of column extraction. Output includes table titles, column and row counts, and the header for each specified column. If no columns are specified, then the headers for all the columns are listed.

-S Same as `-s` option, except the column list is ignored; headers for all columns are reported.

-t output_type Specify the output format. Here output_type should be one of the strings `odt`, `csv`, or `bare`. The default is `odt`, the ODT file format. Selecting `csv` will yield a "Comma-Separated Values" (CSV) file, which can be read by many spreadsheet programs. The `bare` selection produces space separated numeric output, with no ODT header, trailer, or comment lines. The latter two options are intended as aids for transferring data to third party programs; in particular, such output is not in ODT format, and there is no support in OOMMF for translating back from CSV or bare format to ODT format.

-table select Select tables to include in output. Tables are selected by index number; the first table in the file has index 0. The select string consists of one or more selections separated by commas, where each selection is either an individual index number or a range with inclusive endpoints separated by a colon. Example select string: `0:3,7,9:12`. Default is all tables.

-no-table deselect Specify tables to exclude from output. The deselect string has the same format at the `-table select` string. Default is to print all tables, so the effective default deselect string is the empty set.

-w colwidth Minimum horizontal spacing to provide for each column on output. Optional. Default value is 15. Negative `colwidth` values will fill from the left, positive from the right. (This positions the post-formatted data string, retaining any space in the field width portion of the `-f format` specification.) Multiple `-w` options may be interspersed with column selections, in which case each width applies to subsequently selected columns.

col ... Output column selections. These may either be integers representing the position of the column in the input data (with the first column numbered as 0), or else arbitrary strings used for case-insensitive glob-style matching against the column headers. The columns are output in match order, obtained by processing the column selections from left to right. If no columns are specified then by default all columns are selected.

<infile odtcols reads its input from stdin. Use the redirection operator "<" to read input from a file.

>outfile odtcols writes its output to stdout. Use the redirection operator ">" to send output to a file.

Commonly the **-s** switch is used in a first pass, to reveal the column headers; specific column selections may then be made in a second, separate invocation. If no options or columns are specified, then the help message is displayed.

16.21 Oxs Package Management: oxspkg

The **oxspkg** command is used to manage optional Oxs extension packages. Each package is stored in a separate directory under `oommf/app/oxs/contrib/`. These packages can be "installed" and "uninstalled" to and from the `oommf/app/oxs/local/` directory by the **oxspkg** command. The install is a simple copy that does not automatically build the package or link it into the Oxs executable—a separate invocation of **pimake** is needed for that.

Launching

The **oxspkg** launch command is:

```
    tclsh oommf.tcl oxspkg list
or
    tclsh oommf.tcl oxspkg listfiles pkg [pkg ...]
or
    tclsh oommf.tcl oxspkg readme pkg [pkg ...]
or
    tclsh oommf.tcl oxspkg requires pkg [pkg ...]
or
    tclsh oommf.tcl oxspkg install [-v] [-nopatch] pkg [pkg ...]
or
```

```
tclsh oommf.tcl oxspkg uninstall [-f] pkg [pkg ...]
```
or
```
tclsh oommf.tcl oxspkg copyout pkg [pkg ...] destination
```

Glob-style wildcards (*, ?) or the keyword `all` are accepted in package specifications. (If your command shell expands wildcards, as is common on Unix systems, then you may need to escape or quote the wildcards so that they are passed unadulterated to the **oxspkg** program.) The first argument following the **oxspkg** keyword is one of the sub-commands `list`, `listfiles`, `readme`, `requires`, `install`, `uninstall`, or `copyout`:

list Lists all the packges available under `oommf/app/oxs/contrib/`, how many (installable) files are in each package, and the package install status.

listfiles pkg [pkg ...] Lists each of the "installable" files for the selected package. (There may be additional files for the package, e.g. `README` or files with versioning information, included in the `oommf/app/oxs/contrib/<pkg>/` directory. However, those files are ignored by the **oxspkg install** command.)

readme pkg [pkg ...] Prints to `stdout` the contents of the `README` file (if any) for each package.

requires pkg [pkg ...] Lists external libraries required for each package, as declared by any `*.rules` files contained in the package. The user should ensure these libraries are installed on the system and included on the compiler and linker search paths before installing the Oxs extension package. In particular, check the

```
$config SetValue program_compiler_extra_include_dirs ...
$config SetValue program_linker_extra_lib_dirs ...
```

settings in the platform configuration file. There may be additional installation information in the package `README` file.

install [-v] [-nopatch] pkg [pkg ...] Installs (copies) files for the selected package(s) from `oommf/app/oxs/contrib/` to `oommf/app/oxs/local/`. This command **does not** compile the files or link them into the Oxs executable. The user is responsible for making a separate call to **pimake** to build and link the package. If the user is running OOMMF distributed with pre-built binaries, then the user should first `cd` to the OOMMF root directory and run `tclsh oommf.tcl pimake distclean` to delete the distributed binaries before running `tclsh oommf.tcl pimake` to create a new build of OOMMF. This ensures new extension binaries are compatible with the rest of the OOMMF install.

For third-party packages included in OOMMF distributions, the contents of the directory `oommf/app/oxs/contrib/<pkg>/` will mirror some release of the package from the official source of the package. If those files don't build properly with the OOMMF distribution, then a patch file will be included in the parent `oommf/app/oxs/contrib/`

directory. Normally that patch file (if any) is automatically applied as part of the installation procedure. The `-nopatch` option skips the patch step. If there are no patches for a particular package, then the message `No patches found` will be reported during the install process.

The `-v` option requests more verbose output.

uninstall [-f] pkg [pkg ...] Deletes all files in `oommf/app/oxs/local/` associated with the selected package. Here "associated" means a file name match with a file in the package directory `oommf/app/oxs/contrib/<pkg>/`. There is no checking of contents or timestamps between the files. The install directory `oommf/app/oxs/local/<pkg>/` will also be removed if it is empty after deleting the package files or if the `-f` (force) option is specified.

copyout pkg [pkg ...] destination Selects files in the same manner as the `uninstall` command, but rather than deleting the files instead copies the files of each package from `oommf/app/oxs/local/<pkg>/` to `destination/<pkg>/`. The directory `destination` will be created if it does not exist. This command is a development aid for creating patches for packages: Edit the package files in `oommf/app/oxs/local/<pkg>/` as necessary and use **oxspkg copyout** to copy the package files to `destination`. Then, from the `oommf/app/oxs/contrib/` directory, run

```
diff -ru  <pkg>/  destination/<pkg>/  >  <pkg>-<date>.patch
```

for each package to create an updated patch. Here `diff` is the Unix-compatible file comparison utility. If the destination directory is in `oommf/app/oxs/contrib/`, then name it either `tmp`, `new`, or `newcontrib` to prevent **oxspkg** from treating it as a new package directory.

Most of the optional packages controlled by **oxspkg** are from third-party contributors. Some may originate with the OOMMF core development team, but are made optional because they require third-party libraries or are considered too experimental to be included among the standard Oxs extension classes. The various `oommf/app/oxs/contrib/<pkg>/` directories may contain `README` files with additional details.

16.22 Oxs Regression Tests: oxsregression

The **oxsregression** runs a test suite for the Oxs solver. For each test, an instance of **Boxsi** is run and the results are compared against reference results stored in subdirectories under `oommf/app/oxs/regression_tests/`.

Launching

The **oxsregression** launch command is:

```
tclsh oommf.tcl runtests [-autoadd] [-alttestdir] [-cleanup] [-ignoreextra]
   [-keepfail] [-listtests] [-loglevel level] [-noexcludes] [-parallel n]
   [-resultsfile stemname] [-showoutput] [-sigfigs digits] [-threads count]
   [-timeout seconds] [-updaterefdata] [-v] [testa testb ...]
```

where

-autoadd Automatically adds new tests from MIF files found in the examples directory `oommf/app/oxs/examples/`.

-alttestdir Specify an alternative test directory to use in place of the default directory list `oommf/app/oxs/examples/`, `oommf/app/oxs/regression_tests/bug_tests/`, and `oommf/app/oxs/regression_tests/local_tests/`.

-cleanup If **oxsregression** is killed or crashes mid-run, then some temporary result files may be left on disk. This command searches for and offers to delete these stray files.

-ignoreextra Ignore extra columns, if any, in test results as compared to reference results. This is useful in development work when changes to a MIF file introduce additional data table output.

-keepfail Keep results from failed tests. Normally test results are automatically deleted.

-listtests List all selected tests and exit without running any tests.

-loglevel level Controls the amount of log information written to `oxsregression.log` in the regression test directory `oommf/app/oxs/regression_tests/`. The default setting is 0.

-noexcludes Some tests suffer from various numerical problems. These are excluded from testing, unless this option is specified.

-parallel n Run `n` tests concurrently, with default `n=1`. This option is only available when using Tcl 8.6 or later.

-resultsfile stemname Test results are written to temporary files; by default these files have the stem `oxsregression-test-output`. If **oxsregression** is run simultaneously, perhaps on different machines on a shared file system, then overwriting of files from one process can interfere with the processing by another. The `-resultsfile` option can be used to cordon off results between simultaneous runs.

-showoutput If this switch is not specified, then `stdout` and `stderr` output from **Boxsi** is swallowed by **oxsregression**.

-sigfigs digits Number of significant (decimal) digits to use in comparing test to reference results; the default setting is eight.

-threads count Number of threads to run **Boxsi** with. This option is available for threaded builds only. The default is the default thread count for **Boxsi**.

-timeout seconds Maximum number of seconds to wait for one test to finish; any individual test that fails to complete within this time span is summarily terminated. The default time is 150 seconds; use 0 to indicate no timeout.

-updaterefdata For developer use only; this option causes the reference results to be replaced (overwritten) with new test results.

-v Enable verbose output.

testa testb ... Tests to run, with glob-style wildcards (*, ?) accepted. If no tests are specified then all (non-excluded) tests are selected. Subtest selection must be quoted with the test to appear as a single argument, e.g., `"exch6ngbr 1,7,9"`. If no subtests are specified then all subtests are run.

Take notice of the interplay between the `-parallel n` and `-threads count` options. The former is the number of tests run in parallel, and the latter is the number of computation threads active in each test run. The total number of threads active at one time can therefore be as many as `n` × `count`.

16.23 OOMMF and Process ID Information: pidinfo

The **pidinfo** command prints a table mapping OOMMF ID's (OID's) to system process ID's (PID's) and application names.

Launching

The **pidinfo** command line is:

```
tclsh oommf.tcl pidinfo [standard options] [-account name] \
   [-hostport port] [-names] [-noheader] [-pid] [-ports] \
   [-timeout secs] [-wait secs] [-v] [oid ...]
```

where

-account name Specify the account name. The default is the same used by **mmLaunch**: the current user login name, except on Windows 9X, where the dummy account ID "oommf" may be used instead.

-hostport port Use the host server listening on `port`. Default is set by the `Net_Host port` setting in `oommf/config/options.tcl`, or by the environment variable `OOMMF_HOSTPORT` (which, if set, overrides the former). The standard setting is 15136.

-names Display application nicknames, which are used by the MIF 2.1 `Destination` command (Sec. 17.3.2).

-noheader Don't print column headers.

-pid Select processes by system pid rather than OOMMF oid.

-ports Display active server ports for each application.

-timeout secs Maximum time to wait for response from servers, in seconds. Default is five seconds.

-v Display information about the host and account servers.

-wait secs If no match is found, then retry for up to `secs` seconds. Default is zero seconds, i.e., try once.

oid ... List of OOMMF ID's to display information about. Default is all current applications. If the `-pid` option is specified then this selection is by system process ID's rather than OOMMF ID's.

The title bar of running OOMMF applications typically displays the application OID, which are used by OOMMF applications to identify one another. These ID's start at 0 and are incremented each time a newly launched application registers with the account server. The OID's are independent of the operating system PID's. The PID is needed to obtain information, e.g., resource use, about a running process using system utilities. The PID may also be needed to invoke the operating system "kill" facility to terminate a rogue OOMMF application. The **pidinfo** application can be used to correspond OID's or OOMMF application names to PID's for such purposes.

16.24 Platform-Independent Make: pimake

The application **pimake** is used to construct OOMMF end-user executables (and optionally documentation) from the distributed source code. It is similar in operation to the Unix utility program **make**, but it is written entirely in Tcl so that it will run anywhere Tcl is installed. Like **make**, **pimake** controls the building of a *target* from other files. Just as **make** is controlled by rules in files named `Makefile` or `makefile`, **pimake** is controlled by rules in files named `makerules.tcl`. **pimake** generally operates recursively, starting in the current working directory and continuing downward through all subdirectories. (The chief exception to this is the documentation under `oommf/doc`; **pimake** build requests started in the root directory `oommf` ignore the `oommf/doc` subdirectory. Building the OOMMF documentation requires a working installation of LaTeX[4] and either LaTeX2HTML[5] or LaTeXML[6], so OOMMF

[4]https://www.latex-project.org
[5]https://www.latex2html.org
[6]https://dlmf.nist.gov/LaTeXML/

distributions include pre-built versions of the full OOMMF documentation to relieve the end-user the burden of installing those tools. On the other hand, clean requests, including in particular `maintainer-clean`, will recurse into `oommf/doc`.) The OOMMF standard command line option `-cwd` may be used to change the starting directory.

Launching

The **pimake** launch command is:

```
tclsh oommf.tcl pimake [standard options] [-d] [-i] [-k] \
   [-buildgraph] [-color request] [-depgraph] [-dry-run] \
   [-parallel N] [-root] [-v level] [-out file] [-root] [target]
```

where

-d Print verbose information about dependencies.

-i Normally an error halts operation. When `-i` is specified, ignore errors and try to continue updating all dependencies of target.

-k Normally an error halts operation. When `-k` is specified, and an error is encountered, stop processing dependencies which depend on the error, but continue updating other dependencies of target.

-buildgraph Prints the build graph for the requested target, i.e., all the files that would be rebuilt to make the target up-to-date; does not build the target.

-color request Request compiler/linker diagnostics in color, where `request` is one of `always`, `auto`, or `never`. Requires compiler/linker support.

-depgraph Prints the dependency graph for the requested target; does not build the target.

-dry-run Prints the build steps without executing them.

-parallel N Run the build in parallel, with up to `N` execution streams.

-root Change to the OOMMF root directory. Command line options are processed from left to right, so if used in conjunction with the standard option `-cwd`, `-root` should come first.

-v level Sets the level of informational messages reported during processing. Default level is 1, with larger values producing more output.

-out file Write informational output to named file instead of to the standard output.

target The file to build. May also be (and usually is) a symbolic target name. See below for standard symbolic targets. By default, the first target in `makerules.tcl` is built.

There are several targets which may be used as arguments to **pimake** to achieve different tasks. Each target builds in the current directory and all subdirectories. The standard targets are:

upgrade Used immediately after unpacking a distribution, it removes any files which were part of a previous release, but are not part of the unpacked distribution. **WARNING:** This target is currently not maintained. **DO NOT USE.**

all Creates all files created by the `configure` target (see below). Compiles and links all the executables and libraries. Constructs all index files.

configure Creates subdirectories with the same name as the platform type. Constructs a `ocport.h` file which includes C++ header information specific to the platform.

objclean Removes the intermediate object files created by the compile and link steps. Leaves working executables in place. Leaves OOMMF in the state of its distribution with pre-compiled executables.

clean Removes the files removed by the `objclean` target. Also removes the executables and libraries created by the `all` target. Leaves the files generated by the `configure` target.

distclean Removes the files removed by the `clean` target. Also removes all files and directories generated by `configure` target. Leaves only the files which are part of the source code distribution.

maintainer-clean Remove all files which can possibly be generated from other files. If run from the OOMMF root directory then all such files, **including OOMMF documentation files**, will be deleted. Rebuilding may require specialized developer tools, so use with care. This target is intended primarily for developers, not end-users.

help Print a summary of the standard targets.

If `-parallel N` is requested with `N>1` then the various pieces of OOMMF are built in parallel, with up to `N` build streams. If an error occurs in one stream, then no new jobs are started but currently active jobs are allowed to run to completion. Output from the continuing jobs may push error messages off the screen or otherwise make it difficult to discern the cause of error. An immediate relaunch of **pimake** in serial mode (i.e., no `-parallel` option) will usually replicate the error without chatter from other build jobs.

The parallel build process takes into account the dependency information recorded in the various `makerules.tcl` files. However, dependency failures may occur if external applications such as compilers or linkers introduce additional dependencies, for example shared debug symbol tables. If this happens, reset the build with `pimake distclean` and rerun `pimake` in sequential mode.

16.25 Suspending Oxs Processes: suspend

The **suspend** application causes OOMMF Oxs processes (**Oxsii** and **Boxsi**) to write out checkpoint (`.restart`) files and exit. This allows jobs to be resumed at a later time via the **Oxsii**/**Boxsi** restart controls. The **suspend** application is the equivalent of the **Oxsii**/**Boxsi** interactive interface `Suspend` menu option, but allows targeting of multiple Oxs processes.

Launching

The **suspend** command line is:

```
tclsh oommf.tcl suspend [standard options] [-account name] \
   [-hostport port] [-pid] [-q] [-show] [-shownames] [-test] \
   [-timeout secs] oid [...]
```

where

-account name Specify the account name. The default is the same used by **mmLaunch**: the current user login name, except on Windows 9X, where the dummy account ID "oommf" may be used instead.

-hostport port Use the host server listening on `port`. Default is set by the `Net_Host port` setting in `oommf/config/options.tcl`, or by the environment variable `OOMMF_HOSTPORT` (which, if set, overrides the former). The standard setting is 15136.

-pid Select processes by system pid rather than OOMMF oid.

-q Quiet; don't print informational messages.

-show Don't suspend anything, just print matching targets.

-shownames Don't suspend anything, just print nicknames of matching targets, where nicknames are as set by OOMMF command line utility `nickname` or the MIF 2.1 `Destination` command (Sec. 17.3.2).

-test Don't suspend anything, just test that targets are responding.

-timeout secs Maximum time to wait for response from servers, in seconds. Default is five seconds.

oid ... List of one or more oids (OOMMF ID's), application names, nicknames, or keywords "oxsii", "boxsi", or "all". Glob-style wildcards may also be used. This field is required (there are no default suspend targets). If the `-pid` option is specified then numbers are interpreted as referring to system process ID's rather than OOMMF ID's.

The **suspend** command only affects Oxs processes, that is, **Oxsii** and **Boxsi**. The filenames of the resulting checkpoint files (`.restart`) can be found in the corresponding log files (default `oxsii.log` and `boxsi.log`, respectively).

Chapter 17

Problem Specification File Formats (MIF)

Micromagnetic simulations are specified to the OOMMF solvers using the OOMMF *Micromagnetic Input Format* (MIF). There are four distinct versions of this format. The oldest format, version 1.1, is used by the mmSolve 2D solvers (**mmSolve2D** and **batchsolve**) and the **mmProbEd** problem editor. The MIF 2.1 and MIF 2.2 formats are the powerful, native format used by the Oxs 3D solvers. The MIF 1.2 format is a minor modification to the 1.1 format, which can be used as a simple but restricted interface to the Oxs solvers. In all cases values are specified in SI units. A command line utility **mifconvert** is provided to aid in converting MIF 1.1 files to the MIF 2.1 format.

The PyMIF 1.0 format is a variant of the MIF 2.2 format that uses Python as the base language in place of Tcl as used by the MIF 2.x formats.

For all versions it is recommended that MIF files be given names ending with the `.mif` file extension.

17.1 MIF 1.1

The MIF 1.1 format is an older micromagnetic problem specification format used by the mmSolve 2D solvers. It is not compatible with the MIF 2.1 format used by the Oxs 3D solvers. However, the command line tool **mifconvert** may be used as a conversion aid; **mifconvert** is also called automatically by Oxs solvers when a MIF 1.x file is input to them.

A sample MIF 1.1 file is included below. The first line of a MIF file must be of the form "`#` MIF x.y", where x.y represents the format revision number. (The predecessor MIF 1.0 format was not included in any released version of OOMMF.)

After the format identifier line, any line ending in a backslash, '\', is joined to the succeeding line before any other processing is performed. Lines beginning with a '`#`' character are comments and are ignored. Blank lines are also ignored.

All other lines must consist of a *Record Identifier* followed by a parameter list. The Record Identifier is separated from the parameter list by one or more ':' and/or '=' characters.

Whitespace and case is ignored in the Record Identifier field.

The parameter list must be a proper Tcl list. The parameters are parsed (broken into separate elements) following normal Tcl rules; in short, items are separated by whitespace, except as grouped by double quotes and curly braces. Opening braces and quotes must be whitespace separated from the preceding text. The grouping characters are removed during parsing. Any '`#`' character that is found outside of any grouping mechanism is interpreted as a comment start character. The '`#`' and all following characters on that line are interpreted as a comment.

Order of the records in a MIF 1.1 file is unimportant, except as explicitly stated below. If two or more lines contain the same Record Identifier, then the last one takes precedence, with the exception of Field Range records, of which there may be several active. All records are required unless listed as optional. Some of these record types are not supported by **mmProbEd**, however you may modify a MIF 1.1 file using any plain text editor and supply it to **mmSolve2D** using **FileSource**.

For convenience, the Record Identifier tags are organized into several groups; these groups correspond to the top-level buttons presented by **mmProbEd**. We follow this convention below.

17.1.1 Material parameters

- **# Material Name:** This is a convenience entry for **mmProbEd**; inside the MIF 1.1 file it is a comment line. It relates a symbolic name (e.g., Iron) to specific values to the next 4 items. Ignored by solvers.
- **Ms:** Saturation magnetization in A/m.
- **A:** Exchange stiffness in J/m.
- **K1:** Crystalline anisotropy constant in J/m^3. If $K1 > 0$, then the anisotropy axis (or axes) is an easy axis; if $K1 < 0$ then the anisotropy axis is a hard axis.
- **Anisotropy Type:** Crystalline anisotropy type; One of `<uniaxial|cubic>`.
- **Anisotropy Dir1:** Directional cosines of first crystalline anisotropy axis, taken with respect to the coordinate axes (3 numbers). Optional; Default is 1 0 0 (x-axis).
- **Anisotropy Dir2:** Directional cosines of second crystalline anisotropy axis, taken with respect to the coordinate axes (3 numbers). Optional; Default is 0 1 0 (y-axis).

  For uniaxial materials it suffices to specify only Anisotropy Dir1. For cubic materials one must also specify Anisotropy Dir2; the third axis direction will be calculated as the cross product of the first two. The anisotropy directions will be automatically normalized if necessary, so for example 1 1 1 is valid input (it will be modified to .5774 .5774 .5774). For cubic materials, Dir2 will be adjusted to be perpendicular to Dir1 (by subtracting out the component parallel to Dir1).

- **Anisotropy Init:** Method to use to set up directions of anisotropy axes, as a function of spatial location; This is a generalization of the Anisotropy Dir1/2 records. The value for this record should be one of `<Constant|UniformXY|UniformS2>`. `Constant` uses the values specified for Anisotropy Dir1 and Dir2, with no dispersion. `UniformXY` ignores the values given for Anisotropy Dir1 and Dir2, and randomly varies the anisotropy directions uniformly in the xy-plane. `UniformS2` is similar, but randomly varies the anisotropy directions uniformly on the unit sphere (S^2). This record is optional; the default value is `Constant`.

- **Edge K1:** Anisotropy constant similar to crystalline anisotropy constant K1 described above, but applied only along the edge surface of the part. This is a uniaxial anisotropy, directed along the normal to the boundary surface. Units are J/m^3, with positive values making the surface normal an easy axis, and negative values making the surface an easy plane. The default value for Edge K1 is 0, which disables the term.

- **Do Precess:** If 1, then enable the precession term in the Landau-Lifshitz ODE. If 0, then do pure damping only. (Optional; default value is 1.)

- **Gyratio:** The Landau-Lifshitz gyromagnetic ratio, in m/(A.s). This is optional, with default value of 2.21×10^5. See the discussion of the Landau-Lifshitz ODE under the Damp Coef record identifier description.

- **Damp Coef:** The ODE solver in OOMMF integrates the Landau-Lifshitz equation [13, 18], written as

$$\frac{d\mathbf{M}}{dt} = -|\bar{\gamma}|\,\mathbf{M} \times \mathbf{H}_{\text{eff}} - \frac{|\bar{\gamma}|\alpha}{M_s}\,\mathbf{M} \times (\mathbf{M} \times \mathbf{H}_{\text{eff}})\,,$$

  where

  $\bar{\gamma}$ is the Landau-Lifshitz gyromagnetic ratio (m/(A·s)),
  α is the damping coefficient (dimensionless).

  (Compare to (7.2), page 79.) Here α is specified by the "Damp Coef" entry in the MIF 1.1 file. If not specified, a default value of 0.5 is used, which allows the solver to converge in a reasonable number of iterations. Physical materials will typically have a damping coefficient in the range 0.004 to 0.15. The 2D solver engine **mmSolve** requires a non-zero damping coefficient.

17.1.2 Demag specification

- **Demag Type:** Specify algorithm and demagnetization kernel used to calculate self-magnetostatic (demagnetization) field. Must be one of

  – **ConstMag:** Calculates the *average* field in each cell under the assumption that the magnetization is constant in each cell, using formulae from [2, 25, 11]. (The other demag options calculate the field at the center of each cell.)

- **3dSlab:** Calculate the in-plane field components using offset blocks of constant (volume) charge. Details are given in [3]. Field components parallel to the z-axis are calculated using squares of constant (surface) charge on the upper and lower surfaces of the sample.
- **3dCharge:** Calculate the in-plane field component using rectangles of constant (surface) charge on each cell. This is equivalent to assuming constant magnetization in each cell. The z-components of the field are calculated in the same manner as for the 3dSlab approach.
- **FastPipe:** Algorithm suitable for simulations that have infinite extent in the z-direction. This is a 2D version of the 3dSlab algorithm.
- **None:** No demagnetization. Fastest but least accurate method. `:-}`

All of these algorithms except FastPipe and None require that the Part Thickness (cf. the Part Geometry section) be set. Fast Fourier Transform (FFT) techniques are used to accelerate the calculations.

17.1.3 Part geometry

- **Part Width:** Nominal part width (x-dimension) in meters. Should be an integral multiple of Cell Size.
- **Part Height:** Nominal part height (y-dimension) in meters. Should be an integral multiple of Cell Size.
- **Part Thickness:** Part thickness (z-dimension) in meters. Required for all demag types except FastPipe and None.
- **Cell Size:** In-plane (xy-plane) edge dimension of base calculation cell. This cell is a rectangular brick, with square in-plane cross-section and thickness given by Part Thickness. N.B.: Part Width and Part Height should be integral multiples of Cell Size. Part Width and Part Height will be automatically adjusted slightly (up to 0.01%) to meet this condition (affecting a small change to the problem), but if the required adjustment is too large then the problem specification is considered to be invalid, and the solver will signal an error.
- **Part Shape:** Optional. Part shape in the xy-plane; must be one of the following:
  - **Rectangle**
    The sample fills the area specified by Part Width and Part Height. (Default.)
  - **Ellipse**
    The sample (or the magnetically active portion thereof) is an ellipse inscribed into the rectangular area specified by Part Width and Part Height.

– **Ellipsoid**
Similar to the Ellipse shape, but the part thickness is varied to simulate an ellipsoid, with axis lengths of Part Width, Part Height and Part Thickness.

– **Oval r**
Shape is a rounded rectangle, where each corner is replaced by a quarter circle with radius r, where $0 \leq r \leq 1$ is relative to the half-width of the rectangle.

– **Pyramid overhang**
Shape is a truncated pyramid, with ramp transition base width (overhang) specified in meters.

– **Mask filename**
Shape and thickness are determined by a bitmap file, the name of which is specified as the second parameter. The overall size of the simulation is still determined by Part Width and Part Height (above); the bitmap is spatially scaled to fit those dimensions. Note that this scaling will not be square if the aspect ratio of the part is different from the aspect ratio of the bitmap.

The given filename must be accessible to the solver application. At present the bitmap file must be in either the PPM (portable pixmap), GIF, or BMP formats. (Formats other than the PPM P3 (text) format may be handled by spawning an **any2ppm** subprocess.)

White areas of the bitmap are interpreted as being non-magnetic (or having 0 thickness); all other areas are assumed to be composed of the material specified in the "Material Parameters" section. Thickness is determined by the relative darkness of the pixels in the bitmap. Black pixels are given full nominal thickness (specified by the "Part Thickness" parameter above), and gray pixels are linearly mapped to a thickness between the nominal thickness and 0. In general, bitmap pixel values are converted to a thickness relative to the nominal thickness by the formula 1-(R+G+B)/(3M), where R, G and B are the magnitudes of the red, green and blue components, respectively, and M is the maximum allowed component magnitude. For example, black has R=G=B=0, so the relative thickness is 1, and white has R=G=B=M, so the relative thickness is 0.

The code does not perform a complete 3D evaluation of thickness effects. Instead, the approximation discussed in [26] is implemented.

17.1.4 Initial magnetization

- **Init Mag:** Name of routine to use to initialize the simulation magnetization directions (as a function of position), and routine parameters, if any. Optional, with default Random. The list of routines is long, and it is easy to add new ones. See the file `oommf/app/mmsolve/maginit.cc` for details. A few of the more useful routines are:

– **Random**
Random directions on the unit sphere. This is somewhat like a quenched thermal demagnetized state.

– **Uniform θ ϕ**
Uniform magnetization in the direction indicated by the two additional parameters, θ and ϕ, where the first is the angle from the z-axis (in degrees), and the second is the angle from the x-axis (in degrees) of the projection onto the xy-plane.

– **Vortex**
Fits an idealized vortex about the center of the sample.

– **avfFile filename**
The second parameter specifies an OVF/VIO (i.e., "any" vector field) file to use to initialize the magnetization. The grid in the input file will be scaled as necessary to fit the grid in the current simulation. The file must be accessible to the intended solver application.

17.1.5 Experiment parameters

The following records specify the applied field schedule:

- **Field Range:** Specifies a range of applied fields that are stepped though in a linear manner. The parameter list should be 7 numbers, followed by optional control point (stopping criteria) specifications. The 7 required fields are the begin field Bx By Bz in Tesla, the end field Bx By Bz in Tesla, and an integer number of steps (intervals) to take between the begin and end fields (inclusive). Use as many Field Range records as necessary—they will be stepped through in order of appearance. If the step count is 0, then the end field is ignored and only the begin field is applied. If the step count is larger than 0, and the begin field is the same as the last field from the previous range, then the begin field is not repeated.

  The optional control point specs determine the conditions that cause the applied field to be stepped, or more precisely, end the simulation of the magnetization evolution for the current applied field. The control point specs are specified as *-type value* pairs. There are 3 recognized control point types: **-torque**, **-time**, and **-iteration**. If a -torque pair is given, then the simulation at the current applied field is ended when $\|\mathbf{m} \times \mathbf{h}\|$ (i.e., $\|\mathbf{M} \times \mathbf{H}\|/M_s^2$) at all spins in the simulation is smaller than the specified -torque value (dimensionless). If a -time pair is given, then the simulation at the current field is ended when the elapsed simulation time *for the current field step* reaches the specified -time value (in seconds). Similarly, an -iteration pair steps the applied field when the iteration count for the current field step reaches the -iteration value. If multiple control point specs are given, then the applied field is advanced when any one of the specs is met. If no control point specs are given on a range line, then the **Default Control Point Spec** is used.

  For example, consider the following Field Range line:

```
Field Range: 0 0 0 .05 0 0  5  -torque 1e-5 -time 1e-9
```

This specifies 6 applied field values, (0,0,0), (0.01,0,0), (0.02,0,0), ..., (0.05,0,0) (in Tesla), with the advancement from one to the next occurring whenever $\|\mathbf{m} \times \mathbf{h}\|$ is smaller than 1e-5 for all spins, or when 1 nanosecond (simulation time) has elapsed at the current field. (If -torque was not specified, then the applied field would be stepped at 1, 2, 3 4 and 5 ns in simulation time.)

The Field Range record is optional, with a default value of 0 0 0 0 0 0 0.

- **Default Control Point Spec:** List of control point *-type value* pairs to use as stepping criteria for any field range with no control point specs. This is a generalization of and replacement for the *Converge* $|mxh|$ *Value* record. Optional, with default "`-torque 1e-5`."

- **Field Type:** Applied (external) field routine and parameters, if any. This is optional, with default Uniform. At most one record of this type is allowed, but the Multi type may be used to apply a collection of fields. The nominal applied field (NAF) is stepped through the Field Ranges described above, and is made available to the applied field routines which use or ignore it as appropriate.

  The following Field Type routines are available:

  - **Uniform**
    Applied field is uniform with value specified by the NAF.
  - **Ribbon relcharge x0 y0 x1 y1 height**
    Charge "Ribbon," lying perpendicular to the xy-plane. Here relcharge is the charge strength relative to Ms, and (x0,y0), (x1,y1) are the endpoints of the ribbon (in meters). The ribbon extends height/2 above and below the calculation plane. This routine ignores the NAF.
  - **Tie rfx rfy rfz x0 y0 x1 y1 ribwidth**
    The points (x0,y0) and (x1,y1) define (in meters) the endpoints of the center spine of a rectangular ribbon of width ribwidth lying in the xy-plane. The cells with sample point inside this rectangle see an applied field of (rfx,rfy,rfz), in units relative to Ms. (If the field is large, then the magnetizations in the rectangle will be "tied" to the direction of that field.) This routine ignores the NAF.
  - **OneFile filename multiplier**
    Read B field (in Tesla) in from a file. Each value in the file is multiplied by the "multiplier" value on input. This makes it simple to reverse field direction (use -1 for the multiplier), or to convert H fields to B fields (use 1.256637e-6). The input file may be any of the vector field file types recognized by **mmDisp**. The input dimensions will be scaled as necessary to fit the simulation grid, with zeroth order interpolation as necessary. This routine ignores the NAF.

– **FileSeq filename procname multiplier**
This is a generalization of the OneFile routine that reads in fields from a sequence of files. Here "filename" is the name of a file containing Tcl code to be sourced during problem initialization, and "procname" is the name of a Tcl procedure defined in filename, which takes the nominal B field components (in Tesla) and field step count values as imports (4 values total), and returns the name of the vector field file that should be used as the applied B field for that field step. The B field units in the vector field file should be Tesla.

– **Multi routinecount **
**param1count name1 param1 param2 . . . **
**param2count name2 param1 param2 . . . **
. . .
Allows a conglomeration of several field type routines. All entries must be on the same logical line, i.e., end physical lines with '\' continuation characters as necessary. Here routinecount is the number of routines, and param1count is the number parameters (including name1) needed by the first routine, etc.

Note that all lengths are in meters. The coordinates in the simulation lie in the first octant, running from (0,0,0) to (Part Width, Part Height, Part Thickness).

17.1.6 Output specification

- **Base Output Filename:** Default base name used to construct output filenames.
- **Magnetization Output Format:** Format to use in the OVF data block for exported magnetization files. Should be one of "binary 4" (default), "binary 8", or "text *format-spec*", where *format-spec* is a C `printf`-style format code, such as "`%# .17g`". Optional.
- **Total Field Output Format:** Analogous to the Magnetization Output Format, but for total field output files. Optional, with default "binary 4".
- **Data Table Output Format:** Format to use when producing data table style scalar output, such as that sent to **mmDataTable**, **mmGraph**, and **mmArchive**. Should specify a C `printf`-style format code, such as the default "`%.16g`". Optional.

17.1.7 Miscellaneous

- **Converge |mxh| Value:** Nominal value to use as a stopping criterion: When $\|\mathbf{m}\times\mathbf{h}\|$ (i.e., $\|\mathbf{M}\times\mathbf{H}\|/M_s^2$) at all spins in the simulation is smaller than this value, it is assumed that a relaxed (equilibrium) state has been reached for the current applied field. This is a dimensionless value.
NOTE: This Record Identifier is deprecated. Use *Default Control Point Spec* instead.

- **Randomizer Seed:** Value with which to seed random number generator. Optional. Default value is 0, which uses the system clock to generate a semi-random seed.
- **Max Time Step:** Limit the maximum ODE step size to no larger than this amount, in seconds. Optional.
- **Min Time Step:** Limit the minimum ODE step size to no less than this amount, in seconds. Optional.
- **User Comment:** Free-form comment string that may be used for problem identification. Optional.

```
# MIF 1.1
#
# Example from the OOMMF User's Guide.
#
# All units are SI.
#
################## MATERIAL PARAMETERS ######################
Ms:  800e3               # Saturation magnetization in A/m.
A:   13e-12              # Exchange stiffness in J/m.
K1:  0.5e3               # Anisotropy constant in J/m^3.
Anisotropy Type: uniaxial # One of <uniaxial|cubic>.
Anisotropy Dir1: 1 0 0    # Directional cosines wrt to
                          # coordinate axes

################## DEMAG SPECIFICATION ######################
Demag Type: ConstMag # One of <ConstMag|3dSlab|2dSlab
                     #         |3dCharge|FastPipe|None>.

#################### PART GEOMETRY ##########################
Part Width:     0.25e-6    # Nominal part width in m
Part Height:    1.0e-6     # Nominal part height in m
Part Thickness: 1e-9       # Part thickness in m.
Cell Size:      7.8125e-9  # Cell size in m.
#Part Shape:    # One of <Rectangle|Ellipse|Oval|Mask>.
                # Optional.

################ INITIAL MAGNETIZATION ######################
Init Mag: Uniform 90 45 # Initial magnetization routine
                        # and parameters

################ EXPERIMENT PARAMETERS ######################
```

```
# Field Range:  Start_field  Stop_field  Steps
Field Range: -.05 -.01 0.  .05  .01 0. 100
Field Range:  .05  .01 0. -.05 -.01 0. 100
Field Type: Multi 4 \
 7 Ribbon 1 0 1.0e-6 0.25e-6 1.0e-6 1e-9 \
 7 Ribbon 1 0 0      0.25e-6 0      1e-9 \
 9 Tie 100 0 0 0.12e-6 0.5e-6 0.13e-6 0.5e-6 8e-9 \
 1 Uniform
# The above positions ribbons of positive charge along the
# upper and lower edges with strength Ms, applies a large
# (100 Ms) field to the center cells, and also applies a
# uniform field across the sample stepped from
# (-.05,-.01,0.) to (.05,.01,0.) (Tesla), and back, in
# approximately 0.001 T steps.

Default Control Point Spec: -torque 1e-6
# Assume equilibrium has been reached, and step the applied
# field, when the reduced torque |mxh| drops below 1e-6.

################ OUTPUT SPECIFICATIONS ####################
Base Output Filename: samplerun
Magnetization Output Format: binary 8 # Save magnetization
# states in binary format with full (8-byte) precision.

#################### MISCELLANEOUS ########################
Randomizer Seed: 1   # Random number generator seed.
User Comment: Example MIF 1.1 file, with lots of comments.
```

Figure 17.1: Example MIF 1.1 file. (description)

17.2 MIF 1.2

The MIF 1.2 format is a minor modification to the MIF 1.1 format, which supports limited 3D problem specification. It can be read by the Oxs 3D solvers, and, with certain restrictions, by the mmSolve 2D solvers as well. The **mmProbEd** problem editor can read and write this format. The **mifconvert** command line tool can be used to convert between the MIF 1.1 and MIF 1.2 formats, and to convert from the MIF 1.x formats to the Oxs MIF 2.1 format. **mifconvert** is also called automatically by Oxs solvers when a MIF 1.x file is input to them, so questions about the details of Oxs interpretation of MIF 1.x files can be answered by running **mifconvert** separately on the input MIF 1.x file.

There are four differences between the MIF 1.1 and 1.2 formats. In the MIF 1.2 format:

1. The first line reads: `# MIF 1.2`
2. The `CellSize` record takes three parameters: x-dimension, y-dimension, and z-dimension (in meters).
3. The `3dSlab`, `2dSlab`, `3dCharge`, and `FastPipe` parameters of the `DemagType` record are deprecated.
4. The new record `SolverType` is introduced. Valid values are `Euler`, `RK2`, `RK4`, `RKF54`, and `CG`, requesting a first order Euler, second order Runge-Kutta, fourth order Runge-Kutta, fifth(+fourth) order Runge-Kutta-Fehlberg, and Conjugate-Gradient solvers, respectively. This record is optional, with default value of `RKF54`.

If the `CellSize` record has only one parameter, then it is interpreted in the same manner as in the MIF 1.1 format, i.e., the parameter is taken as the x- and y-dimension of the computation cell, and the z-dimension is set to the part thickness.

The mmSolve 2D solvers will accept files in the MIF 1.2 format provided the `CellSize` record meets the restrictions of those solvers, namely, the x- and y-dimensions must be the same, and the z-dimension must equal the part thickness. The `SolverType` record, if any, is ignored.

The Oxs 3D solvers will read files in the MIF 1.2 format, but support only the `ConstMag` and `None` demagnetization kernels. All other `DemagType` records are silently converted to `ConstMag`. The `SolverType` record is converted into the appropriate solver+driver pair.

17.3 MIF 2.1

The MIF 2.x format was introduced with the Oxs 3D solver. It is *not* backwards compatible with the MIF 1.x formats, however a conversion utility, **mifconvert**, is available to aid in converting MIF 1.x files to the MIF 2.1 format.

17.3.1 MIF 2.1 File Overview

The first line of a MIF file must be of the form "`#` MIF x.y", where x.y represents the format revision number, here 2.1. Unlike MIF 1.1 files, the structure of MIF 2.1 files are governed by the requirement that they be valid Tcl scripts, albeit with a handful of extensions. These files are evaluated inside a Tcl interpreter, which may be a "safe" interpreter, i.e., one in which disk and other system access is disabled. (Refer to the documentation of the Tcl `interp` command for details on safe interpreters.) The security level is controlled by the `MIFinterp` option in the `options.tcl` customization file. The default setting is

```
Oc_Option Add Oxs* MIFinterp safety custom
```

which enables all the Tcl interpreter extensions described in MIF 2.1 Extension Commands below, but otherwise provides a standard Tcl safe interpreter. The keyword `custom` above may be replaced with either `safe` or `unsafe`. The `safe` selection is similar to `custom`, except that the `DateSort`, `ReadFile` and `RGlob` extensions are not provided, thereby eliminating all disk access at the MIF script level. At the other extreme, choosing `unsafe` provides an unrestricted Tcl interpreter. This option should be used with caution, especially if you are working with MIF files from an unknown or untrusted source.

After the first line, there is considerable flexibility in the layout of the file. Generally near the top of the file one places any `OOMMFRootDir`, `Parameter`, and `RandomSeed` statements, as desired.

This is followed by the major content of the file, the various `Specify` blocks, which initialize `Oxs_Ext` objects:

- Atlas (one or more)
- Mesh (one)
- Energy terms (one or more)
- Evolver (one)
- Driver(one)

The `Specify` blocks are processed in order, so any block that is referred to by another block must occur earlier in the file. For that reason, the main atlas object, which is referenced in many other `Specify` blocks, is generally listed first. The atlas object defines the spatial extent of the simulation, and optionally declares subregions inside the simulation volume.

The mesh object details the spatial discretization of the simulation volume. Conventionally its `Specify` block follows the `Specify` block for the main atlas object; the mesh is referenced by the driver, so in any event the mesh `Specify` block needs to precede the driver `Specify` block.

The energy terms describe the typical micromagnetic energies and fields that determine the evolution of the simulation, such as exchange energy, magnetostatic fields, and anisotropy energies. Material parameters, such as the anisotropy constant `K1` and the exchange constant `A`, are generally specified inside the `Specify` block for the relevant energy, e.g., `Oxs_UniaxialAnisotropy` or `Oxs_Exchange6Ngbr`. The exception to this is saturation magnetization, `Ms`, which is declared in the driver `Specify` block. The initial magnetization, `m0`, is also specified in the driver `Specify` block. In many cases these material parameters may be varied spatially by defining them using scalar or vector field objects. As discussed in the section on Specify Conventions, auxiliary objects such as scalar and vector fields can be defined either inline (i.e., inside the body of the referencing `Specify` block) or in their own, standalone top-level `Specify` blocks. In the latter case, the auxiliary `Specify` blocks must precede the referencing `Specify` blocks in the MIF 2.1 file.

Given the energies and fields, the evolver and driver form a matched pair that advance the magnetic state from an initial configuration, obeying either Landau-Lifshitz-Gilbert (LLG) dynamics or direct energy minimization. For energy minimization studies, the driver must be an `Oxs_MinDriver` object, and the evolver must be a minimization evolver. At the time of this writing, the only minimization evolver packaged with OOMMF is the `Oxs_CGEvolve` conjugate-gradient evolver. For time-evolution (LLG) simulations, the driver must be an `Oxs_TimeDriver` object, and the evolver must be a time evolver, such as `Oxs_RungeKuttaEvolve`. The evolver to be used is cited inside the driver `Specify` block, so the evolver must precede the driver in the MIF 2.1 file. As noted above, the pointwise saturation magnetization `Ms` and initial magnetization configuration `m0` are declared inside the driver `Specify` block as well.

The pre-specified outputs, indicated by zero or more `Destination` and `Schedule` commands, are conventionally placed after the `Specify` blocks. Output selection can also be modified at runtime using the **Oxsii** or **Boxsi** interactive interfaces.

Auxiliary Tcl procs may be placed anywhere in the file, but commonly either near their point of use or else at the bottom of the MIF file. If a proc is only referenced from inside `Specify` blocks, then it can be placed anywhere in the file. On the other hand, if a proc is used at the top level of the MIF file, for example to dynamically create part of the problem specification "on-the-fly," then it must be defined before it is used, in the normal Tcl manner.

A sample MIF 2.1 file is included below. More details on the individual `Oxs_Ext` objects can be found in the Standard Oxs_Ext Child Classes section of the Oxs documentation.

17.3.2 MIF 2.1 Extension Commands

In addition to the standard Tcl commands (modulo the safe Tcl restrictions outlined above), a number of additional commands are available in MIF 2.1 files: `Specify`, `ClearSpec`, `DateSort`, `Destination`, `GetStateData`, `Ignore`, `OOMMFRootDir`, `Parameter`, `Random`, `NormalRV`, `RandomSeed`, `Report`, `ReadFile`, `RGlob`, and `Schedule`.

Specify An Oxs simulation is built as a collection of `Oxs_Ext` (Oxs Extension) objects. In general, `Oxs_Ext` objects are specified and initialized in the input MIF 2.1 file using the `Specify` command, making Specify blocks the primary component of the problem definition. The `Specify` command takes two arguments: the name of the `Oxs_Ext` object to create, and an *initialization string* that is passed to the `Oxs_Ext` object during its construction. The objects are created in the order in which they appear in the MIF file. Order is important in some cases; for example, if one `Oxs_Ext` object refers to another in its initialization string, then the referred to object must precede the referrer in the MIF file.

Here is a simple Specify block:

```
Specify Oxs_EulerEvolve:foo {
  alpha 0.5
  start_dm 0.01
```

```
}
```

The name of the new `Oxs_Ext` object is "Oxs_EulerEvolve:foo." The first part of this name, up to the colon, is the C++ class name of the object. This must be a child of the `Oxs_Ext` class. Here, `Oxs_EulerEvolve` is a class that integrates the Landau-Lifshitz ODE using a simple forward Euler method. The second part of the name, i.e., the part following the colon, is the *instance name* for this particular instance of the object. In general, it is possible to have multiple instances of an `Oxs_Ext` child class in a simulation, but each instance must have a unique name. These names are used for identification by output routines, and to allow one Specify block to refer to another Specify block appearing earlier in the MIF file. If the second part of the name is not given, then as a default the empty string is appended. For example, if instead of "Oxs_EulerEvolve:foo" above the name was specified as just "Oxs_EulerEvolve", then the effective full name of the created object would be "Oxs_EulerEvolve:".

The second argument to the `Specify` command, here everything between the curly braces, is a string that is interpreted by the new `Oxs_Ext` (child) object in its constructor. The format of this string is up to the designer of the child class, but there are a number of conventions that designers are encouraged to follow. These conventions are described in Specify Conventions below.

ClearSpec This command is used to disable one or all preceding `Specify` commands. In particular, one could use `ClearSpec` to nullify a Specify block from a base MIF file that was imported using the `ReadFile` command. Sample usage is

```
ClearSpec Oxs_EulerEvolve:foo
```

where the parameter is the full name (here `Oxs_EulerEvolve:foo`) of the Specify block to remove. If no parameter is given, then all preceding Specify blocks are removed.

DateSort Given a list of filenames, returns the list sorted by modification time, with oldest first. The files must all exist. Sample usage is

```
set last_omf [lindex [DateSort [RGlob *omf]] end]
```

Here `last_omf` will be set to the `*.omf` file in the current directory with the most recent modification time.

Destination The format for the `Destination` command is

```
Destination <desttag> <appname> [new] [output_options]
```

This command associates a symbolic *desttag* with an application. The tags are used by the `Schedule` command (see below) to refer to specific application instances. The *appname* may either be an OOMMF application name, e.g., mmDisp, or else a specific application instance in the form application:nickname, such as mmDisp:Spock. In the

first case, the tag is associated with the running instance of the requested application (here `mmDisp`) with the lowest OOMMF ID (OID) that has not yet been associated with another tag. If no running application can be found that meets these criteria, then a new instance of the application is launched.

If the *appname* refers to a specific application instance, then the tag is associated with the running instance of the application (say **mmDisp**) that has been assigned the specified nickname. Name matching is case insensitive. If there is no running copy of the application meeting this condition, then a new instance of the application is launched and it is assigned the specified nickname. The OOMMF account service directory guarantees that there is never more than one instance of an application with a given nickname. However, as with the object name in the `Specify` command, instances of two different applications, e.g., **mmDisp** and **mmGraph**, are allowed to share nicknames, because their full instance names, say mmDisp:Spock and mmGraph:Spock, are unique.

The `Destination` commands are processed in the order in which they appear in the MIF file. No *desttag* may appear in more than one `Destination` command, and no two destination tags may refer to the same application instance. To ensure the latter, the user is advised to place all `Destination` commands referring to specific instances (e.g., mmDisp:Spock) before any `Destination` commands using generic application references (e.g., mmDisp). Otherwise a generic reference might be associated to a running application holding a nickname that is referenced by a later `Destination` command.

The tag association by the `Destination` command is only known to the solver reading the MIF file. In contrast, assigned instance nicknames are recognized across applications. In particular, multiple solvers may reference the same running application by nickname. For example, several sequential solver runs could send stage output to the same **mmGraph** widget, to build up overlapping hysteresis loops.

If the optional `new` keyword is included in the `Destination` command, then a fresh copy of the requested application is always launched for association with the given tag. The `new` option can be safely used with any generic application reference, but caution must be taken when using this option with specific instance references, because an error is raised if the requested nickname is already in use.

The trailing `options` are label + value pairs corresponding to the MIF 2.2 `SetOptions` command, but with each label preceded by a hyphen. For example

```
Destination archiveA mmArchive -scalar_output_format %20.12e
```

Any such options override the current `SetOptions` settings for this destination.

GetStateData The `GetStateData` command retrieves data attached to a specific magnetization state:

```
GetStateData [-glob|-exact|-regexp] [-pairs] [--] <state_id> \
             [pattern ...]
```

The data associated with a state are stored as key-value pairs. If no patterns are specified then `GetStateData` returns the list of keys available for the given state. If one or more patterns are specified, then all values with keys matching some pattern are collected. If the `-pairs` option is specified then the return is an even length list of keys and values interleaved. If `-pairs` is not specified then the return is a list of just the values. The values are returned in key-match order. Key matching style is controlled by the first slate of options, with default being glob. Two hyphens may be used to denote the end of options.

The `state_id` is a positive integer identifying the state. This is generally obtained via a `script_args` option in the `Specify` block of a conforming `Oxs_Ext` object. For example,

```
proc SpinMag { stage_time state_id } {
   lassign [GetStateData $state_id *:Mx *:My *:Mz] Mx My Mz
   ...
}
Specify Oxs_ScriptUZeeman {
  script SpinMag
  script_args {stage_time base_state_id}
}
```

Typically two states may be accessed this way: the step base state and a candidate (test) state. The former, accessed as `base_state_id`, corresponds to the last valid, accepted magnetization state. The latter, accessed as `current_state_id`, is the latest working state from the evolver object. In some cases these two states may coincide.

The keys associated with a state vary with the details of the simulation. The following keys are always available:

`state_id`	`previous_state_id`	`iteration_count`
`stage_number`	`stage_iteration_count`	`stage_start_time`
`stage_elapsed_time`	`total_elapsed_time`	`last_timestep`
`step_done`	`stage_done`	`run_done`
`max_absMs`		

Times are in seconds, the `step`/`stage`/`run_done` values are one of 1 (done), 0 (not done) or -1 (not yet determined), and `max_absMs` is in A/m.

Additional key-value pairs may be attached to a state by `Oxs_Ext` objects. For example, `Oxs_RungeKuttaEvolve` adds the average magnetization x-component under the key name `Oxs_RungeKuttaEvolve:<instance_name>:Mx`. (Here `<instance_name>` is

the instance name of the object; this is typically an empty string or something like "evolver".) See the documentation for the various `Oxs_Ext` objects for details.

Moreover, the keys available for a state may depend on the simulation status or processing step. In particular, the current state indexed by `current_state_id` typically only has the default keys from the table above available. For this reason, and additionally because the `current_state_id` is only a test state that may be rejected, user scripts should generally avoid using data tied to the current state in favor of data collected from the base state. Likewise, the available keys may be different for a state (even a base state) marking the start of a new stage as compared to states arising inside a stage. In case of problems, a `Report` command inside a script proc can be used to dump state information to the Oxsii console, for example,

```
proc SpinMag { stage_time state_id } {
   Report "State $state_id, Keys: [GetStateData $base_state_id]"
   lassign [GetStateData $state_id *:Mx *:My *:Mz] Mx My Mz
   ...
}
```

or

```
proc SpinMag { stage_time state_id } {
   set report {}
   foreach {key value} [GetStateData $state_id *] {
      append report [format "%42s : $value\n" $key]
   }
   Report "--- State data ---\n$report"
   lassign [GetStateData $state_id *:Mx *:My *:Mz] Mx My Mz
   ...
}
```

For example use of `GetStateData`, see the sample files `spinmag.mif` and `spinmag2.mif` in the directory `oommf/app/oxs/examples/`.

Ignore The `Ignore` command takes an arbitrary number of arguments, which are thrown away without being interpreted. The primary use of `Ignore` is to temporarily "comment out" (i.e., disable) Specify blocks.

OOMMFRootDir This command takes no arguments, and returns the full directory path of the OOMMF root directory. This is useful in conjunction with the `ReadFile` command for locating files within the OOMMF hierarchy, and can also be used to place output files. File paths must be created directly since the Tcl `file` command is not accessible inside safe interpreters. For example

```
set outfile [OOMMFRootDir]/data/myoutput
```

In this context one should always use Tcl path conventions. In particular, use forward slashes, "/", to separate directories.

Parameter The Oxs interfaces (Oxsii and Boxsi) allow specified variables in the MIF file to be set from the command line via the `-parameters` option. This functionality is enabled inside the MIF file via the `Parameter` command:

`Parameter` *varname optional_default_value*

Here *varname* is the name of a variable that may be set from the command line. If it is not set on the command line then the variable is set to the optional default value, if any, or otherwise an error is raised. An error is also raised if a variable set on the command line does not have a corresponding `Parameter` command in the MIF file. See also the notes on variable substitution below.

Random Returns a pseudo-random number in the interval $[0, 1]$, using a C-library random number generator. This random number generator is specified by the `OMF_RANDOM` macro in the `ocport.h` file found in the system-specific subdirectory of `oommf/pkg/oc/`. The standard Tcl `expr rand()` command is also available.

NormalRV Returns a pseudo-random number with standard normal distribution (mean 0, variance 1) using the C++11 `std::normal_distribution` library.

RandomSeed Initializes both the Tcl and the C/C++-library random number generators. One integer parameter may be specified to be used as the seed. If no parameter is given, or if `RandomSeed` is not called, then a seed is drawn from the system clock.

Report Intended primarily as a MIF debugging aid, `Report` takes one string argument that is printed to the solver interface console and the Oxs log file. It is essentially a replacement for the standard Tcl `puts` command, which is not available in safe interpreters.

ReadFile The Tcl `read` command is absent from safe interpreters. The `ReadFile` command is introduced as a replacement available in "custom" and "unsafe" interpreters. `ReadFile` takes two arguments, the file to be read and an optional translation specification. The file may either be specified with an absolute path, i.e., one including all its directory components, or with a relative path interpreted with respect to the directory containing the MIF file. The `OOMMFRootDir` command can be used to advantage to locate files in other parts of the OOMMF directory tree.

The translation specification should be one of `binary`, `auto` (the default), `image` or `floatimage`. The first two translation modes provide the functionality of the `-translation` option of the Tcl `fconfigure` command. Refer to the Tcl documentation for details. Specifying `image` causes the input file to be read as an image file. The file will be read directly if it in the PPM P3 (text), PPM P6 (binary), or uncompressed

BMP formats; otherwise it is filtered through the OOMMF **any2ppm** program. (Note that **any2ppm** requires Tk, and Tk requires a display.) The input file is converted into a string that mimics a PPM P3 text file, minus the leading "P3". In particular, after conversion the first 3 whitespace separated values are image width, height and maxvalue, followed by an array of 3 × width × height values, where each triplet corresponds to the red, green and blue components of an image pixel, sequenced in normal English reading order. Each component is in the range $[0, maxvalue]$. This output contains no comments, and may be treated directly as a Tcl list. The `floatimage` option is very similar to the `image` option, except that the third value (i.e., maxvalue) in the resulting string is always "1", and the succeeding red, green and blue values are floating point values in the range $[0, 1]$.

In all cases, the return value from the `ReadFile` command is a string corresponding to the contents of the (possibly translated) file. For example,

```
eval [ReadFile extra_mif_commands.tcl]
```

can be used to embed a separate Tcl file into a MIF 2.1 file.

Here's a more complicated example that uses a color image file to create a vector field:

```
set colorimage [ReadFile mirror.ppm floatimage]
set imagewidth [lindex $colorimage 0]
set imageheight [lindex $colorimage 1]
set imagedepth [lindex $colorimage 2] ;# Depth value should be 1
if {$imagedepth != 1} {
   Report "ReadFile returned unexpected list value."
}

proc ColorField { x y z } {
     global colorimage imagewidth imageheight
     set i [expr {int(floor(double($x)*$imagewidth))}]
     if {$i>=$imagewidth} {set i [expr {$imagewidth-1}]}
     set j [expr {int(floor(double(1-$y)*$imageheight))}]
     if {$j>=$imageheight} {set j [expr {$imageheight-1}]}
     set index [expr {3*($j*$imagewidth+$i)+3}]  ;# +3 is to skip header
     set vx [expr {2*[lindex $colorimage $index]-1}] ; incr index  ;# Red
     set vy [expr {2*[lindex $colorimage $index]-1}] ; incr index  ;# Green
     set vz [expr {2*[lindex $colorimage $index]-1}] ; incr index  ;# Blue
     return [list $vx $vy $vz]
}

Specify Oxs_ScriptVectorField:sample {
   atlas :atlas
   norm 1.0
```

```
   script ColorField
}
```

If the input image is large, then it is best to work with the image list (i.e., the variable `colorimage` in the preceding example) directly, as illustrated above. The image list as returned by `ReadFile` is in a packed format; if you make modifications to the list values then the memory footprint of the list can grow substantially.

The `ReadFile` command is not available if the `MIFinterp safety` option is set to `safe` in the `options.tcl` customization file.

RGlob This command is modeled on the Tcl `glob` command (q.v.), but is restricted to the current working directory, that is, the directory holding the MIF file. The syntax is

```
RGlob [-types typelist] [--] <pattern> [...]
```

The optional `typelist` restricts the match to files meeting the typelist criteria. The optional `--` switch marks the end of options. The one or more `pattern`'s should be glob-style patterns (strings containing asterisks and question marks) intended to match filenames in the current working directory. See the Tcl `glob` documentation for details on the `-types` option and glob pattern details.

One use of this command is to identify files created by earlier runs of Oxs. For example, suppose we wanted to use the mmArchive magnetization output from the third stage of a previous MIF file with basename "sample". Output files are tagged by stage number (here "2" since stages are counted from 0) and iteration. The iteration is generally not known a priori, but assuming the output files are in the same directory as the current MIF file, we could use a command like

```
set file [RGlob sample-Oxs_MinDriver-Magnetization-02-???????.omf]
```

to determine the name of the magnetization file. If more than one magnetization state was saved for that stage, then the variable `file` will hold a list of filenames. In this case the Tcl `lsort` command can be used to select the one with the highest iteration number. The `file` variable can be used in conjunction with the `Oxs_FileVectorField` class to import the magnetization into the new simulation, for example to set the initial magnetization configuration.

The `RGlob` command is not available if the `MIFinterp safety` option is set to `safe` in the `options.tcl` customization file. If `MIFinterp safety` is set to `unsafe`, then the standard (and more capable) Tcl `glob` command will be available.

Schedule The `Schedule` command is used to setup outputs from the MIF file. This functionality is critical for solvers running in batch mode, but is also useful for setting up default connections in interactive mode.

The syntax for the `Schedule` command is

```
Schedule <outname> <desttag> <event> <frequency>
```

The `Schedule` command mirrors the interactive output scheduling provided by the **Oxsii** and **Boxsi** graphical interfaces (Ch. 7). The first parameter to the `Schedule` command is the name of the output being scheduled. These names are the same as those appearing in the "Outputs" list in the Oxs graphical interfaces, for example "DataTable" or "Oxs_CubicAnisotropy:Nickel:Field." The name must be presented to the `Schedule` command as a single argument; if the name includes one or more spaces then use double quotes to protect the spaces. Aside from the DataTable output which is always present, the *outname*'s are MIF file dependent.

The second parameter to the `Schedule` command is a destination tag. This is a tag associated to a running application by a previous `Destination` command (see above). The symbolic destination tag replaces the application:OID nomenclature used in the graphical interface, because in general it is not possible to know the OOMMF ID for application instances at the time the MIF file is composed. In fact, some of the applications may be launched by `Destination` commands, and so don't even have OID's at the time the `Destination` command is processed.

The *event* parameter should be one of the keywords `Step`, `Stage`, or `Done`. For `Step` and `Stage` events the *frequency* parameter should be a non-negative integer, representing with what frequency of the specified event should output be dispatched. For example, if `Step 5` is given, then on every fifth step of the solver (iterations 0, 5, 10, ...) output of the indicated type will be sent to the selected destination. Set *frequency* to 1 to send output each time the event occurs. A value of 0 for *frequency* results in output on only the very first event of that type; in particular, `Step 0` will output the simulation initial state, but will not fire on any subsequent `Step` events. The `Done` event occurs at the successful completion of a simulation; as such, there is at most one "Done" event per simulation. Accordingly, the *frequency* parameter for `Done` events is optional; if present it should be the value 1.

There are examples of scheduling with the `Destination` and `Schedule` commands in the sample MIF 2.1 file (Sec. 17.3.5, pages 251–254). There, three destinations are tagged. The first refers to a possibly already running instance of **mmGraph**, having nickname Hysteresis. The associated `Schedule` command sends DataTable output to this application at the end of each Stage, so hysteresis graphs can be produced. The second destination tag references a different copy of **mmGraph** that will be used for monitoring the run. To make sure that this output is rendered onto a blank slate, the `new` keyword is used to launch a fresh copy of **mmGraph**. The `Schedule` command for the monitor destination delivers output to the monitoring **mmGraph** every 5 iterations of the solver. The last `Destination` command tags an arbitrary **mmArchive** application, which is used for file storage of DataTable results at the end of each stage, and snapshots of the magnetization and total field at the end of every third stage. Note that double quotes enclose the "Oxs_EulerEvolve::Total field"

output name. Without the quotes, the `Schedule` command would see five arguments, "Oxs_EulerEvolve::Total", "field", "archive", "Stage", and "3".

17.3.3 Specify Conventions

The Specify blocks in the input MIF file determine the collection of `Oxs_Ext` objects defining the Oxs simulation. As explained above, the `Specify` command takes two arguments, the name of the `Oxs_Ext` object to create, and an initialization string. The format of the initialization string can be arbitrary, as determined by the author of the `Oxs_Ext` class. This section presents a number of recommended conventions which `Oxs_Ext` class authors are encouraged to follow. Any `Oxs_Ext` classes that don't follow these conventions should make that fact explicitly clear in their documentation. Details on the standard `Oxs_Ext` classes included with OOMMF can be found in the Oxs section.

Initialization string format
Comments
Oxs_Ext referencing
User defined scalar outputs
Attributes
Grouped lists
User defined support procedures

Initialization string format

Consider again the simple Specify block presented above:

```
Specify Oxs_EulerEvolve:foo {
  alpha 0.5
  start_dm 0.01
}
```

The first convention is that the initialization string be structured as a Tcl list with an even number of elements, with consecutive elements consisting of a label + value pairs. In the above example, the initialization string consists of two label + value pairs, "alpha 0.5" and "start_dm 0.01". The first specifies that the damping parameter α in the Landau-Lifshitz ODE is 0.5. The second specifies the initial step size for the integration routine. Interested parties should refer to a Tcl programming reference (e.g., [34]) for details on forming a proper Tcl list, but in short the items are separated by whitespace, and grouped by double quotes or curly braces ("`{`" and "`}`"). Opening braces and quotes must be whitespace separated from the preceding text. Grouping characters are removed during parsing. In this example the list as a whole is set off with curly braces, and individual elements are white space delimited. Generally, the ordering of the label + value pairs in the initialization string is irrelevant, i.e., `start_dm 0.01` could equivalently precede `alpha 0.5`.

Sometimes the value portion of a label + value pair will itself be a list, as in this next example:

```
Specify Oxs_BoxAtlas:myatlas {
        ...
```

```
}

Specify Oxs_RectangularMesh:mymesh {
    cellsize { 5e-9 5e-9 5e-9 }
    atlas Oxs_BoxAtlas:myatlas
}
```

Here the value associated with "cellsize" is a list of 3 elements, which declare the sampling rate along each of the coordinate axes, in meters. (`Oxs_BoxAtlas` is a particular type of `Oxs_Atlas`, and "..." mark the location of the `Oxs_BoxAtlas` initialization string, which is omitted because it is not pertinent to the present discussion.)

Attributes

Sometimes it is convenient to define label + value pairs outside a particular Specify block, and then import them using the "attributes" label. For example:

```
Specify Oxs_LabelValue:probdata {
  alpha 0.5
  start_dm 0.01
}

Specify Oxs_EulerEvolve {
  attributes :probdata
}
```

The `Oxs_LabelValue` object is an `Oxs_Ext` class that does nothing except hold label + value pairs. The "attributes" label acts as an include statement, causing the label + value pairs contained in the specified `Oxs_LabelValue` object to be embedded into the enclosing `Specify` initialization string. This technique is most useful if the label + value pairs in the `Oxs_LabelValue` object are used in multiple Specify blocks, either inside the same MIF file, or across several MIF files into which the `Oxs_LabelValue` block is imported using the `ReadFile` MIF extension command.

Comments

The standard Tcl commenting mechanism treats all text running from an initial `#` symbol through to the end of a line as a comment. You may note in the above examples that newlines are treated the same as other whitespace inside the curly braces delimiting the `Specify` initialization string. Because of this and additional reasons, Tcl comments cannot be used inside Specify blocks. Instead, by convention any label + value pair where label is "comment" is treated as a comment and thrown away. For example:

```
Specify Oxs_UniaxialAnisotropy {
  axis { 1 0 0 }
```

```
    comment {K1  4500e3}
    K1 530e3
    comment { 530e3 J/m^3 is nominal for Co }
  }
```

Pay attention to the difference between "comment" used here as the label portion of a label + value pair, and the MIF extension command "Ignore" used outside Specify blocks. In particular, `Ignore` takes an arbitrary number of arguments, but the value element associated with a comment label must be grouped as a single element, just as any other value element.

Grouped lists

As noted earlier, sometimes the value portion of a label + value pair will be a list. Some Oxs objects support *grouped lists*, which provide a type of run-length encoding for lists. Consider the sample list

```
{ 1.1  1.2  1.2  1.2  1.2  1.3 }
```

In a grouped list the middle run of 1.2's may be represented as a sublist with a repeat count of 4, like so

```
{ 1.1  { 1.2  4 } 1.3 :expand: }
```

Here the `:expand:` keyword, when appearing as the last element of the top level list, enables the group expansion mechanism. Any preceding element, such as `{ 1.2 4 }`, that 1) is a proper sublist, and 2) has a positive integer as the last element, is treated as a grouped sublist with repeat count given by the last element. No element of the top-level list is ever interpreted as a repeat count. For example, the short form of the list

```
{ 1e-9 1e-9 1e-9 1e-9 1e-9 1e-9 }
```

is

```
{ { 1e-9 6 } :expand: }
```

Note the additional level of brace grouping. Grouped lists may also be nested, as in this example

```
{ 5.0 { 5.1 { 5.2 3 } 5.3 2 } :expand: }
```

which is equivalent to

```
{ 5.0  5.1  5.2  5.2  5.2  5.3  5.1  5.2  5.2  5.2  5.3 }
```

There are some difficulties with this mechanism when the list components are strings, such as filenames, that may contain embedded spaces. For example, consider the list

```
{ "file 3" "file 3" "5 file" }
```

If we tried to write this as

```
{ { "file 3" 2 } "5 file" :expand: }
```

we would find that, because of the nested grouping rules, this grouped list gets expanded into

```
{ file file file file file file "5 file" }
```

Here the trailing "3" in "file 3" is interpreted as a repeat count. Following normal Tcl rules, the double quotes are treated as equivalents to braces for grouping purposes. However, the keyword `:noexpand:` may be used to disable further expansion, like so

```
{ { {"file 3" :noexpand:} 2 } "5 file" :expand: }
```

The `:noexpand:` keyword placed at the end of a list disables all group expansion in that list. Although it is an unlikely example, if one had a flat, i.e., non-grouped list with last element ":expand:", then one would have to disable the grouping mechanism that would otherwise be invoked by appending `:noexpand:` to the list. In flat lists generated by program code, it is recommended to append `:noexpand:` just to be certain that the list is not expanded.

As a matter of nomenclature, standard (i.e., flat) lists and single values are also considered grouped lists, albeit trivial ones. Any Oxs object that accepts grouped lists in its Specify block should explicitly state so in its documentation.

`Oxs_Ext` referencing

The "atlas" value in the mesh Specify block of the preceding example refers to an earlier `Oxs_Ext` object, "Oxs_BoxAtlas:myatlas". It frequently occurs that one `Oxs_Ext` object needs access to another `Oxs_Ext` object. In this example the mesh object `:mymesh` needs to query the atlas object `:myatlas` in order to know the extent of the space that is to be gridded. The atlas object is defined earlier in the MIF input file by its own, separate, top-level Specify block, and the mesh object refers to it by simply specifying its name. Here the full name is used, but the short form `:myatlas` would suffice, provided no other `Oxs_Ext` object has the same short name.

Alternatively, the `Oxs_RectangularMesh` object could define an `Oxs_BoxAtlas` object inline:

```
Specify Oxs_RectangularMesh:mymesh {
    atlas {
        Oxs_BoxAtlas {
            ...
        }
    }
    cellsize { 5e-9 5e-9 5e-9 }
}
```

In place of the name of an external atlas object, a two item list is provided consisting of the type of object (here `Oxs_BoxAtlas`) and the corresponding initialization string. The initialization string is provided as a sublist, with the same format that would be used if that object were initialized via a separate Specify block.

More commonly, embedded `Oxs_Ext` objects are used to initialize spatially varying quantities. For example,

```
Specify Oxs_UniaxialAnisotropy {
  axis { Oxs_RandomVectorField {
           min_norm 1
           max_norm 1
  }}
  K1  { Oxs_UniformScalarField { value 530e3 } }
}
```

The magneto-crystalline anisotropy class `Oxs_UniaxialAnisotropy` supports cellwise varying K1 and anisotropy axis directions. In this example, the anisotropy axis directions are randomly distributed. To initialize its internal data structure, `Oxs_UniaxialAnisotropy` creates a local `Oxs_RandomVectorField` object. This object is also a child of the `Oxs_Ext` hierarchy, which allows it to be constructed using the same machinery invoked by the `Specify` command. However, it is known only to the enclosing `Oxs_UniaxialAnisotropy` object, and no references to it are possible, either from other Specify blocks or even elsewhere inside the same initialization string. Because it cannot be referenced, the object does not need an instance name. It does need an initialization string, however, which is given here as the 4-tuple "min_norm 1 max_norm 1". Notice how the curly braces are nested so that this 4-tuple is presented to the `Oxs_RandomVectorField` initializer as a single item, while "Oxs_RandomVectorField" and the associated initialization string are wrapped up in another Tcl list, so that the value associated with "axis" is parsed at that level as a single item.

The value associated with "K1" is another embedded `Oxs_Ext` object. In this particular example, K1 is desired uniform (homogeneous) throughout the simulation region, so the trivial `Oxs_UniformScalarField` class is used for initialization (to the value 530×10^3 J/m^3). In the case of uniform fields, scalar or vector, a shorthand notation is available that implicitly supplies a uniform `Oxs_Ext` field class:

```
Specify Oxs_UniaxialAnisotropy {
  axis { 1 0 0 }
  K1  530e3
}
```

which is equivalent to

```
Specify Oxs_UniaxialAnisotropy {
  axis { Oxs_UniformVectorField {
           vector { 1 0 0 }
```

```
  }}
  K1  { Oxs_UniformScalarField { value 530e3 } }
}
```

While embedding `Oxs_Ext` objects inside Specify blocks can be convenient, it is important to remember that such objects are not available to any other `Oxs_Ext` object—only objects declared via top-level Specify blocks may be referenced from inside other Specify blocks. Also, embedded `Oxs_Ext` objects cannot directly provide user output. Furthermore, the only `Oxs_Energy` energy objects included in energy and field calculations are those declared via top-level Specify blocks. For this reason `Oxs_Energy` terms are invariably created via top-level Specify blocks, and not as embedded objects.

User defined support procedures

A number of `Oxs_Ext` classes utilize user-defined Tcl procedures (procs) to provide extended runtime functionality. The most common examples are the various field initialization script classes, which call a user specified Tcl proc for each point in the simulation discretization mesh. The proc returns a value, either scalar or vector, which is interpreted as some property of the simulation at that point in space, such as saturation magnetization, anisotropy properties, or an external applied field.

Here is an example proc that may be used to set the initial magnetization configuration into an approximate vortex state, with a central core in the positive z direction:

```
proc Vortex { x_rel y_rel z_rel } {
   set xrad [expr {$x_rel-0.5}]
   set yrad [expr {$y_rel-0.5}]
   set normsq [expr {$xrad*$xrad+$yrad*$yrad}]
   if {$normsq <= 0.0125} {return "0 0 1"}
   return [list [expr {-1*$yrad}] $xrad 0]
}
```

The return value in this case is a 3D vector representing the spin direction at the point `(x_rel,y_rel,z_rel)`. Procs that are used to set scalar properties, such as saturation magnetization M_s, return a scalar value instead. But in both cases, the import argument list specifies a point in the simulation mesh.

In the above example, the import point is specified relative to the extents of the simulation mesh. For example, if `x_rel` were 0.1, then the x-coordinate of the point is one tenth of the way between the minimum x value in the simulation and the maximum x value. In all cases `x_rel` will have a value between 0 and 1.

In most support proc examples, relative coordinates are the most flexible and easiest representation to work with. However, by convention, scripting `Oxs_Ext` classes also support absolute coordinate representations. The representation used is selected in the `Oxs_Ext` object Specify block by the optional `script_args` entry. The Tcl proc itself is specified by the `script` entry, as seen in this example:

```
proc SatMag { x y z } {
   if {$z < 20e-9} {return 8e5}
   return 5e5
}

Specify ScriptScalarField:Ms {
   atlas :atlas
   script_args { rawpt }
   script SatMag
}
```

The value associated with the label `script_args` should in this case be a subset of {`relpt rawpt minpt maxpt span scalars vectors`}, as explained in the `Oxs_ScriptScalarField` documentation (page 99). Here `rawpt` provides the point representation in problem coordinates, i.e., in meters. Other `Oxs_Ext` objects support a different list of allowed `script_args` values. Check the documentation of the `Oxs_Ext` object in question for details. Please note that the names used in the proc argument lists above are for exposition purposes only. You may use other names as you wish. It is the order of the arguments that is important, not their names. Also, MIF 2.1 files are parsed first in toto before the Specify blocks are evaluated, so the support procs may be placed anywhere in a MIF 2.1 file, regardless of the location of the referencing Specify blocks. Conversely, MIF 2.2 files are parsed in a single pass, with Specify blocks evaluated as they are read. Therefore for MIF 2.2 files it is generally best to place proc definitions ahead of Specify blocks in which they are referenced.

The command call to the Tcl support proc is actually built up by appending to the `script` value the arguments as specified by the `script_args` value. This allows additional arguments to the Tcl proc to be specified in the `script` value, in which case they will appear in the argument list in front of the `script_args` values. The following is equivalent to the preceding example:

```
proc SatMag { zheight Ms1 Ms2 x y z } {
   if {$z < $zheight} {return $Ms1}
   return $Ms2
}

Specify ScriptScalarField:Ms {
   script_args { rawpt }
   script {SatMag 20e-9 8e5 5e5}
}
```

Notice in this case that the `script` value is wrapped in curly braces so that the string `SatMag 20e-9 8e5 5e5` will be treated as the single value associated with the label `script`.

As seen in the earlier example using the `Vortex` Tcl proc, support procedures in MIF 2.1 files will frequently make use of the Tcl `expr` command. If you are using Tcl version 8.0 or later, then the cpu time required by the potentially large number of calls to such procedures

can be greatly reduced by grouping the arguments to `expr` commands in curly braces, as illustrated in the `Vortex` example. The braces aid the operation of the Tcl byte code compiler, although there are a few rare situations involving multiple substitution where such bracing cannot be applied. See the Tcl documentation for the `expr` command for details.

Sometimes externally defined data can be put to good use inside a Tcl support proc, as in this example:

```
# Lay out a 6 x 16 mask, at global scope.
set mask {
   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1
   1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1
   1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1
}

proc MyShape { xrel yrel znotused } {
   global mask  ;# Make mask accessible inside proc
   set Ms 8e5   ;# Saturation magnetization of element
   set xindex [expr {int(floor($xrel*16))}]
   set yindex [expr {5 - int(floor($yrel*6))}]
   set index [expr {$yindex*16+$xindex}]
   # index references point in mask corresponding
   # to (xrel,yrel)
   return [expr {[lindex $mask $index]*$Ms}]
}
```

The variable `mask` holds a Tcl list of 0's and 1's defining a part shape. The mask is brought into the scope of the `MyShape` proc via the Tcl `global` command. The relative x and y coordinates are converted into an index into the list, and the proc return value is either 0 or 8e5 depending on whether the corresponding point in the mask is 0 or 1. This is essentially the same technique used in the `ColorField` proc example presented in the `ReadFile` MIF extension command documented above (Sec. 17.3.2), except that there the data structure values are built from a separate image file rather than from data embedded inside the MIF file.

User defined scalar outputs

OOMMF `Oxs_Ext` objects support a general method allowing users to create scalar (DataTable) outputs from default scalar and vector field outputs. These scalar outputs are defined by "`user_output`" sub-blocks inside Specify blocks. The format is:

```
user_output {
    name output_name
```

```
    source_field source
    select_field weighting
    normalize norm_request
    exclude_0_Ms novacuum
    user_scaling scale
    units units
}
```

The first parameter, **name**, specifies the label attached to this output in the DataTable results; the full name will be the Specify block instance name, followed by *:output_name*. This label must follow the rules for ODT column labels; in particular, embedded newlines and carriage returns are not allowed.

The second parameter, **source_field**, specifies the scalar or 3-vector field output that the output is derived from. The *source* value should match the label for the source field output as displayed in the "Output" pane of the Oxsii or Boxsi interactive interface; this can also be found in the documentation for the source field `Oxs_Ext` class. If the source field is from the same class as the user output, then *source* can use the short form of the name (i.e., the component following the last ":"); otherwise the full name must be used.

The third parameter, **select_field**, references a field that is used to weight the source field to create the scalar output. The select field should be of the same field type, scalar or vector, as the source field. The output is computed according to

$$\sum_i W_{\text{select}}[i] \cdot V_{\text{source}}[i] / \sum_i \|W_{\text{select}}[i]\| \tag{17.1}$$

where the sums are across all cells in the simulation, $W_{\text{select}}[i]$ is the value of the select field at cell i, and $V_{\text{source}}[i]$ is the value of the source field at cell i. The "$\cdot$" indicates multiplication for scalar fields, or the scalar (dot) product for vector fields.

The first three parameters are required, the remaining parameters are optional. The first of the optional parameters, **normalize**, affects the denominator in (17.1). If *norm_request* is 1 (the default), then the output is computed as shown in (17.1). If *norm_request* is 0, then instead the denominator is replaced with the number of cells in the simulation, i.e., $\sum_i 1$.

The second optional parameter, **exclude_0_Ms**, is a convenience operator; if *novacuum* is 1, then the select field is reset so that it is zero at all cells in the simulation where the saturation magnetization is zero. This is especially useful when you want to compute the average magnetization of a shaped part. The change to the select field is made before the denominator in (17.1) is computed, so setting **exclude_0_Ms** to 1 is equivalent to defining the select field as being zero where M_s is zero in the first place. The default value for this parameter is 0, which results in the select field being used exactly as defined.

The **user_scaling** parameter (default value 1.0) allows the user to define a constant factor that is multiplied against the result of (17.1). This can be used, for example, in conjuction with the **units** parameter to implement unit conversion. The *units* value is an arbitrary string (for example, A/m) that is placed in the DataTable output. This label must

follow the rules for ODT unit labels; in particular, embedded newlines and carriage returns are not allowed. If *units* is not set, then the units string is copied from the units for the source field.

The following example shows four user outputs based off the demagnetization field:

```
Specify Oxs_BoxAtlas:atlas [subst {
  xrange {0 $cube_edge}
  yrange {0 $cube_edge}
  zrange {0 $cube_edge}
}]

Specify Oxs_BoxAtlas:octant [subst {
  xrange {0 [expr {$cube_edge/2.}]}
  yrange {0 [expr {$cube_edge/2.}]}
  zrange {0 [expr {$cube_edge/2.}]}
}]

Specify Oxs_AtlasVectorField:octant_field_y {
   atlas :octant
   default_value {0 0 0}
   values {
      octant {0 1 0}
   }
}

Specify Oxs_AtlasScalarField:octant_complement {
   atlas :octant
   default_value 1
   values {
      octant 0
   }
}

Specify Oxs_Demag {
  user_output {
     name "Hdemag_x"
     source_field Field
     select_field {1 0 0}
  }
  user_output {
     name "octant Hdemag_y"
     source_field Field
     select_field :octant_field_y
```

```
  }
  user_output {
     name "body Hdemag_z"
     source_field Field
     select_field {0 0 1}
     exclude_0_Ms 1
  }
  user_output {
     name "octant_complement E_demag"
     source_field "Energy density"
     select_field :octant_complement
  }
}
```

The first user output, "`Hdemag_x`," returns the x-component of the demagnetization field, averaged across the entire simulation volume. This output will appear in DataTable output with the label "`Oxs_Demag::Hdemag_x`." The `source_field` parameter "Field" refers to the "Field" output of the surrounding `Oxs_Ext` object, which here means `Oxs_Demag::Field`. The select field $(1, 0, 0)$, uniform across the simulation volume, is used to extract the x-component of the demagnetization field. Separate user output blocks with analogous select fields would be necessary to obtain the y- and z-field components. The select field can also be used to obtain the projection of a field into a particular direction; for example, `select_field {0.7071068 0.7071068 0 }` would yield the projection of the demagnetization field in the direction 45° between the x- and y-axes.

The second output, "`octant Hdemag_y`," is similar, but here the the y component of the demagnetization field is averaged across only the first octant of the simulation volume. The averaging volume and component selection are defined by the `:octant_field_y` field object, which is $(0, 1, 0)$ in the first octant and $(0, 0, 0)$ everywhere else.

In the third output, the "`exclude_0_Ms`" option restricts the averaging to only those regions of the simulation with $M_s \neq 0$. This option is convenient for finding averages over irregularly shaped bodies.

The final user output, "`octant_complement E_demag`," reports the average demag energy density *outside* the first octant. For this the select field `:octant_complement` is set to 0 inside the first octant and 1 everywhere else. Note that unlike the first three examples, the "Energy density" output is a scalar field, to the select field must be a scalar field too.

The source code for user defined scalar outputs can be found in the files `ext.h` and `ext.cc` in the directory `oommf/app/oxs/base/`. Example MIF files include `cube.mif`, `pbcbrick.mif`, `spinvalve.mif`, and `stdprob2.mif` in `oommf/app/oxs/examples/`.

17.3.4 Variable Substitution

One powerful consequence of the evaluation of MIF 2.1 input files by Tcl is the ability to define and use variables. For example, the Oxs interfaces (Oxsii and Boxsi) use the `-parameter`

command line option in conjunction with the MIF Parameter command to set variables from the command line for use inside the MIF input file. Variables in Tcl are evaluated (i.e., value substituted) by prefixing the variable name with the symbol "$". For example, if `cellsize` is a variable holding the value `5e-9`, then `$cellsize` evaluates to `5e-9`.

Unfortunately, there are complications in using variables inside Specify blocks. Consider this simple example:

```
Parameter cellsize 5e-9
Specify Oxs_RectangularMesh:BadExample {
  comment {NOTE: THIS DOESN'T WORK!!!}
  cellsize {$cellsize $cellsize $cellsize}
  atlas :atlas
}
```

This **doesn't work**, because the curly braces used to set off the `Specify` initialization string also inhibit variable substitution. There are several ways to work around this, but the easiest is usually to embed the initialization string inside a `subst` (substitution) command:

```
Parameter cellsize 5e-9
Specify Oxs_RectangularMesh:GoodExample [subst {
  comment {NOTE: This works.}
  cellsize {$cellsize $cellsize $cellsize}
  atlas :atlas
}]
```

Here the square brackets, "`[`" and "`]`", cause Tcl to perform *command substitution*, i.e., execute the string inside the square brackets as a Tcl command, in this case the `subst` command. See the Tcl documentation for `subst` for details, but the default usage illustrated above performs variable, command and backslash substitutions on the argument string.

One more example, this time involving both variable and command substitution:

```
set pi [expr {4*atan(1.0)}]
set mu0 [expr {4*$pi*1e-7}]
Specify Oxs_UZeeman [subst {
  comment {Set units to mT}
  Hscale [expr {0.001/$mu0}]
  Hrange {
     {  0  0  0   10  0  0   2 }
     { 10  0  0  -10  0  0   2 }
  }
}]
```

Note that the `subst` command is evaluated at global scope, so that the global variable `mu0` is directly accessible.

17.3.5 Sample MIF 2.1 File

```
# MIF 2.1
#
# All units are SI.
#
# This file must be a valid Tcl script.
#

# Initialize random number generators with seed=1
RandomSeed 1

# Individual Oxs_Ext objects are loaded and initialized via
# Specify command blocks.  The following block defines the
# extents (in meters) of the volume to be modeled.  The
# prefix "Oxs_BoxAtlas" specifies the type of Oxs_Ext object
# to create, and the suffix ":WorldAtlas" is the name
# assigned to this particular instance.  Each object created
# by a Specify command must have a unique full name (here
# "Oxs_BoxAtlas:WorldAtlas").  If the suffix is not
# explicitly given, then the default ":" is automatically
# assigned.  References may be made to either the full name,
# or the shorter suffix instance name (here ":WorldAtlas")
# if the latter is unique. See the Oxs_TimeDriver block for
# some reference examples.
Specify Oxs_BoxAtlas:WorldAtlas {
  xrange {0 500e-9}
  yrange {0 250e-9}
  zrange {0 10e-9}
}

# The Oxs_RectangularMesh object is initialized with the
# discretization cell size (in meters).
Specify Oxs_RectangularMesh:mesh {
  cellsize {5e-9 5e-9 5e-9}
  atlas :WorldAtlas
}

# Magnetocrystalline anisotropy block.   The setting for
# K1 (500e3 J/m^3) implicitly creates an embedded
# Oxs_UniformScalarField object.  Oxs_RandomVectorField
# is an explicit embedded Oxs_Ext object.
Specify Oxs_UniaxialAnisotropy {
```

```
  K1  530e3
  axis { Oxs_RandomVectorField {
           min_norm 1
           max_norm 1
  } }
}

# Homogeneous exchange energy, in J/m.  This may be set
# from the command line with an option like
#    -parameters "A 10e-12"
# If not set from the command line, then the default value
# specified here (13e-12) is used.
Parameter A 13e-12
Specify Oxs_UniformExchange:NiFe [subst {
  A  $A
}]

# Define a couple of constants for later use.
set PI [expr {4*atan(1.)}]
set MU0 [expr {4*$PI*1e-7}]

# The Oxs_UZeeman class is initialized with field ranges in A/m.
# The following block uses the multiplier option to allow ranges
# to be specified in mT.  Use the Tcl "subst" command to enable
# variable and command substitution inside a Specify block.
Specify Oxs_UZeeman:AppliedField [subst {
  multiplier [expr 0.001/$MU0]
  Hrange {
    {  0  0  0   10  0  0   2 }
    { 10  0  0  -10  0  0   2 }
    {  0  0  0    0 10  0   4 }
    {  1  1  1    5  5  5   0 }
  }
}]

# Enable demagnetization (self-magnetostatic) field
# computation.  This block takes no parameters.
Specify Oxs_Demag {}

# Runge-Kutta-Fehlberg ODE solver, with default parameter values.
Specify Oxs_RungeKuttaEvolve {}
```

```
# The following procedure is used to set the initial spin
# configuration in the Oxs_TimeDriver block.  The arguments
# x, y, and z are coordinates relative to the min and max
# range of each dimension, e.g., 0<=x<=1, where x==0
# corresponds to xmin, x==1 corresponds to xmax.
proc UpDownSpin { x y z } {
  if { $x < 0.45 } {
    return "0 1 0"
  } elseif { $x > 0.55 } {
    return "0 -1 0"
  } else {
    return "0 0 1"
  }
}

Specify Oxs_TimeDriver {
 evolver Oxs_RungeKuttaEvolve
 stopping_dm_dt 0.01
 mesh :mesh
 Ms 8e5   comment {implicit Oxs_UniformScalarField object}
 m0 { Oxs_ScriptVectorField {
        script {UpDownSpin}
        norm  1
        atlas :WorldAtlas
 } }
 basename example
 comment {If you don't specify basename, then the default
          is taken from the MIF filename.}
}

# Default outputs
Destination hystgraph mmGraph:Hysteresis
Destination monitor   mmGraph   new
Destination archive   mmArchive

Schedule DataTable hystgraph Stage 1
Schedule DataTable monitor   Step 5
Schedule DataTable archive   Stage 1
Schedule Oxs_TimeDriver::Magnetization archive Stage 3
Schedule "Oxs_RungeKuttaEvolve::Total field" archive Stage 3
```

Figure 17.2: Example MIF 2.1 file. (description)

17.4 MIF 2.2

The MIF 2.2 format, introduced with OOMMF 1.2a4, is a minor modification to the MIF 2.1 format. MIF 2.2 provides a few additional commands, and is mostly backwards compatible with MIF 2.1, except as detailed below.

17.4.1 Differences between MIF 2.2 and MIF 2.1 Formats

1. The first line of a MIF 2.2 file must be "`# MIF 2.2`".

2. The `basename`, `scalar_output_format` and `vector_field_output_format` options to the `Oxs_TimeDriver` and `Oxs_MinDriver` objects are no longer supported. Instead, there is a new top-level extension command, `SetOptions`, where these options are declared. The `SetOptions` block also supports new options for controlling output formats, mesh types, and filenames.

3. In the MIF 2.1 format, MIF files are processed in a two pass mode. During the first pass, `Specify` commands simply store the contents of the `Specify` blocks without creating any `Oxs_Ext` objects. The `Oxs_Ext` objects associated with each `Specify` block are created in the second pass from the data stored in the first pass. In the MIF 2.2 format, this is replaced with a one pass mode, where `Oxs_Ext` objects are created at the time that the `Specify` commands are parsed. This processing model is more intuitive for MIF file authors, but has two main consequences. The first is that in MIF 2.1 format files, Tcl procs that are used only inside `Specify` commands can be placed anywhere inside the MIF file (for example, commonly at the end), because they won't be called during the first pass. As long as they are defined at any point during the first pass, they will be available for use in the second pass. In contrast, in the MIF 2.2 format, Tcl procs definitions must generally be moved forward, before any references in `Specify` blocks. The second consequence is that `Oxs_Ext` objects defined by `Specify` commands are available for use inside the MIF file. This allows support for the new commands discussed next.

17.4.2 MIF 2.2 New Extension Commands

In addition to the commands available in MIF 2.1 files, MIF 2.2 introduces the following new commands: `GetMifFilename`, `GetMifParameters`, `GetOptions`, `SetOptions`, `DefaultFieldFilename`, `EvalScalarField`, `EvalVectorField`, `GetAtlasExtent`, `GetAtlasRegions`, and `GetAtlasRegionByPosition`.

GetMifFilename The `GetMifFilename` command returns the full (absolute) name of the MIF file being read. This command takes no parameters.

GetMifParameters This command takes no parameters, and returns an even numbered list of "Parameter" label + value pairs as set on the command line or in the Load Problem dialog box. If no parameters were specified, then the return will be an empty list.

GetOptions The `GetOptions` command takes no parameters. It returns the accumulated contents of all preceding `SetOptions` blocks, as an even numbered list of label + value pairs.

SetOptions In MIF 2.1 files, the output basename and output file formats are specified inside the driver's Specify block. In MIF 2.2 these specifications are moved to a separate `SetOptions` block. This block can be placed anywhere in the MIF file, but is typically placed near the start of the file so that it affects all output initializations. The `SetOptions` command takes a single argument, which is a list of label + value pairs. The supported labels are:

- `basename`
- `scalar_output_format`
- `scalar_field_output_format`
- `scalar_field_output_meshtype`
- `scalar_field_output_filename_script`
- `vector_field_output_format`
- `vector_field_output_meshtype`
- `vector_field_output_filename_script`

The `basename` value is used as a prefix for output filename construction by the data output routines. If `basename` is not specified, then the default value is the filename of the input MIF file minus its extension (`.mif`). The `scalar_output_format` value is a C-style printf string specifying the output format for DataTable output. The default value is "`%.17g`". The `scalar(vector)_field_output_format` value should be a two element list that specifies the style and precision for output sent to **mmDisp** and **mmArchive**. The first element in the list should be one of `binary` or `text`, specifying the output style. If binary output is selected, then the second element specifying precision should be either 4 or 8, denoting component binary output length in bytes. For text output, the second element should be a C-style printf string like that used by `scalar_output_format`. Be forewarned that for large field files text-style output may be many times slower that binary; if you require text format for post-processing, it may be better to save from **Oxsii**/**Boxsi**

in binary format and convert afterwards to text using **avf2ovf**. The default value for `scalar(vector)_field_output_format` is "`binary 8`". The grid type for the `scalar(vector)_field_output_meshtype` options should be either "rectangular" (default) or "irregular". The "`scalar(vector)_field_output_filename_script`" value is a Tcl script used to construct filenames for the respective output type.

Typically `SetOptions` is declared near the top of the MIF file, before any `Specify` blocks, so that the options apply to all `Oxs_Ext` objects. However, multiple `SetOptions` blocks are allowed. Values specified in one `SetOption` block will be superseded if reset by later `SetOption` blocks. Values not specified in a `SetOption` block retain the previous setting. Judicious placement of `SetOptions` blocks allows formats and filenames to be varied across different outputs in one MIF file.

For outputs associated with a `Destination` tag (Sec. 17.3.2) in the MIF file, any options appended onto the `Destination` command take precedence. Otherwise, the options used are those active at the point of the `Destination` command, except for options specified as an empty string — those instead inherit the options active at the point of the `Specify` command for the output owner.

Any outputs not associated with a `Destination` command operate with the options active at the bottom of the MIF file.

The filename for `DataTable` output written through **mmArchive** has the form

```
<basename>.odt
```

For the field outputs, the default filename format is

```
<basename>-<outputname>-<stage>-<iteration>.<ext>
```

where the stage and iteration are two and seven digit integers, with leading zeros as needed. (If the total number of stages in a simulation is larger than 99, then the stage format width is expanded to fit.) The extension is determined by the type of output: `.omf` for magnetization, `.ohf` for H fields, `.obf` for B fields, `.oef` for energy density fields, and `.ovf` for everything else. The filename format can be altered using a filename script. This script should be a Tcl `proc` that takes one argument and returns the desired filename. The import argument is a Tcl `dict` structure that includes the following fields:

basename	The active `SetOption` basename, extended to an absolute pathname
event	Generating event: `Step`, `Stage`, `Done`, or `Interactive`
fullname	Output owner:name, as it appears in the **Oxsii**/**Boxsi** interactive interface; e.g., `Oxs_UZeeman::Field`
iteration	Iteration count

iteration_fmt	Suggested iteration format; defaults to "%07d"
name	The output short name, e.g. `"Total field"`
owner	The output owner including optional instance name, e.g. `Oxs_UZeeman:` or `Oxs_RungeKuttaEvolve:evolver`
ptype	The protocol type; either `scalarField` or `vectorField`
saniname	File-system friendly "sanitized" version of fullname, e.g., `Oxs_RungeKuttaEvolve-evolver-Total_field`
stage	Stage number
stage_fmt	Suggested stage format; default to "%02d"
state_id	Internal state number
units	Output units, e.g., "A/m"; this will be an empty string if the output is unitless

Inside the filename proc use the Tcl `dict get` command to retrieve items from the params dict as needed. You can also call

```
DefaultFieldFilename $params
```

to get the default scalar or vector field filename as described above.

The following MIF fragment illustrates `SetOptions` usage including a vector field filename script that truncates the iteration count on stage events:

```
proc ShortStageVF { params } {
   set filename [DefaultFieldFilename $params]
   set event [dict get $params event]
   if {[string compare Stage $event] == 0} {
      # Strip trailing iteration count (but retaining extension)
      regsub -- {-[0-9]+(\.[a-zA-Z]+)$} $filename {\1} filename
   }
   return $filename
}

SetOptions {
 basename fubar
 scalar_output_format %.12g
 scalar_field_output_format {text %.4g}
 scalar_field_output_meshtype irregular
 vector_field_output_format {binary 4}
 vector_field_output_filename_script ShortStageVF
}
```

DefaultFieldFilename Can be called from inside an output filename script with the `$params` import to retrieve the MIF default filename. See the SetOptions section for details and an example.

EvalScalarField This command allows access in a MIF file to values from a scalar field defined in a preceding `Specify` block. For example,

```
Oxs_AtlasScalarField:Ms {
   atlas :atlas
   default_value 0
   values {
      Adisks 520e3
      Bdisks 520e3
   }
}}

set Ms_a [EvalScalarField :Ms 50e-9 20e-9 2e-9]
```

The four arguments to `EvalScalarField` are a reference to the scalar field (here `:Ms`), and the three coordinates of the point where you want the field evaluated. The coordinates are in the problem coordinate space, i.e., in meters.

EvalVectorField This command is the same as the `EvalScalarField` command, except that the field reference is to a vector field, and the return value is a three item list representing the three components of the vector field at the specified point.

GetAtlasExtent This command takes one or two arguments. The first a reference to an atlas, and the optional second argument is the name of a region inside that atlas. The return is a 6-tuple, `xmin ymin zmin xmax ymax zmax`, defining the corner points of the axes-parallel bounding box for either the atlas as a whole (one argument) or the specified region (two arguments). Sample usage:

```
lassign [GetAtlasExtent :atlas] xmin ymin zmin xmax ymax zmax
Report "Region oct000 extents: [GetAtlasExtent :atlas oct000]"
```

GetAtlasRegions This command takes one argument, which is a reference to an atlas, and returns an ordered list of all the regions in that atlas. The first item on the returned list will always be "universe", which includes all points not in any of the other regions, including in particular any points outside the nominal bounds of the atlas. Sample usage:

```
set regions_list [GetAtlasRegions :atlas]
```

GetAtlasRegionByPosition This command takes four arguments: a reference to atlas, followed by the x, y, and z coordinates of a point using problem coordinates (i.e.,

meters). The return value is the name of the region containing the specified point. This name will match exactly one of the names on the list returned by the `GetAtlasRegions` command for the given atlas. Note that the return value might be the "universe" region. Sample usage:

```
set rogue [GetAtlasRegionByPosition :atlas 350e-9 57e-9 8e-9]
```

17.4.3 Sample MIF 2.2 File

```
# MIF 2.2

###############
# Constants
set pi [expr 4*atan(1.0)]
set mu0 [expr 4*$pi*1e-7]


###############
# Command-line controls
Parameter seed 1
Parameter thickness 6e-9
Parameter stop 1e-2

# Texturing angle, phideg, in degrees, from 0 to 90; 0 is all z.
Parameter phideg 10;


###############
# Output options
SetOptions [subst {
   basename "polyuniaxial_phi_$phideg"
   scalar_output_format %.12g
   scalar_field_output_format {text %.4g}
   scalar_field_output_meshtype irregular
   vector_field_output_format {binary 4}
}]


###############
# Rogue grain:
# If RoguePt is an empty string, then no rogue grain is selected.  OTOH,
# If RoguePt is set to a three item list consisting of x, y, and z coords
#   in the problem coordinate system (i.e., in meters), then the grain
```

```
#   containing that point is individually set as specified below.
Parameter RoguePt {263.5e-9 174.5e-9 3e-9}


###############
# Support procs:
proc Ellipse { Ms x y z} {
   set x [expr {2*$x-1.}]
   set y [expr {2*$y-1.}]
   if {$x*$x+$y*$y<=1.0} {
      return $Ms
   }
   return 0.0
}


###############
# Material constants
set Ms 1.40e6
set Ku 530e3
set A  8.1e-12


###############
# Atlas and mesh
set xsize 400e-9
set ysize 400e-9
set xycellsize 1.0e-9
set zcellsize  3.0e-9

set grain_count 260
set grain_map polycrystal-map-mif.ppm

set colormap {}
for {set i 0} {$i<$grain_count} {incr i} {
   lappend colormap [format "#%06x" $i]
   lappend colormap $i
}

Specify Oxs_ImageAtlas:world [subst {
   xrange {0 $xsize}
   yrange {0 $ysize}
```

```
   zrange {0 $thickness}
   viewplane xy
   image $grain_map
   colormap {
      $colormap
   }
   matcherror 0.0
}]

Specify Oxs_RectangularMesh:mesh [subst {
   cellsize {$xycellsize $xycellsize $zcellsize}
   atlas :world
}]


#################################
# Uniaxial Anisotropy

# Generate TEXTURED random unit vector
set phirange [expr {1-cos($phideg*$pi/180.)}]
proc Texture {} {
   global pi phirange

   set theta [expr {(2.*rand()-1.)*$pi}]
   set costheta [expr {cos($theta)}]
   set sintheta [expr {sin($theta)}]

   set cosphi [expr {1.-$phirange*rand()}]
   set sinphi [expr {1.0-$cosphi*$cosphi}]
   if {$sinphi>0.0} { set sinphi [expr {sqrt($sinphi)}] }

   set x [expr {$sinphi*$costheta}]
   set y [expr {$sinphi*$sintheta}]
   set z [expr {$cosphi}]

   return [list $x $y $z]
}


# Set a random unit vector for each grain region
set axes {}
for {set i 0} {$i<$grain_count} {incr i} {
```

```
   lappend axes $i
   lappend axes [Texture]
}

# Sets the rogue grain ($Rogue < $grain_count)
if {[llength $RoguePt] == 3} {
   # The :Regions field maps region name (which is a number)
   # to the corresponding number.
   set regionmap {}
   for {set i 0} {$i<$grain_count} {incr i} {lappend regionmap $i $i }
   Specify Oxs_AtlasScalarField:Regions [subst {
      atlas :world
      values [list $regionmap]
   }]
   foreach {x y z} $RoguePt { break }
   set Rogue [EvalScalarField :Regions $x $y $z]
   set item_number [expr 2*$Rogue+1]
   set axes [lreplace $axes $item_number $item_number {1 0 0}]
}

Specify Oxs_AtlasVectorField:axes [subst {
   atlas :world
   norm 1.0
   values [list $axes]
}]

Specify Oxs_UniaxialAnisotropy [subst {
   K1 $Ku
   axis :axes
}]


##################################
# Exchange
set A_list {}
for {set i 0} {$i<$grain_count} {incr i} {
   lappend A_list $i $i $A
}

Specify Oxs_Exchange6Ngbr [subst {
   default_A $A
   atlas world
```

```
   A   [list $A_list]
}]


##################################
# Zeeman (applied) field
set field 10000         ;# Maximum field (in Oe)
Specify Oxs_UZeeman [subst {
   multiplier [expr (1./($mu0*1e4))*$field]
   Hrange  {
      { 0 0 0   0 0 1   10}
   }
}]


##################################
# Driver and Evolver

Specify Oxs_CGEvolve:evolve {}

Specify Oxs_MinDriver [subst {
   evolver evolve
   stopping_mxHxm $stop
   mesh :mesh
   Ms { Oxs_ScriptScalarField {
      atlas :world
      script_args {relpt}
      script {Ellipse $Ms}
   } }
   m0 { 0 0 -1 }
}]
```

Figure 17.3: Example MIF 2.2 file. (description)

17.5 Tips for writing MIF 2.x files

MIF 2.x files are Tcl scripts, and so composing a MIF file is a programming exercise, with all the pitfalls that entails. In this section we detail some tips for authoring MIF files.

Generally a good place to start is to find an existing MIF 2.x file, either one you've written earlier or one from the `oommf/app/oxs/examples/` directory, that has some similarity to the

one you want to write. Make a copy of that and start to edit. Alternatively, you can use **mmProbEd** set to MIF 1.2 mode to create a draft problem, and create a MIF 2.x file from the MIF 1.2 output with **mifconvert**. If you need any complex functionality then an elementary understanding of Tcl is essential. In particular, you should be familiar with Tcl lists, arrays, and some of the basic Tcl commands such as `set`, `expr`, `for`, `foreach`, `if/else`, `lrange`, and `subst`. You can find a list of Tcl tutorials on the Tcler's Wiki Online Tcl and Tk Tutorials[1] page. The *Tcl Tutorial* by Chris verBurg and *Learn Tcl in Y Minutes* linked on that page are good places to start. The Tcl Dodekalogue[2] is also a handy reference for figuring out confusing error messages. Many more resources, including online manuals, can be found at the Tcl Developer Xchange[3].

Although MIF files are Tcl scripts, they also rely on a number of MIF extension commands. The most prominent of these is the `Specify` command, used to initialize `Oxs_Ext` (Oxs extension) objects, such as region definitions (atlases), mesh discretization, energy terms (exchange, anisotropy, dipole-dipole, applied fields), and solution method (energy minimization or LLG integration). The contents of each `Specify` block, which depend upon the particular `Oxs_Ext` being initialized, are documented Standard Oxs_Ext Child Classes section.

Other commonly used MIF extension commands are `Parameter`, `Destination`, and `Schedule`. `Parameter` operates similar to the Tcl `set` command, but allows the user to change the default value at runtime, through either the `Params` box in the `Oxsii Load Problem` dialog, or via the `-parameters` command line option to `Boxsi`. The `Destination` and `Schedule` commands are used primarily for non-interactive set up of simulation output for `Boxsi`.

As in any programming activity, bugs happen. Here are a few tips to aid in debugging your MIF code:

0. Check to see if your favorite programming editor (FPE) has support for Tcl syntax highlighting and indenting. This can help spot a lot of errors, particularly brace nesting issues, before you hand the file over to the Tcl interpreter inside OOMMF.

1. For development purposes, set the problem dimensions and cell sizes so the total number of cells is relatively small. This will allow problems to load and run faster, making it easier to detect and correct errors. Use the `Oxsii|Help|About` menu to see the number of cells in the simulation. For larger simulations you may want to temporarily disable the `Oxs_Demag` module, because that module can take some time to initialize and is usually the slowest module to run. If the cellsize magnetization is set correctly, then `Oxs_Demag` will usually be okay; you might want to take a little extra care when working with periodic boundary conditions, however, because demagnetization effects across PBC's can be nonintuitive.

[1] https://wiki.tcl-lang.org/page/Online+Tcl+and+Tk+Tutorials
[2] https://wiki.tcl-lang.org/page/Dodekalogue
[3] https://www.tcl-lang.org/

2. When the MIF file is ready, launch `Oxsii` and bring up the `File|Load` dialog box. Select the `Browse` check box, as otherwise the `Load` dialog box is automatically closed when you select `OK` to load the file. The `Browse` option allows you to correct syntax errors in your MIF file without having to repeatedly open the `File|Load` dialog box and reselect your MIF file and parameters.

3. If the load fails, open the MIF file in your FPE and use the error message to help locate the point of failure in the file. Correct, save to disk, and try loading again.

4. If you can't figure out an error message, or to just double-check that your file is being interpreted as you intend, add MIF `Report` commands to the file to print the contents of variables or other state information to the `Oxsii` console. (The `Oxsii` console is launched via the `File|Show Console` menu item on the main `Oxsii` window. If you are running `Boxsi`, `Report` output goes to the `Boxsi` log file, `oommf/boxsi.log`.)

5. Once the file loads without errors, send `Magnetization` output from `Oxsii` to `mmDisp` for a visual check of the simulation structure. Use `Ctrl-` or `Shift-Ctrl-<left mouse click>` in `mmDisp` to view magnetization component values at various locations. Bring up the `mmDisp Options|Configure` dialog box to adjust the coloring, pixel and arrow selection, etc. The `# of Colors` and `Data Scale` settings in particular can make it easier to see small differences in the selected `Color Quantity`. The `Arrow span` setting can be used to control the number of levels of arrows that get displayed in the slice view. For example, if the cell dimension in the out-of-view-plane direction is 4 nm, then setting `Arrow span` to `4e-9` will cause a single slice of the magnetization to be displayed as you adjust the slice control slider in the main `mmDisp` window. If you set `Arrow span` to `8e-9` you'll see two layers of overlapping arrows, which can be helpful for checking interface conditions.

6. By default `mmDisp` opens with a top-down view along the z-axis. For multilayer structures it is helpful to see cross-sectional views along other axes too; these are available from the `View|Viewpoint` submenu in the main `mmDisp` window. Also, the `mmDisp` rendering of high aspect ratio cells can be rather poor. In this situation you may want to enable pixel display in the `mmDisp` configuration dialog with pixel size set smaller than 1, and adjust the background color to make the individual cells visible.

7. The `Magnetization` output viewed in `mmDisp` allows you to check that you have set `Ms` and `m0` properly in the driver `Specify` block. You can send other fields such as anisotropy and exchange to `mmDisp` to check parameter settings for those energy terms as well. In this context it can be helpful to adjust `m0` to align in specific directions for testing. You may want to open two instances of `mmDisp`, with `Magnetization` displayed in one and an energy field in the other, to help correspond energy field values with magnetization structure.

8. If you are having problems implementing some functionality, do a search through the MIF files in `oommf/app/oxs/examples` for something similar. For example, a

text search for "pulse" will turn up matches in `pillar.mif`, `pingpillar.mif`, and `pulse.mif`, each illustrating how to implement an applied field pulse with various temporal profiles.

17.6 PyMIF 1.0

The PyMIF 1.0 (aka MIF/Python) format, introduced in OOMMF 2.1a2, uses Python as the underlying language as opposed to the MIF 2.x formats which are Tcl-based.

17.6.1 Introduction to PyMIF 1.0

The MIF 2.x (MIF/Tcl) formats discussed in the previous sections prescribe an OOMMF micromagnetic simulation in terms of its constitutive objects (`Oxs_Ext` elements representing energies and fields, problem geometry, system evolution, inputs and outputs) laid out and assembled by Tcl script executed by a Tcl interpreter embedded inside the Oxs solver. In practice, this means that MIF 2.x files are actual Tcl scripts extended with a handful of pre-defined Tcl procs such as `Specify`, `Destination`, and `Schedule`.

The concept behind the PyMIF 1.0 (MIF/Python) format is similar, but with Python replacing Tcl, and pre-defined Python functions instead of Tcl procs. The extension functions currently supported include `Destination`, `Parameter`, `RandomSeed`, `Specify`, `Schedule` and `SetOptions`. Each of these behave in the same manner as their namesake in MIF/Tcl (cf. MIF 2.1 and MIF 2.2 extension command sections) but with call signatures distinguished by differences between Tcl and Python language syntax. For example,

```
Destination table mmDataTable
Schedule DataTable table Step 5
```

in MIF/Tcl becomes

```
Destination('table', 'mmDataTable')
Schedule('DataTable', 'table', 'Step', 5)
```

in MIF/Python. A number of example MIF/Python files are included with the OOMMF distribution in the the directory `oommf/app/oxs/examples_py/`.

The `Specify` and `SetOptions` commands, however, have a somewhat less transparent translation. These commands take name + value pairs as imports. In MIF/Tcl these are handled as Tcl lists with an even number of elements. In MIF/Python the name + value pairs are represented by dictionaries. For example

```
SetOptions {
   basename acsample
   scalar_output_format "%.12g"
}
```

in MIF/Tcl is

```
SetOptions({'basename'    : 'acsample'},
   'scalar_output_format' : '%.12g'})
```

in MIF/Python. There is a slight twist with `Specify`, where the MIF/Tcl syntax

```
Specify Oxs_UniformExchange:uexch {
  A  13e-12
}
```

sets the class and instance name as a separate first parameter, followed by the name + value pair list. In the MIF/Python `Specify` call these are rolled into the dictionary argument:

```
Specify({'class' : 'Oxs_UniformExchange', 'instance' : 'uexch',
   'A' : 13e-12
})
```

A more complex example involving sub-lists is

```
Parameter cellsize 10                ;# Cell size in nm
set cellsize [expr {$cellsize*1e-9}] ;# Convert to meters
Specify Oxs_RectangularMesh:mesh [subst {
  cellsize {$cellsize $cellsize $cellsize}
  atlas :atlas
}]
```

This MIF/Tcl script becomes

```
Parameter('cellsize', 10)   # Cell size in nm
cellsize *= 1e-9            # Convert to meters
Specify({'class' : 'Oxs_RectangularMesh', 'instance' : 'mesh',
   'cellsize' : [cellsize, cellsize, cellsize],
   'atlas' : ':atlas'
})
```

in MIF/Python. Note in particular how the Tcl list

```
{$cellsize $cellsize $cellsize}
```

is replaced with the corresponding Python list object

```
[cellsize, cellsize, cellsize]
```

Despite the similarities, MIF/Tcl and MIF/Python 1.0 have a significant functional difference in that the latter is not processed by a Python interpreter embedded in `Oxs`, but is rather converted from PyMIF to MIF/Tcl by a standalone Python executable running the `mifpy2tcl.py` script. The `Oxs` executables `Oxsii` and `Boxsi` run `mifpy2tcl.py` on the fly as needed to process PyMIF scripts, but `mifpy2tcl.py` can also be run outside of `Oxs`. This allows one to preview and modify the MIF/Tcl version before feeding it back to `Oxs` for native processing.

The downside of this is that the Python processing is completed before the `Oxs` simulation proper begins, so it is not possible to have Python routines active inside an OOMMF simulation. This is particularly noticeable in simulations that leverage a Tcl proc to provide a time varying field pulse, such as the MIF/Tcl examples `acsample.mif` and `pulse.mif`. The work-around provided by MIF/Python 1.0 is a wrapper function, `TclCode`, that accepts Tcl script as a string which is passed verbatim from the MIF/Python file to the MIF/Tcl conversion. For example, the PyMIF content

```
TclCode('''
proc SineField { total_time } {
   set amp 8e4
   set freq 5e9
   set Hx [expr {$amp*sin($freq*$total_time)}]
   set dHx [expr {$amp*$freq*cos($freq*$total_time)}]
   return [list $Hx 0 0 $dHx 0 0]
}
''')
Specify({'class' : 'Oxs_ScriptUZeeman',
         'script_args' : 'total_time',
         'script' : 'SineField'})
```

is converted into

```
proc SineField { total_time } {
   set amp 8e4
   set freq 5e9
   set Hx [expr {$amp*sin($freq*$total_time)}]
   set dHx [expr {$amp*$freq*cos($freq*$total_time)}]
   return [list $Hx 0 0 $dHx 0 0]
}
Specify Oxs_ScriptUZeeman {
 script_args total_time
 script SineField
}
```

for MIF/Tcl.

Performing variable substitutions within the `TclCode` string at the PyMIF script level, perhaps to support command line `Parameter`s, is complicated by syntax collisions between Tcl and Python, particularly involving curly braces ({, }). The Python `string.Template` class can be employed to bypass this obstacle. See the `acsample_py.mif` example file in `oommf/app/oxs/examples_py/` for specifics.

Note: PyMIF files are passed to a Python engine for evaluation in the same manner as other Python `.py` scripts, and should be treated in a like manner.

17.6.2 Sample PyMIF 1.0 File

Note: The first line of a PyMIF 1.0 file must be "# MIF/Python 1.0".

```
# MIF/Python 1.0
# MIF Example File: sample.mif
# Description: Hysteresis loop using Runge-Kutta evolver
#     and time driver.

from math import pi
mu0 = 4*pi*1e-7

RandomSeed(1)

Parameter('cellsize', 5e-9)

Specify({'class' : 'Oxs_BoxAtlas', 'instance' : 'atlas',
   'xrange' : [0, 200e-9],
   'yrange' : [0, 100e-9],
   'zrange' : [0, 20e-9]
})

Specify({'class' : 'Oxs_RectangularMesh', 'instance' : 'mesh',
   'cellsize' : [cellsize, cellsize, cellsize],
   'atlas' : ':atlas'
})

Specify({'class' : 'Oxs_CubicAnisotropy',
   'K1' : -4.5e3,
   'axis1' : [1, 0, 0],
   'axis2' : [0, 1, 0]
})

Specify({'class' : 'Oxs_UniformExchange',
   'A' : 9e-12,
   'comment' : "In the line above 'A' is a single scalar, not a scalar field"
})

Specify({'class' : 'Oxs_UZeeman',
   'multiplier' : 0.001/mu0,
   'Hrange' : [[  0,  0,  0,  100,   1,   0,  2],
               [100,  1,  0, -100,   0,   0,  2],
               [  0,  0,  0,    0, 100,   0,  4],
```

```
                [ 10, 10, 10,   50,  50,  50,  0]]
})

Specify({'class' : 'Oxs_Demag'})

Specify({'class' : 'Oxs_RungeKuttaEvolve', 'instance' : 'evolve'})

SetOptions({'basename' : 'sample'})

Specify({'class' : 'Oxs_TimeDriver',
   'evolver' : ':evolve',
   'stopping_dm_dt' : 0.1,
   'mesh' : ':mesh',
   'Ms' : 490e3,
   'm0' : [1, 0.02, 0.01]
})
```

Figure 17.4: Example PyMIF 1.0 file. (description)

Chapter 18

Data Table File Format (ODT)

Textual output from solver applications that is not of the vector field variety is output in the *OOMMF Data Table* (ODT) format. This is an ASCII text file format, with column information in the header and one line of data per record. Any line ending in a '\' character is joined to the succeeding line before any other processing is performed. Any leading '#' characters on the second line are removed.

As with the OVF format, all non-data lines begin with a '#' character, comments with two '#' characters. (This makes it easier to import the data into external programs, for example, plotting packages.) An example is included below.

The first line of an ODT file should be the file type descriptor

```
# ODT 1.0
```

It is also recommended that ODT files be given names ending in the file extension `.odt` so that ODT files may be easily identified.

The remaining lines of the ODT file format should be comments, data, or any of the following 5 recognized descriptor tag lines:

- **# Table Start:** Optional, used to segment a file containing multiple data table blocks. Anything after the colon is taken as an optional label for the following data block.
- **# Title:** Optional; everything after the colon is interpreted as a title for the table.
- **# Columns:** Required. One parameter per column, designating the label header for that column. Spaces may be embedded in a column label by using the normal Tcl grouping mechanisms (i.e., double-quotes and braces).
- **# Units:** Optional. If given, it should have one parameter for each column, giving a unit label for the corresponding column. Spaces may be embedded in the unit labels, in the same manner as for column headers.

```
# ODT 1.0
# Table Start
# Title: This is a small sample ODT file.
#
## This is a sample comment.  You can put anything you want
## on comment lines.
#
# Columns: Iteration "Applied Field"  {Total Energy}    Mx
# Units:      {}          "mT"           "J/m^3"       "A/m"
              103          50            0.00636      787840
             1000          32            0.00603      781120
            10300       -5000            0.00640     -800e3
# Table End
```

Figure 18.1: Sample ODT file. (description)

- **# Table End**: Optional, no parameters. Should be paired with a corresponding Table Start record.

Data may appear anywhere after the Columns descriptor record and before any Table End line, with one record per line. The data should be numeric values separated by whitespace. The two character open-close curly brace pair, {}, is used to indicate a missing value.

Embedded newlines and carriage returns are not allowed in the title, columns, or units records.

The command line utility, **odtcols**, can be a useful tool for examining and partitioning ODT files.

Chapter 19

Vector Field File Format (OVF)

Vector field files specify vector quantities (e.g., magnetization or magnetic flux density) as a function of spatial position. The *OOMMF Vector Field* (OVF) format is the output vector field file format used by both the 2D and 3D micromagnetic solvers. It is also the input data type read by **mmDisp**. There are three versions of the OVF format supported by OOMMF. The OVF 1.0 and 2.0 formats are preferred formats and the only ones written by OOMMF software. They support both rectangular and irregular meshes, in binary and ASCII text.

The OVF 0.0 format (formerly SVF) is an older, simpler format that can be useful for importing three-dimensional vector field data into OOMMF from other programs. (A fourth format, the *VecFil* or *Vector Input/Output* (VIO) format, was used by some precursors to the OOMMF code. Although OOMMF is able to read the VIO format, its use is deprecated.)

In all these formats, the field domain (i.e., the spatial extent) lies across three dimensions, with units typically expressed in meters or nanometers. In all the formats except the OVF 2.0 format, the field values are also three dimensional; the value units are more varied, but are most often Tesla or A/m. The OVF 2.0 format is more general, in that the field values can be of any arbitrary dimension $N > 0$. (This dimension, however, is fixed within the file.) If $N = 3$, then the OVF 2.0 format supports the same types of data as the OVF 1.0 format, but another common case is $N = 1$, which represents scalar fields, such as energy density (in say, J/m^3).

The recommended file extensions for OVF files are `.omf` for magnetization files, `.ohf` for magnetic field (**H**) files, `.obf` for magnetic flux density (**B**) files, `.oef` for energy density files, or `.ovf` for generic files.

19.1 The OVF 0.0 format

The OVF 0.0 format is a simple ASCII text format supporting irregularly sampled data. It is intended as an aid for importing data from non-OOMMF programs, and is backwards compatible with the format used for problem submissions for the first μMAG standard

problem[1].

Users of early releases of OOMMF may recognize the OVF 0.0 format by its previous name, the Simple Vector Field (SVF) format. It came to the attention of the OOMMF developers that the file extension `.svf` was already registered in several MIME systems to indicate the Simple Vector Format[2], a vector graphics format. To avoid conflict, we have stopped using the name Simple Vector Field format, although OOMMF software still recognizes the `.svf` extension and you may still find example files and other references to the SVF format.

A sample OVF 0.0 file is shown below. Any line beginning with a '`#`' character is a comment, all others are data lines. Each data line is a whitespace separated list of 6 elements: the x, y and z components of a node position, followed by the x, y and z components of the field at that position. Input continues until the end of the file is reached.

It is recommended (but not required) that the first line of an OVF file be

```
# OOMMF: irregular mesh v0.0
```

This will aid automatic file type detection. Also, three special (extended) comments in OVF 0.0 files are recognized by **mmDisp**:

```
## File: <filename or extended filename>
## Boundary-XY: <boundary vertex pairs>
## Grid step: <cell dimension triple>
```

All these lines are optional. The "File" provides a preferred (possibly extended) filename to use for display identification. The "Boundary-XY" line specifies the ordered vertices of a bounding polygon in the xy-plane. If given, **mmDisp** will draw a frame using those points to ostensibly indicate the edges of the simulation body. Lastly, the "Grid step" line provides three values representing the average x, y and z dimensions of the volume corresponding to an individual node (field sample). It is used by **mmDisp** to help scale the display.

Note that the data section of an OVF 0.0 file takes the simple form of columns of ASCII formatted numbers. Columns of whitespace separated numbers expressed in ASCII are easy to import into other programs that process numerical datasets, and are easy to generate, so the OVF 0.0 file format is useful for exchanging vector field data between OOMMF and non-OOMMF programs. Furthermore, the data section of an OVF 0.0 file is consistent with the data section of an OVF 1.0 file that has been saved as an irregular mesh using text data representation. This means that even though OOMMF software now writes only the OVF 1.0 format for vector field data, simple interchange of vector field data with other programs is still supported.

[1] https://www.ctcms.nist.gov/%7Erdm/stdprob_1.html
[2] http://www.softsource.com/svf/

```
# OOMMF: irregular mesh v0.0
## File: sample.ovf
## Boundary-XY: 0.0 0.0 1.0 0.0 1.0 2.0 0.0 2.0 0.0 0.0
## Grid step: .25 .5 0
#  x      y      z        m_x      m_y      m_z
  0.01   0.01   0.01   -0.35537  0.93472 -0.00000
  0.01   1.00   0.01   -0.18936  0.98191 -0.00000
  0.01   1.99   0.01   -0.08112  0.99670 -0.00000
  0.50   0.50   0.01   -0.03302  0.99945 -0.00001
  0.99   0.05   0.01   -0.08141  0.99668 -0.00001
  0.75   1.50   0.01   -0.18981  0.98182 -0.00000
  0.99   1.99   0.01   -0.35652  0.93429 -0.00000
```

Figure 19.1: Example OVF 0.0 file. (description)

19.2 The OVF 1.0 format

A commented sample OVF 1.0 file is provided below. An OVF file has an ASCII header and trailer, and a data block that may be either ASCII or binary. All non-data lines begin with a '`#`' character; double '`##`' mark the start of a comment, which continues until the end of the line. There is no line continuation character. Lines starting with a '`#`' but containing only whitespace characters are ignored.

All non-empty non-comment lines in the file header are structured as label+value pairs. The label tag consists of all characters after the initial '`#`' up to the first colon (':') character. Case is ignored, and all space and tab characters are eliminated. The value consists of all characters after the first colon, continuing up to a '`##`' comment designator or the end of the line.

The first line of an OVF file should be a file type identification line, having the form

```
# OOMMF: rectangular mesh v1.0
```

or

```
# OOMMF: irregular mesh v1.0
```

where the value "rectangular mesh v1.0" or "irregular mesh v1.0" identifies the mesh type and revision. While the OVF 1.0 format was under development in earlier OOMMF releases, the revision strings `0.99` and `0.0a0` were sometimes recorded on the file type identification line. OOMMF treats all of these as synonyms for `1.0` when reading OVF files.

The remainder of the file is conceptually broken into Segment blocks, and each Segment block is composed of a (Segment) Header block and a Data block. Each block begins with a "`# Begin: <block type>`" line, and ends with a corresponding "`# End: <block type>`" line. The number of Segment blocks is specified in the

```
# Segment count: 1
```

line. Currently only 1 segment is allowed. This may be changed in the future to allow for multiple vector fields per file. This is followed by

```
# Begin: Segment
```

to start the first segment.

19.2.1 Segment Header block

The Segment Header block start is marked by the line "`#` Begin: Header" and the end by "`#` End: Header". Everything between these lines should be either comments or one of the following file descriptor lines. They are order independent. All are required unless otherwise stated. Numeric values are floating point values unless "integer" is explicitly stated.

- **title:** Long filename or title.
- **desc:** Description line. Optional. Use as many as desired. Description lines may be displayed by post-processing programs, unlike comment lines which are ignored by all automated processing.
- **meshunit:** Fundamental mesh spatial unit, treated as a label. The comment marker '`##`' is not allowed in this label. Example value: "nm".
- **valueunit:** Fundamental field value unit, treated as a label. The comment marker '`##`' is not allowed in this label. Example: "kA/m."
- **valuemultiplier:** Values in the data block are multiplied by this to get true values in units of "valueunit." This simplifies the use of normalized values.
- **xmin, ymin, zmin, xmax, ymax, zmax:** Six separate lines, specifying the bounding box for the mesh, in units of "meshunit." This may be used by display programs to limit the display area, and may be used for drawing a boundary frame if "boundary" is not specified.
- **boundary:** List of *(x,y,z)* triples specifying the vertices of a boundary frame. Optional.
- **ValueRangeMaxMag, ValueRangeMinMag:** The maximum and minimum field magnitudes in the data block, in the same units and scale as used in the data block. These are for optional use as hints by postprocessing programs; for example, **mmDisp** will not display any vector with magnitude smaller than ValueRangeMinMag. If both ValueRangeMinMag and ValueRangeMaxMag are zero, then the values should be ignored.

- **meshtype:** Grid structure; should be either "rectangular" or "irregular." Irregular grid files should specify "pointcount" in the header; rectangular grid files should specify instead "xbase, ybase, zbase," "xstepsize, ystepsize, zstepsize," and "xnodes, ynodes, znodes."
- **pointcount:** Number of data sample points/locations, i.e., nodes (integer). For irregular grids only.
- **xbase, ybase, zbase:** Three separate lines, denoting the position of the first point in the data section, in units of "meshunit." For rectangular grids only.
- **xstepsize, ystepsize, zstepsize:** Three separate lines, specifying the distance between adjacent grid points, in units of "meshunit." Required for rectangular grids, but may be specified as a display hint for irregular grids.
- **xnodes, ynodes, znodes:** Three separate lines, specifying the number of nodes along each axis (integers). For rectangular grids only.

19.2.2 Data block

The data block start is marked by a line of the form

```
# Begin: data <representation>
```

where `<representation>` is one of "text", "binary 4", or "binary 8". Text mode uses the ASCII specification, with individual data items separated by an arbitrary amount of whitespace (spaces, tabs and newlines). Comments are not allowed inside binary mode data blocks, but are permitted inside text data blocks. To portably detect the start of a binary data block, the end of line marker should follow either the Unix and macOS convention of a simple newline, or else the Windows carriage return + newline pair.

The binary representations are IEEE floating point in network byte order (MSB). To ensure that the byte order is correct, and to provide a partial check that the file hasn't been sent through a non 8-bit clean channel, the first datum is a predefined value: 1234567.0 (Hex: 49 96 B4 38) for 4-byte mode, and 123456789012345.0 (Hex: 42 DC 12 21 83 77 DE 40) for 8-byte mode. The data immediately follow the check value.

The structure of the data depends on whether the "meshtype" declared in the header is "irregular" or "rectangular". For irregular meshes, each data element is a 6-tuple, consisting of the x, y and z components of the node position, followed by the x, y and z components of the field at that position. Ordering among the nodes is not relevant. The number of nodes is specified in the "pointcount" line in the segment header.

For rectangular meshes, data input is field values only, in x, y, z component triples. These are ordered with the x index incremented first, then the y index, and the z index last. This is nominally Fortran order, and is adopted here because commonly x will be the longest dimension, and z the shortest, so this order is more memory-access efficient than the normal

C array indexing of z, y, x. The size of each dimension is specified in the "xnodes, ynodes, znodes" lines in the segment header.

In any case, the first character after the last data item should be a newline, followed by

```
# End: data <representation>
```

where `<representation>` must match the value in the "Begin: data" line. This is followed by a

```
# End: segment
```

line that ends the segment, and hence the file.

Note: An OVF 1.0 file with ASCII data and irregular meshtype is also a valid OVF 0.0 (SVF) file, although as a OVF 0.0 file the value scaling as specified by "`# valueunit`" and "`# valuemultiplier`" header lines is inactive.

```
# OOMMF: rectangular mesh v1.0
#
## This is a comment.
## No comments allowed in the first line.
#
# Segment count: 1   ## Number of segments.  Should be 1 for now.
#
# Begin: Segment
# Begin: Header
#
# Title: Long filename or title goes here
#
# Desc: 'Description' tag, which may be used or ignored by postprocessing
# Desc: programs. You can put anything you want here, and can have as many
# Desc: 'Desc' lines as you want.  The ## comment marker is disabled in
# Desc: description lines.
#
## Fundamental mesh measurement unit.  Treated as a label:
# meshunit: nm
#
# meshtype: rectangular
# xbase: 0.      ## (xbase,ybase,zbase) is the position, in
# ybase: 0.      ## 'meshunit', of the first point in the data
# zbase: 0.      ## section (below).
#
# xstepsize: 20. ## Distance between adjacent grid pts.: on the x-axis,
# ystepsize: 10. ## 20 nm, etc.  The sign on this value determines the
# zstepsize: 10. ## grid orientation relative to (xbase,ybase,zbase).
```

```
#
# xnodes: 200    ## Number of nodes along the x-axis, etc. (integers)
# ynodes: 400
# znodes:   1
#
# xmin:    0.    ## Corner points defining mesh bounding box in
# ymin:    0.    ## 'meshunit'.  Floating point values.
# zmin:  -10.
# xmax: 4000.
# ymax: 4000.
# zmax:   10.
#
## Fundamental field value unit, treated as a label:
# valueunit: kA/m
# valuemultiplier: 0.79577472  ## Multiply data block values by this
#                              ## to get true value in 'valueunits'.
#
# ValueRangeMaxMag:  1005.3096  ## These are in data block value units,
# ValueRangeMinMag:  1e-8       ## and are used as hints (or defaults)
#     ## by postprocessing programs.  The mmDisp program ignores any
#     ## points with magnitude smaller than ValueRangeMinMag, and uses
#     ## ValueRangeMaxMag to scale inputs for display.
#
# End: Header
#
## Anything between '# End: Header' and '# Begin: data text',
## '# Begin: data binary 4' or '# Begin: data binary 8' is ignored.
##
## Data input is in 'x-component y-component z-component' triples,
## ordered with x incremented first, then y, and finally z.
#
# Begin: data text
1000 0 0 724.1 0. 700.023
578.5 500.4 -652.36
<...data omitted for brevity...>
252.34 -696.42 -671.81
# End: data text
# End: segment
```

Figure 19.2: Commented OVF sample file. (description)

19.3 The OVF 2.0 format

The OVF 2.0 format is a modification to the OVF 1.0 format that also supports fields across three spatial dimensions but having values of arbitrary (but fixed) dimension. In the OVF 2.0 format:

1. The first line reads: `# OOMMF OVF 2.0` for both regular and irregular meshes.

2. In the Segment Header block, the new record `valuedim` is required. This must specify an integer value, N, bigger or equal to one.

3. In the Segment Header block, the new `valueunits` record replaces the `valueunit` record of OVF 1.0. Instead of a single unit value, `valueunits` should be a (Tcl) list of value units, each treated as an unparsed label. The list should either have length N (as specified by `valuedim`), in which case each element denotes the units for the corresponding dimension index, or else the list should have length one, in which case the single element is applied to all dimension indexes. The old `valueunit` record is not allowed in OVF 2.0 files.

4. In the Segment Header block, the new `valuelabels` record is required. This should be an N-item (Tcl) list of value labels, one for each value dimension. The labels identify the quantity in each dimension. For example, in an energy density file, N would be 1, `valueunits` could be J/m^3, and `valuelabels` might be "Exchange energy density".

5. In the Segment Header block, the `valuemultiplier`, `boundary`, `ValueRangeMaxMag` and `ValueRangeMinMag` records of the OVF 1.0 format are not supported.

6. In the Data block, for regular meshes each record consists of N values, where N is the value dimension as specified by the `valuedim` record in the Segment Header. The node ordering is the same as for the OVF 1.0 format. For irregular meshes, each record consists of $N + 3$ values, where the first three values are the x, y and z components of the node position. For data blocks using text representation with $N = 3$, the Data block in OVF 1.0 and OVF 2.0 files are exactly the same.

7. The data layout for data blocks using binary representation is also the same as in OVF 1.0 files, except that all binary values are written in a little endian (LSB) order, as compared to the MSB order used in the OVF 1.0 format. This includes the initial check value (IEEE floating point value 1234567.0 for 4-byte format, corresponding to the LSB hex byte sequence 38 B4 96 49, and 123456789012345.0 for 8-byte format, corresponding to the LSB hex byte sequence 40 DE 77 83 21 12 DC 42) as well as the subsequent data records.

In all other respects, the OVF 1.0 and OVF 2.0 are the same. An example OVF 2.0 file for an irregular mesh with $N = 2$ follows.

```
# OOMMF OVF 2.0
#
# Segment count: 1
#
# Begin: Segment
# Begin: Header
#
# Title: Long filename or title goes here
#
# Desc: Optional description line 1.
# Desc: Optional description line 2.
# Desc: ...
#
## Fundamental mesh measurement unit.  Treated as a label:
# meshunit: nm
#
# meshtype: irregular
# pointcount: 5      ## Number of nodes in mesh
#
# xmin:    0.    ## Corner points defining mesh bounding box in
# ymin:    0.    ## 'meshunit'.  Floating point values.
# zmin:    0.
# xmax:   10.
# ymax:    5.
# zmax:    1.
#
# valuedim: 2    ## Value dimension
#
## Fundamental field value units, treated as labels (i.e., unparsed).
## In general, there should be one label for each value dimension.
# valueunits:  J/m^3  A/m
# valuelabels: "Zeeman energy density"  "Anisotropy field"
#
# End: Header
#
## Each data records consists of N+3 values: the (x,y,z) node
## location, followed by the N value components.  In this example,
## N+3 = 5, the two value components are in units of J/m^3 and A/m,
## corresponding to Zeeman energy density and a magneto-crystalline
## anisotropy field, respectively.
#
# Begin: data text
```

```
0.5 0.5 0.5  500.  4e4
9.5 0.5 0.5  300.  5e3
0.5 4.5 0.5  400.  4e4
9.5 4.5 0.5  200.  5e3
5.0 2.5 0.5  350.  2.1e4
# End: data text
# End: segment
```

Figure 19.3: Commented OVF 2.0 sample file. (description)

Chapter 20

Integer Field File Format (OIF)

The OIF format is loosely based on the OVF formats, but stores non-negative integer values rather than floating point in its data block. Moreover, the integers are typically relatively small values indexing into a list of labels, rather than representing raw data quantities directly. One client of OIF files is the Oxs_FileAtlas class (Sec. 7.3.1), where the OIF integers map out the locations of distinct regions in the atlas. A sample OIF file is included below.

The OIF file structure consists of an identification record followed by header and data blocks. All non-data lines begin with a '`#`' character. Comments may be embedded inside the header block, with the '`##`' pair denoting the start of a comment. Comments run to the end of the line. The data block consists of non-negative integer values in either plain text or little endian binary form.

The file type identification record must be the first line of an OIF file, with the form

```
# OOMMF OIF 1.0
```

Segment count and begin records, as in the OVF formats, may follow the identification line, but are optional in OIF files and are ignored. The next required record is the header block start record

```
# Begin: Header
```

The header block continues until the header block end record

```
# End: Header
```

All records inside the header block are structured as label+value pairs. The label tag consists of all characters after the initial '`#`' up to the first colon (':') character. Case is ignored, and all space and tab characters are eliminated. The value consists of all characters after the first colon, continuing up to a '`##`' comment designator or the end of the line.

Records inside the header block include

- **meshtype:** Must have the value `rectangular`. Required

- **xnodes, ynodes, znodes:** Three separate lines, specifying the number of nodes along each axis (integer). Required.
- **xbase, ybase, zbase:** Three separate lines, specifying one corner of the volume covered by the file (floating point). Optional.
- **labels:** A single record containing a (Tcl) list of alphanumeric labels associated with values in the data block. This is an optional record that clients may ignore, but if used the first label corresponds to the value 0, the second label to 1, and so on. If supplied, the maximum value in the data block should not be greater than the number of elements in this list.
- **xstepsize, ystepsize, zstepsize:** Three separate lines, specifying the distance between adjacent grid points. Optional. The x/y/zbase and x/y/zstepsize may be ignored by the client, but if used the volume represented runs across

$$\begin{aligned}(\mathrm{xbase}, \mathrm{xbase} + \mathrm{xnodes} \cdot \mathrm{xstepsize}) \\ \times (\mathrm{ybase}, \mathrm{ybase} + \mathrm{ynodes} \cdot \mathrm{ystepsize}) \\ \times (\mathrm{zbase}, \mathrm{zbase} + \mathrm{znodes} \cdot \mathrm{zstepsize})\end{aligned}$$

  with the first node in the data block located at

$$(\mathrm{xbase} + \mathrm{xstepsize}/2, \mathrm{ybase} + \mathrm{ystepsize}/2, \mathrm{zbase} + \mathrm{zstepsize}/2)$$

  and the last at

$$\begin{aligned}(\mathrm{xbase} + (\mathrm{xnodes} - 1/2) \cdot \mathrm{xstepsize}, \\ \mathrm{ybase} + (\mathrm{ynodes} - 1/2) \cdot \mathrm{ystepsize}, \\ \mathrm{zbase} + (\mathrm{znodes} - 1/2) \cdot \mathrm{zstepsize})\,.\end{aligned}$$

The order of records inside the header block does not matter.

The data block follows the header block, and starts with a record of the form

```
# Begin: data text
```

for plain text data or

```
# Begin: data binary n
```

for binary data, where n is 1, 2, or 4, indicating the width in bytes of each item in the data block. (To portably detect the start of a binary data block, the end of line marker should follow either the Unix and macOS convention of a simple newline, or else the Windows carriage return + newline pair.)

In text data blocks, the xnodes · ynodes · znodes data items are whitespace separated. Binary data blocks are more compact, with items distinguished by item width. Multi-byte

binary values are stored with the least significant byte first (little endian). To guard again byte ordering mishaps, and to provide a partial check that the file hasn't been sent through a non 8-bit clean channel, the first item in a binary data block is a check value: 255 (hex 0xFF), 65 306 (hex 0xFF1A), or 83 827 228 (hex 0x04FF1A1C), for 1 byte, 2 byte, and 4 byte binary respectively. (These will read sequentially in the file as 0xFF, 0x1A 0xFF, and 0x1C 0x1A 0xFF 0x04.) Including the check value, each binary data block contains a total of $1 + \text{xnodes} \cdot \text{ynodes} \cdot \text{znodes}$ items.

In all cases the ordering runs with the x index incremented first, then the y index, and the z index last. This is the same ordering as used by the OVF formats. (Note that as compared to images used by the `Oxs_ImageAtlas` class (Sec. 7.3.1), where the y index typically runs from the top to the bottom of the image, the y index in OIF files would be interpreted as running from the "bottom" to the "top".) Text and data blocks both end with a trailing end of line marker (newline or carriage return + newline) followed by either

```
# End: data text
```

or

```
# End: data binary n
```

for text and binary blocks, respectively.

```
# OOMMF OIF 1.0
# Begin: Header
## Two '#' characters denote a comment
# meshtype: rectangular
# xbase: 2.5e-9
# ybase: 2.5e-9
# zbase: 2.5e-9
# xstepsize: 5e-9
# ystepsize: 5e-9
# zstepsize: 4e-9
# labels: universe Fe Ni Co spacer
#
## The x/y/zbase, x/y/zstepsize, and labels records are optional. Each
## may be treated as a hint and ignored by a client application. In
## particular, the Oxs class Oxs_FileAtlas Specify block may override
## these values. However, as an example, if the Oxs_FileAtlas Specify
## block does not include a regionmap entry then 0 in the data section
## below will be identified with the universe region, 1 to Fe, 2 to Ni,
## 3 to Co, and 4 to spacer. The data are read with from start to finish
## with increasing x values first, then y, and finally z. So in this
## example the bottom layer is solid Fe, and the top layer has Ni on the
## left and Co on the right. The run of zeros in the middle layer
```

```
## delineates a "universe" region across the maximum y value.
#
## The x/y/znodes records are required. The product
##    xnodes * ynodes * znodes
## equals the number of entries in the data block.
#
# xnodes: 6
# ynodes: 4
# znodes: 3
# End: Header
# Begin: data text
1  1  1  1  1  1
1  1  1  1  1  1
1  1  1  1  1  1
1  1  1  1  1  1

4  4  4  4  4  4
4  4  4  4  4  4
4  4  4  4  4  4
0  0  0  0  0  0

2  2  2  3  3  3
2  2  2  3  3  3
2  2  2  3  3  3
2  2  2  3  3  3
# End: data text
```

Figure 20.1: OIF 1.0 sample file, with comments. (description)

Chapter 21

Troubleshooting

The OOMMF developers rely on reports from OOMMF users to alert them to problems with the software and its documentation, and to guide the selection and implementation of new features. See the Credits for instructions on how to contact the OOMMF developers.

The more complete your report, the fewer followup messages will be required to determine the cause of your problem. Usually when a problem arises there is an error message produced by the OOMMF software. A stack trace may be offered that reveals more detail about the error. When reporting an error, it will help the developers diagnose the problem if users cut and paste into their problem report the error message and stack trace exactly as reported by OOMMF software. In addition, **PLEASE** include a copy of the output generated by `tclsh oommf.tcl +platform`; this is important because it will help OOMMF developers identify problems that are installation or platform dependent.

Before making a report to the OOMMF developers, please check the following list of fixes for known problems. Additional problems discovered after release will be posted to version specific "patch" pages at the OOMMF web site.

1. When compiling, there is an error about being unable to open system header files like `stdlib.h`, `time.h`, `math.h`, or system libraries and related program startup code. This usually indicates a bad compiler installation. If you running on Windows and building with the Microsoft Visual C++ command line compiler, did you remember to run `vcvars32.bat` to set up the necessary environment variables? If you are using the Borland C++ compiler, are the `bcc32.cfg` and `ilink32.cfg` files properly configured? In all cases, check carefully any notes in the chapter Advanced Installation pertaining to your compiler.

2. When compiling, there is an error something like:

   ```
   <30654> pimake 1.x.x.x MakeRule panic:
   Don't know how to make '/usr/include/tcl.h'
   ```

   This means the header file `tcl.h` is missing from your Tcl installation. Other missing header files might be `tk.h` from the Tk installation, or `Xlib.h` from an X Window

System installation on Unix. In order to compile OOMMF, you need to have the development versions of Tcl, Tk, and (if needed) X installed. The way to achieve that is platform-dependent. On Windows you do not need an X installation, but when you install Tcl/Tk be sure to request a "full" installation, or one with "header and library files". On Linux, be sure to install developer packages as well as user packages. Other platforms are unlikely to have this problem. In the case of `Xlib.h`, it is also possible that the `tkConfig.sh` file has an incorrect entry for `TK_XINCLUDES`. A workaround for this is to add the following line to your `oommf/config/platforms/`*platform* file:

```
$config SetValue TK_XINCLUDES "-I/usr/X11R6/include"
```

Adjust the include directory as appropriate for your system.

3. When compiling, there is an error indicating that exceptions are not supported.

   Parts of OOMMF are written in C++, and exceptions have been part of the C++ language for many years. If your compiler does not support them, it is time to upgrade to one that does. OOMMF 1.2 requires a compiler capable of compiling source code which uses C++ exceptions.

4. Compiling with gcc produces syntax errors on lines involving `auto_ptr` templates.

   This is known to occur on RedHat 5.2 systems. The `auto_ptr` definition in the system STL header file `memory` (located on RedHat 5.2 systems in the directory `/usr/include/g++`) is disabled by two `#if` statements. One solution is to edit this file to turn off the `#if` checks. If you do this, you will also have to fix two small typos in the definition of the `release()` member function.

5. When compiling there is an error message arising from system include directories being too early in the include search path. Try adding the offending directories to the `program_compiler_c++_system_include_path` property in the platform file, e.g.,

   ```
   $config SetValue program_compiler_c++_system_include_path \
         [list /usr/include /usr/local/include]
   ```

6. On Solaris, gcc reports many errors like

   ```
   ANSI C++ forbids declaration 'XSetTransientForHint' with no type
   ```

   On many Solaris systems, the header files for the X Window System are not ANSI compliant, and gcc complains about that. To work around this problem, edit the file `oommf/config/platforms/solaris.tcl` to add the option `-fpermissive` to the gcc command line.

7. On Windows, when first starting `oommf.tcl`, there is an error:

   ```
   Error launching mmLaunch version 1.x.x.x:
           couldn't execute "...\omfsh.exe": invalid argument
   ```

This cryptic message most likely means that the pre-compiled OOMMF binaries which were downloaded are for a different version of Tcl/Tk than is installed on your system. Download OOMMF again, taking care this time to retrieve the binaries that match the release of Tcl/Tk you have installed.

8. When first starting `oommf.tcl`, there is an error:

   ```
   Error in startup script:  Neither Omf_export nor
   Omf_export_list set in
   ```

   The file `oommf/pkg/net/omfExport.tcl` may be missing from your OOMMF installation. If necessary, download and install OOMMF again.

9. When launching `oommf.tcl` on Unix systems, there is an error of the form:

   ```
   error while loading shared library:  libtk8.4.so:  cannot open
   shared object file:  No such file or directory
   ```

   This typically happens because the libtk#.#.so (and/or libtcl#.#.so) files are installed in a directory not included in the ld.so runtime linker/loader search path. One way to fix this is to add that directory (say /usr/local/lib) to the `LD_LIBRARY_PATH` environment variable. For example, include

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
   ```

   in your `~/.bashrc` file (bash shell users) or

   ```
   setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/usr/local/lib
   ```

   in your `~/.cshrc` file (csh or tcsh shell users). Another option is to modify the ld.so cache; see the ld.so and ldconfig man pages for details.

10. When starting OOMMF in the Cygwin environment on Windows, the mmLaunch window appears briefly, then disappears without any error messages.

    Some old versions of Tcl/Tk included with the Cygwin environment (i.e., `/usr/bin/tclsh`) had bugs in the socket code that caused OOMMF to crash in this manner. This problem is fixed in current Cygwin releases.

11. I ran out of memory!

    Are you using **mmGraph** to monitor a long-running simulation? All data sent to **mmGraph** is kept in memory by default. See the **mmGraph** documentation for information on how to manage this problem.

12. When running many OOMMF applications, some of them become non-responsive.

    Operating systems place an upper limit on the number of files a process may have open at one time. The number varies, but the default count is usually a few hundred to

perhaps a few thousand. OOMMF applications communicate with one another across localhost sockets. Each socket connection counts as an open file, so if you are running hundreds of OOMMF applications under one account you may hit this limit. You can explore increasing the open file limit (e.g., on Linux and macOS see the `-n` option to `ulimit`), but a more robust solution is to divide your collection of OOMMF jobs into several smaller independent groups and run each group under separate host + account servers with the **launchhost** utility.

13. When trying to simulate a hysteresis loop, the magnetization doesn't reverse even at very large fields.

    There may be a symmetry breaking problem. If the problem geometry and magnetization is symmetric about for example the x-axis, and the applied field is exactly on the x-axis, then magnetization switching can be frustrated by net zero torque. For example if $\mathbf{m} = (1, 0, 0)$ and $\mathbf{H} = (-1, 0, 0)$, then $\mathbf{m} \times \mathbf{H} = 0$. This is analogous to a ball stuck at the exact top of a hill with maximum potential energy, but not knowing which way to roll downhill. The fix is to introduce some asymmetry, either in the problem geometry or by directing the applied field slightly off the coordinate axis.

14. My simulation runs too slow!

    This is largely just the nature of the beast. Micromagnetics operates at the nanometer and nanosecond time scale. Simulation parts larger than a few micrometers or simulation times longer than a few tens of nanoseconds will require a long run time, or may be completely infeasible. However, there are a few things you can check:

    - The cell discretization size should generally be smaller than both the magnetostatic- and magnetocrystalline-exchange lengths ($\sqrt{2A/(\mu_0 M_s)}$ and $\sqrt{A/|K_1|}$, respectively). If the cell size is larger than this then the magnetization sampling may be too coarse to adequately represent the magnetization variation, and the results will be suspect. On the other hand, cell sizes much smaller than the exchange length also have adverse effects. Obviously, smaller cells mean more total cells in the simulation, and the compute time per iteration grows with the total cell count. (The growth is generally somewhat worse than linear.) But additionally, the exchange coupling between neighboring cells grows inversely with cell size, so small cells mean strong exchange coupling. This restricts the allowable per-iteration magnetization configuration change, meaning reduced time steps for LLG integration and smaller energy minimization steps for conjugate-gradient solves. Thus, cell sizes smaller than necessary increase both the compute time per iteration and the number of iterations.
    - Simulations with a large number of cells also require a large amount of memory. The per-cell memory usage depends on the simulation details, but it is typically between 512 and 1024 bytes per cell. Nearly all of this memory will be accessed

on each iteration, so compute time will explode if the simulation memory requirements outstrip available physical memory and your system resorts to swapping memory to disk. You can check the total cell count in the **Oxsii** `Help|About` dialog box.

- Are torque-based stopping criteria (`stopping_mxHxm`, `stopping_dm_dt`) appropriate? If too tight compute time can be wasted with little to no gain in accuracy.
- The **Oxsii** and **Boxsi** 3D solvers are designed to run multiple compute threads in parallel. (Conversely, the older **mmSolve2D** solver is sequential only, which by itself is sufficient reason to migrate to **Oxsii**/**Boxsi**.) The compute thread count is controlled by the `Threads` entry in the **Oxsii** `Load Problem` dialog box, or the `-threads` command line option to **Boxsi**. Increasing the thread count will generally improve simulation speed, up to the number of hardware threads on the computer. Beyond that context switching overhead between threads will usually reduce performance.
- If you are running multiple simulations in parallel on a multi-cpu Linux box, it may be worthwhile to restrict each simulation to a separate NUMA memory node (`NUMA` option in the **Oxsii** `Load Problem` dialog box, or `-numanodes` command line option to **Boxsi**). This requires OOMMF be built with the `use_numa option`.
- Magnetization evolution with LLG dynamics is largely dominated by precession about the effective field, especially if the damping is small. This is unavoidable in dynamic simulations studying, for example, the details of magnetization reversal. But in quasi-static simulations LLG integration can spend a lot of compute cycles precessing about local energy minima, shedding energy slowly as the magnetic configuration gradually descends into the energy minimum. In those cases it can be an order of magnitude quicker to do direct energy minimization with, for example, conjugate-gradient minimization.
- Think carefully about the specific information you want to glean from the micromagnetic simulation. Can the problem be reduced to something simpler, either a smaller geometry or a smaller time window? Perhaps the simulation can be decomposed into several more manageable parts? Can boundary conditions be introduced to decrease the active simulation size?

Chapter 22

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. Quick Start is based on notes from Dianne P. O'Leary.

The OOMMF[1] code is being developed mainly by Michael Donahue and Donald Porter. Robert D. McMichael made contributions to the early development of the 2D micromagnetic solver. Jason Eicke wrote the first version of the problem editor, and worked on the self-magnetostatic module of the 2D micromagnetic solver.

Numerous users have contributed to the development of OOMMF by submitting bug reports, small pieces of code, or suggestions for improvements. Many thanks to all these people, including Atif Aziz, Loris Bennett, Richard Boardman, Greg Brown, Dieter Buntinx, Ngoc-Nga Dao, Hans Fangohr, Colm Faulkner, Olivier Gérardin, Ping He, Michael Ho, Mingyu Hu, Mansoor B. A. Jalil, Jörg Jorzick, Pierre-Olivier Jubert, Pavel Kabos, Michael Kleiber, Kristof Lebecki, Oliver Lemcke, H. T. Leung, David Lewis, Sang Ho Lim, Yi Liu, Van Luu, Andy P. Manners, Damien McGrouther, Johan Moulin, Wong Lai Mun, Edward Myers, Andrew Newell, Valentine Novosad, Andrew Perrella, Angeline Phoa, Anil Prabhakar, Robert Ravlic, Stephen E. Russek, Renat Sabirianov, Zhupei Shi, Xiaobo Tan, Alexei Temiryazev, Alexander Thieme, Stephen Thompson, Vassilios Tsiantos, Pieter Visscher, Ruifang Wang, Scott L. Whittenburg, Kong Xiangyang, Ming Yan, Tan Swee Yong, Chengtao Yu, Steven A. Zielke, and Pei Zou.

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>`.

Acknowledgement is appreciated if the software is used. We recommend citing the following NIST technical report [7]:

> M. J. Donahue and D. G. Porter
> OOMMF User's Guide, Version 1.0
> Interagency Report NISTIR 6376
> National Institute of Standards and Technology, Gaithersburg, MD (Sept 1999).

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

and optionally include the URL of the OOMMF home page, https://math.nist.gov/oommf/. To help us keep our bibliography page[2] current, please direct publication information to <michael.donahue@nist.gov>.

[2]https://math.nist.gov/oommf/bibliography.html

Bibliography

[1] A. Aharoni, *Introduction to the Theory of Ferromagnetism* (Oxford, New York, 1996).

[2] A. Aharoni, "Demagnetizing factors for rectangular ferromagnetic prisms," *J. App. Phys.*, **83**, 3432–3434 (1998). doi: 10.1063/1.367113

[3] D. V. Berkov, K. Ramstöck, and A. Hubert, "Solving micromagnetic problems: Towards an optimal numerical method," *Phys. Stat. Sol. (a)*, **137**, 207–222 (1993). doi: 10.1002/PSSA.2211370118

[4] W. F. Brown, Jr., "Theory of the approach to magnetic saturation," *Physical Review*, **58**, 736–743 (1940). doi: 10.1103/PhysRev.58.736

[5] W. F. Brown, Jr., *Micromagnetics* (J. Wiley, New York, 1963).

[6] M. J. Donahue and R. D. McMichael, "Exchange energy representations in computational micromagnetics," *Physica B*, **233**, 272–278 (1997). doi: 10.1016/S0921-4526(97)00310-4

[7] 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). doi: 10.6028/NIST.IR.6376

[8] J. R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," *J. Comp. Appl. Math.*, **6**, 19–26 (1980). doi: 10.1016/0771-050X(80)90013-3

[9] J. R. Dormand and P. J. Prince, "A reconsideration of some embedded Runge-Kutta formulae," *J. Comp. Appl. Math.*, **15**, 203–211 (1986). doi: 10.1016/0377-0427(86)90027-0

[10] J. Fidler and T. Schrefl, "Micromagnetic modelling — the current state of the art," *J. Phys. D: Appl. Phys.*, **33**, R135–R156 (2000). doi: 10.1088/0022-3727/33/15/201

[11] H. Fukushima, Y. Nakatani, and N. Hayashi, "Volume average demagnetizing tensor of rectangular prisms," *IEEE Trans. Magn.*, **34**, 193–198 (1998). doi: 10.1109/20.650225

[12] L. Garcia-Palacios and F. Lazaro, "Langevin-dynamics study of the dynamical properties of small magnetic particles," *Physical Review B*, **58**, 14937–14958 (1998). doi: 10.1103/PhysRevB.58.14937

[13] T. L. Gilbert, "A Lagrangian formulation of the gyromagnetic equation of the magnetization field," *Phys. Rev.*, **100**, 1243 (1955).

[14] P. R. Gillette and K. Oshima, "Magnetization reversal by rotation," *J. Appl. Phys.*, **29**, 529–531 (1958). doi: 10.1063/1.1723211

[15] A. Hubert, "muMAG standard problem no. 3," (1998), [Online; accessed 17-Oct-2023]. URL https://www.ctcms.nist.gov/~rdm/results3.html

[16] T. Koehler, H. N. Bertram, A. Liu, C. Seberino, and R. D. McMichael, "muMAG standard problem no. 2," (1998), [Online; accessed 17-Oct-2023]. URL https://www.ctcms.nist.gov/~rdm/std2/results.html

[17] A. E. LaBonte, "Two-dimensional Bloch-type domain walls in ferromagnetic films," *Journal of Applied Physics*, **40**, 2450–2458 (1969). doi: 10.1063/1.1658014

[18] L. Landau and E. Lifshitz, "On the theory of the dispersion of magnetic permeability in ferromagnetic bodies," *Physik. Z. Sowjetunion*, **8**, 153–169 (1935). doi: 10.1016/B978-0-08-036364-6.50008-9

[19] J. W. Lau, R. D. McMichael, and M. J. Donahue, "Implementation of two-dimensional polycrystalline grains in object oriented micromagnetic framework," *Journal of Research of the National Institute of Standards and Technology*, **114**, 57–67 (2009). doi: 10.6028/jres.114.005

[20] K. M. Lebecki, M. J. Donahue, and M. W. Gutowski, "Periodic boundary conditions for demagnetization interactions in micromagnetic simulations," *J. Phys. D: Appl. Phys.*, **41**, 175005 (2008). doi: 10.1088/0022-3727/41/17/175005

[21] R. D. McMichael and M. J. Donahue, "Head to head domain wall structures in thin magnetic strips," *IEEE Trans. Magn.*, **33**, 4167–4169 (1997). doi: 10.1109/20.619698

[22] R. D. McMichael, R. Koch, T. Schrefl, and J. Eicke, "muMAG standard problem no. 4," (2000), [Online; accessed 17-Oct-2023]. URL https://www.ctcms.nist.gov/~rdm/std4/results.html

[23] M. Najafi and D. G. Porter, "muMAG standard problem no. 5," (2014), [Online; accessed 17-Oct-2023]. URL https://www.ctcms.nist.gov/~rdm/std5/results.html

[24] L. Néel, "Some theoretical aspects of rock magnetism," *Adv. Phys.*, **4**, 191–242 (1955). doi: 10.1080/00018735500101204

[25] A. J. Newell, W. Williams, and D. J. Dunlop, "A generalization of the demagnetizing tensor for nonuniform magnetization," *J. Geophysical Research - Solid Earth*, **98**, 9551–9555 (1993). doi: 10.1029/93JB00694

[26] D. G. Porter and M. J. Donahue, "Generalization of a two-dimensional micromagnetic model to non-uniform thickness," *Journal of Applied Physics*, **89**, 7257–7259 (2001). doi: 10.1063/1.1363606

[27] M. R. Scheinfein, J. Unguris, J. L. Blue, K. J. Coakley, D. T. Pierce, and R. J. Celotta, "Micromagnetics of domain walls at surfaces," *Phys. Rev. B*, **43**, 3395–3422 (1991). doi: 10.1103/PhysRevB.43.3395

[28] J. C. Slonczewski, "Current-driven excitation of magnetic multilayers," *Journal of Magnetism and Magnetic Materials*, **159**, L1–L2 (1996). doi: 10.1016/0304-8853(96)00062-5

[29] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis* (Springer, New York, 1993), 2nd edn.

[30] E. C. Stoner and E. P. Wohlfarth, "A mechanism of magnetic hysteresis in heterogeneous alloys," *Phil. Trans. Royal Soc. London*, **A240**, 599–642 (1948). doi: 10.1098/rsta.1948.0007

[31] A. Thiaville, Y. Nakatani, J. Miltat, and Y. Suzuki, "Micromagnetic understanding of current-driven domain wall motion in patterned nanowires," *Europhys. Lett.*, **69**, 990–996 (2005). doi: 10.1209/epl/i2004-10452-6

[32] A. Thiaville, S. Rohart, É. Jué, V. Cros, and A. Fert, "Dynamics of Dzyaloshinskii domain walls in ultrathin magnetic films," *Europhys. Lett.*, **100**, 57002 (2012). doi: 10.1209/0295-5075/100/57002

[33] P.-E. Weiss, "La variation du ferro-magnétisme avec la température," *Comptes Rendus*, **143**, 1136–1149 (1906).

[34] B. B. Welch, *Practical Programming in Tcl and Tk* (Prentice Hall, Upper Saddle River, New Jersey USA, 2000), 3rd edn.

[35] J. Xiao, A. Zangwill, and M. D. Stiles, "Boltzmann test of Slonczewski's theory of spin-transfer torque," *Phys. Rev. B*, **70**, 172405 (pages 4) (2004). doi: 10.1103/PhysRevB.70.172405

[36] S. Zhang and Z. Li, "Roles of nonequilibrium conduction electrons on the magnetization dynamics of ferromagnets," *PhysṘev. Lett.*, **93**, 127204 (2004). doi: 10.1103/PhysRevLett.93.127204

Index

account service directory, 27, 130, 131
  expires, 28
  launching of, 28
animations, 181
announcements, 2
antialias, 183
application
  any2ppm, 35, 123, 131, 171, 222
  avf2odt, 172
  avf2ovf, 109, 177, 256
  avf2ppm, 154, 162, 181, 184
  avf2ps, 154, 184
  avf2svg, 188
  avfdiff, 188
  batchmaster, 133
  batchslave, 131
  batchsolve, 130, 133, 218
  bootstrap, 29–31
  Boxsi, 26, 228
  boxsi, 42, 211
  computefield, 191
  crc32, 194
  FileSource, 121, 219
  ghostscript, 186
  gzip, 152
  killoommf, 195
  lastjob, 42, 47, 196, 199
  launchhost, 28, 49, 197
  logreview, 42, 47, 196, 199
  mag2hfield, 201
  make, 214
  mifconvert, 24, 202, 218, 228
  mifpy2tcl, 203
  mmArchive, 21, 23, 25, 40, 93, 143, 165, 255
  mmDataTable, 21, 23, 25, 40, 93, 142, 145, 146
  mmDisp, 21, 23, 25, 40, 126, 132, 149, 151, 165, 181, 184, 255
  mmGraph, 21, 23, 25, 40, 93, 130, 143, 145, 157, 165, 289
  mmHelp, 168
  mmLaunch, 21, 33, 36, 39, 42, 46, 124, 129, 130
  mmProbEd, 21, 22, 24, 118, 121, 131, 218, 292
  mmSolve, 220
  mmSolve2D, 21, 22, 124, 130, 131, 218, 219
  odtcalc, 205
  odtcat, 206
  odtcols, 207
  OOMMF Batch System, 130
  Oxs, 35, 165, 196, 199
  Oxsii, 21, 36, 228
  oxsii, 24
  oxspkg, 209
  pidinfo, 28, 204, 213
  pimake, 10, 214
  ppmquant, 182
  ppmtogif, 182
  regression, 211
  ssh, 134, 137, 138
  suspend, 217
  tclsh, 3
  Windows Explorer, 32
  wish, 3
  Xvfb, 3–4, 31, 172
architecture, 27

batch processing 2D, *see* application, OOMMF Batch System
batch processing, *see* application, Boxsi and application, launchhost
bitmap files, *see* file, bitmap
Borland C++, *see* platform, Windows, Borland C++
boundary, 156, 161
bug reports, *see* reporting bugs

cell size, 221
citation information, 292
client, 27
client-server architecture, 27
color
  discretization, 160
  map, 154
  quantity, 154, 160
colorbar, making, 184
colorwheel, making, 184
communication protocol, 135
compilers, 4
compressed files, 146, 152
configuration values, 7
  oommf_thread_count, 8
  oommf_thread_limit, 8
  oommf_threads, 8
  path_directory_temporary, 8
  thread_count, 8
  thread_limit, 8
contact information, 292
contributors, 292
control points, *see* simulation 2D, control point
CRC-32, 194
crystalline anisotropy, 219
curve break, 146
customize, 11
  file format translation, 152–153
  help file browser, 168
  host server port, 28
cut-and-paste, 144
Cygwin, *see* platform, Windows, Cygwin environment

data
  print, 147, 153, 158, 162
  save, 23, 25, 147, 150, 153, 158, 165, 166
  scale, 155, 156
  slice selection, 156, 158
  zoom, 156
decompress, *see* compressed files
DefaultFieldFilename command (MIF), 258
demagnetization, 220
Destination command (MIF), 165, 231
download, 4

e-mail, 2, 292
edge anisotropy, 220
energy
  anisotropy, 126, 128
  crystalline anisotropy, 219
  demag, 127, 128, 292
  edge anisotropy, 220
  exchange, 126, 128
  total, 126, 128, 132
  Zeeman, 127, 128
environment variables
  customization, 12
  LD_LIBRARY_PATH, 9
  OOMMF_NUMANODES, 15, 38, 44
  OOMMF_OUTDIR, 38, 39, 44, 45
  OOMMF_RESTARTFILEDIR, 39, 45
  OOMMF_TCL_CONFIG, 7
  OOMMF_TCLSH, 7
  OOMMF_TEMP, 8
  OOMMF_THREADLIMIT, 8
  OOMMF_THREADS, 8, 39, 45
  OOMMF_TK_CONFIG, 8
  OOMMF_WISH, 8
  PATH, 9
  TCL_LIBRARY, 9, 20
  TK_LIBRARY, 9
EvalScalarField command (MIF), 258
EvalVectorField command (MIF), 258
exchange stiffness, 219

FFT, 129, 221

field
applied, 126, 132, 224
effective, 128
update count, 126
field range, 223
file
bitmap, 35, 55–58, 123, 131, 171, 181, 222, 235–237
bmp, 171, 172, 181, 222
checkpoint, 93
configuration, 146, 148, 153, 182, 186
conversion, 171, 172, 177, 181, 184, 188, 191, 201–203
data table, 23, 25, 125, 132, 136, 137, 146, 166, 172, 205–207, 225, 271
demag field, 191
difference, 188
gif, 171, 182, 222
hosts, *see* platform, Windows, hosts file
HTML, 168
log, 37, 42, 43, 47, 127, 131, 132, 196, 199
magnetization, 131, 137, 191, 201, 225
mask, 35, 55–58, 123, 131, 222, 246
MIF, 202, 203, 218
MIF 1.1, 118
MIF 1.2, 118
MIF 1.x, 121, 127, 131, 132, 135–138
obf, *see* file, vector field
odt, *see* file, data table
ohf, *see* file, vector field
omf, *see* file, magnetization
ovf, *see* file, vector field
pdf, 186
PostScript, 184
ppm, 171, 181, 182, 222
PyMIF, 203
restart, *see* file, checkpoint
scalar field, 191
svf, 273, 274, 278
SVG, 188
VecFil, 273
vector field, 23, 25, 125, 132, 152, 153, 156, 166, 172, 177, 181, 184, 188, 191, 201, 223, 225, 273
vio, 172, 177, 188, 223, 273

GetMifFilename command (MIF), 255
GetMifParameters command (MIF), 255
grid, 128, 163, 178, 277
gyromagnetic ratio, 220

host service directory, 27–28
expires, 28
launching of, 28

installation, 3
Python, 8
Tcl/Tk, 7–9
instance name (MIF 2), 231
Internet, *see* TCP/IP
iteration, 126, 132

Landau-Lifshitz, *see* ODE, Landau-Lifshitz
launch
command line arguments, 29–31
foreground, 29
from command line, 29
standard options, 30–31
version requirement, 29–30
with bootstrap application, 29
with mmLaunch, 34
license, v
log file, *see* file, log

magnetization, 126, 132
magnetization initial, 222
margin, 161
mask file, *see* file, mask
materials, 219
max angle, 127
memory use, 148
mesh, *see* grid
MIF, *see* file, mif
MIF 2.1 Commands, 230
MIF 2.1 Overview, 228
MIF 2.2 Commands, 254

mmLaunch user interface, 33–34, 39, 46, 125, 129, 165, 166
movies, *see* animations
mxh, *see* simulation 2D, mxh

NetPBM, 182
network socket, 2, 135
nicknames, 30, 204, 214, 231
NUMA, 8, 14–15, 37–38, 43–44, 48, *see also* parallelization

OBS, *see* application, OOMMF Batch System
ODE
  Landau-Lifshitz, 79, 82, 85, 128, 129, 220
  predictor-corrector, 129
  Runge-Kutta, 129
  step size, 226
OID's, 204, 213
opaque pixel, 161
optimization, 13
options.tcl, 11
output schedule, 22, 23, 25, 125
Oxs, 35
Oxs_Ext child classes, 49
  contributed, 117
  Oxs_AffineOrientScalarField, 103
  Oxs_AffineOrientVectorField, 112
  Oxs_AffineTransformScalarField, 105
  Oxs_AffineTransformVectorField, 112
  Oxs_AtlasScalarField, 98
  Oxs_AtlasVectorField, 107
  Oxs_BoxAtlas, 51
  Oxs_CGEvolve, 87
  Oxs_CubicAnisotropy, 63
  Oxs_Demag, 70
  Oxs_EllipseAtlas, 52
  Oxs_EllipsoidAtlas, 53
  Oxs_EulerEvolve, 79
  Oxs_Exchange6Ngbr, 64
  Oxs_ExchangePtwise, 65
  Oxs_FileAtlas, 53
  Oxs_FileVectorField, 109
  Oxs_FixedZeeman, 72
  Oxs_ImageAtlas, 55
  Oxs_ImageScalarField, 106
  Oxs_ImageVectorField, 115
  Oxs_LabelValue, 116
  Oxs_LinearScalarField, 98
  Oxs_MaskVectorField, 114
  Oxs_MinDriver, 95
  Oxs_MultiAtlas, 58
  Oxs_ParallelepipedAtlas, 51
  Oxs_PeriodicRectangularMesh, 61
  Oxs_PlaneRandomVectorField, 111
  Oxs_RandomScalarField, 99
  Oxs_RandomSiteExchange, 66
  Oxs_RandomVectorField, 110
  Oxs_RectangularMesh, 61
  Oxs_RungeKuttaEvolve, 82
  Oxs_ScriptAtlas, 59
  Oxs_ScriptOrientScalarField, 102
  Oxs_ScriptOrientVectorField, 111
  Oxs_ScriptScalarField, 99
  Oxs_ScriptUZeeman, 72
  Oxs_ScriptVectorField, 108
  Oxs_SimpleDemag, 71
  Oxs_SpinXferEvolve, 85
  Oxs_StageZeeman, 76
  Oxs_TimeDriver, 91
  Oxs_TransformZeeman, 74
  Oxs_TwoSurfaceExchange, 67
  Oxs_UniaxialAnisotropy, 62
  Oxs_UniformExchange, 65
  Oxs_UniformScalarField, 97
  Oxs_UniformVectorField, 106
  Oxs_UZeeman, 71
  Oxs_VecMagScalarField, 101

parallelization, 8, 14–15, 37, 39, 43, 45, 48, *see also* NUMA
part geometry, 221
PID's, 204, 213
platform, 287
  configuration, 5
  macOS

configuration, 19
names, 6–7, 15–17
Unix
configuration, 17–18
executable Tcl scripts, 31
PostScript to printer, 147, 153
X server, 165
Windows
configuration, 9, 19–20
Cygwin environment, 7, 19
dummy user ID, 195, 204, 213, 217
file extension associations, 31–32
file path separator, 9
hosts file, 9
Microsoft Visual C++, 19
MinGW g++, 19
no Tcl configuration file, 7
setting environment variables, 20
wildcard expansion, 177, 182
platform.tcl, 11
platforms, 4
ports, 197
precession, 220
processes
host server, 197
killing, 195
suspend, 217

random numbers, 226
record identifier, 219
reporting bugs, 287, 292
requirement
application version, *see* launch, version requirement
C++ compiler, 4
disk space, 11
display, 35, 123, 131
Python, 8
ssh, 134
Tcl/Tk, 3
TCP/IP, 3
Tk, 35, 123, 131, 172
Tk 8.0+, 171
running applications, 22, 24, 34, 39, 46, 196

sampling, 160
saturation magnetization, 219
Schedule command (MIF), 237
segment block, 275
self-magnetostatic, *see* demagnetization
server, 27
services, 27
SetOptions command (MIF), 94, 96, 255
simulation 2D
restarting, 124, 131
simulation 3D
restarting, 39, 45
simulation 2D, 123, 124, 130, 292
control point, 23, 125, 129, 135, 137, 223, 224
equilibrium, 23
interactive control, 23, 127, 131
iteration, 223
mxh, 126, 223, 225
scheduling, 133, 138
termination, 128, 131
time, 126, 223
simulation 3D
batch, 42
interactive, 36
interactive control, 25
restarting, 93
stage, 25
Specify attributes (MIF), 240
Specify block (MIF), 230
Specify comments (MIF), 240
Specify conventions (MIF), 239
Specify grouped lists (MIF), 241
Specify initialization string (MIF), 239
Specify Oxs_Ext referencing (MIF), 242
Specify user defined scalar outputs (MIF), 246
Specify user defined support procs (MIF), 244
standard options, 30–31
step size, 126

task script, 133, 135
Tcl list, 219, 239
TCP/IP, 3, 27
temporary files, 8
threads, 125
threads, parallel, *see* parallelization
time step, 126
torque, *see* simulation 2D, mxh
total field, *see* field, effective

user ID, 27

variable substitution (MIF), 249
vortex, 223

working directory, 5, 10, 29, 131, 138

Xvfb, *see* application, Xvfb