Molecular Zeeman Library

Reference Manual
Edition 1.0, for MZL Version 1.0
1 December 2004

Bernard Leroy
Centre national de la recherche scientifique & Observatoire de Paris
Laboratoire d'études spatiales et d'instrumentation en astrophysique (LESIA)

The Texinfo source for this manual is included in the package's archive `http://bass2000.obspm.fr/mzl/download/mzl-1.0.tar.gz`.

1 Introduction

The Molecular Zeeman Library (MZL) is a collection of routines for numerically computing the Zeeman effect in diatomic molecules. The routines have been written in C, and present a modern Applications Programming Interface (API) for C programmers, allowing wrappers to be written for very high level languages. A C-Fortran interface is also provided for the benefit of Fortran programmers. The source code is distributed under the GNU General Public License.

1.1 The object of MZL

The Molecular Zeeman Library provides routines to compute the Zeeman effect in states of diatomic molecules described by Hund's cases (a), (b), or intermediate between (a) and (b). The computations are restricted to cases of astrophysical interest. This implies that some approximations have been assumed; they are described below (see Chapter 2 [Scientific context], page 4).

Routines for computing not only the splitting of the energy terms but also the splitting of individual band lines are provided. In the latter case, as is usual in astrophysics, the whole Zeeman pattern lies within the profile of the broadened unperturbed line. It is therefore convenient to compute the weighted mean displacements of the patterns the σ-components by weighing with the relative intensities of the individual transitions. Routines for the latter as well as for the weighted mean displacements are available.

Note that the Molecular Zeeman Library is not self-contained: the molecular constants have to be provided by the user (see Section 6.1 [C Data Types], page 17 or Section 7.1 [Data Types], page 23).

1.2 MZL is Free Software

The subroutines in the Molecular Zeeman Library are "free software"; this means that everyone is free to use them, and to redistribute them in other free programs. The library is not in the public domain; it is copyrighted and there are conditions on its distribution. These conditions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of the software that they might get from you.

Specifically, we want to make sure that you have the right to share copies of programs that you are given which use the Molecular Zeeman Library, that you receive their source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of any code which uses the Molecular Zeeman Library, you must give the recipients all the rights that you have received. You must make sure that they, too, receive or can get the source code, both to the library and the code which uses it. And you must tell them their rights. This means that the library should not be redistributed in proprietary programs.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the Molecular Zeeman Library. If these programs are modified by someone

else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions for the distribution of software related to the Molecular Zeeman Library are found in the GNU General Public License (see [GNU General Public License], page 35).

`http://www.gnu.org/copyleft/gpl-faq.html`

1.3 Obtaining MZL

The source code for the library can be obtained in different ways, by copying it from a friend, or downloading it from the internet:

`http://bass2000.obspm.fr/mzl/download/mzl-1.0.tar.gz`

The preferred platform for the library is a GNU system, which allows it to take advantage of additional features in the GNU C compiler and GNU C library. However, the library is fully portable and should compile on most systems.

1.4 Referring to MZL

If you use the Molecular Zeeman Library in your research work, the author would appreciate your acknowledging it; in a publication, kindly use a phrase such as the following:

This research has made use of the Molecular Zeeman Library (Leroy, 2004).

and refer to the MZL by citing the reference manual, e.g.,

B. Leroy, 2004: *Molecular Zeeman Library Reference Manual* (avalaible on-line at http://bass2000.obspm.fr/mzl/download/mzl-ref.pdf)

1.5 No Warranty

The software described in this manual has no warranty, it is provided "as is". It is your responsibility to validate the behavior of the routines and their accuracy using the source code provided. Consult the GNU General Public license for further details (see [GNU General Public License], page 35).

1.6 Reporting Bugs

A list of known bugs can be found in the '`BUGS`' file included in the MZL distribution. Details of compilation problems can be found in the '`INSTALL`' file.

If you find a bug which is not listed in these files please report it to `Bernard.Leroy@obspm.fr`.

All bug reports should include:

- The version number of MZL
- The hardware and operating system
- The (C and/or Fortran) compilers used, including version number and compilation options

- A description of the bug behavior
- A short program which exercises the bug

It is also useful if you can report whether the same problem occurs when the library is compiled without optimization. Thank you.

1.7 Further Information

The developer of the library can be reached via his email address, Bernard.Leroy@obspm.fr or via his postal address:

Observatoire de Paris
LESIA - Bat. 14
92195 Meudon Cedex
France

2 Scientific context

This chapter describes the scientific background used in building the MZL.

2.1 The Unperturbed State

The description of the molecular state is restricted to Hund's cases (a) and (b), and to intermediate couplings between (a) and (b). The computations are done with Hund's case (a) wave functions, $|\Lambda S\Sigma;\Omega JM>$, as a basis set ($\Omega = \Lambda+\Sigma$). In this basis, the Hamiltonian, $\mathcal{H}$, of the molecule in the absence of a magnetic field has the following elements (see Hougen, Kovács and Herzberg in Section 2.3 [References], page 5):

$$\begin{aligned} <\Lambda\Sigma|\mathcal{H}|\Lambda\Sigma> &= A_v\Lambda\Sigma + B_v[J(J+1) - \Omega^2 + S(S+1) - \Sigma^2] \\ &\quad + \gamma[\Sigma^2 - S(S+1)] + \lambda[3\Sigma^2 - S(S+1)] \\ <\Lambda\Sigma|\mathcal{H}|\Lambda\Sigma \pm 1> &= (B_v - \gamma/2)\sqrt{S(S+1) - \Sigma(\Sigma \pm 1)}\sqrt{J(J+1) - \Omega(\Omega \pm 1)} \end{aligned}$$

where A_v and B_v ($= B_e - \alpha_e(v+1/2) + \cdots$) are the constant of spin-orbit interaction and the rotational constant for a given vibrational state v, γ is the constant of spin-rotation interaction, and λ the constant of spin-spin interaction (the corresponding term in the Hamiltonian element above is valid only for ${}^3\Sigma$ states[1]). All the constants are in cm^{-1}.

The above expressions imply that the Λ-doubling effect is not taken into account, which is reasonable in astrophysical applications.

The intermediate case wavefunctions are computed as linear combinations of case (a) wavefunctions as:

$$|\Psi_i^{ab}> = \sum_{\alpha=-S}^{S} C_{\alpha i}|\Psi_\alpha^a>$$

where the coefficients $C_{\alpha i}$ are the components of the eigenvectors of $\mathcal{H}$.

The ratio $Y = A_v/B_v$ is an indicator of how close a state is to the pure Hund's cases; if $Y \gg J(J+1)$, then the state is well described by a pure Hund's case (a), whereas if $Y \ll J(J+1)$, it is well described by a pure Hund's case (b).

Let us warn the user that there may be species and/or states whose description does not fit well in the model underlying the MZL. Should the user be confronted with such a nasty case, they should consult the specialized litterature on molecules.

[1] For other states for which this interaction should be taken into account, the Hamiltonian should be modified in an appropriate way. However, no possibility of dynamically changing the Hamiltonian has been implemented.

2.2 Zeeman effect

The largest contribution to the Zeeman effect is due to the magnetic moment associated with the electronic spin and orbital angular momentum. No contribution of nuclear origin is taken into account. The perturbation matrix, $\mathcal{H}'$, has therefore the following elements (see Kovács and Schadee in Section 2.3 [References], page 5):

$$< \Lambda\Sigma|\mathcal{H}'|\Lambda\Sigma > = \frac{(\Lambda + 2\Sigma)\Omega}{J(J+1)} M\Delta\sigma_0$$
$$< \Lambda\Sigma|\mathcal{H}'|\Lambda\Sigma \pm 1 > = \frac{\sqrt{S(S+1) - \Sigma(\Sigma \pm 1)}\sqrt{J(J+1) - \Omega(\Omega \pm 1)}}{J(J+1)} M\Delta\sigma_0$$

where $\Delta\sigma_0 = \mu_B H\,\mathrm{cm}^{-1}$ (μ_B is Bohr's magneton and H the magnetic field strength in Gauss).

The Zeeman shift of energy level i is given by

$$\Delta E_i = \sum_\alpha \sum_\beta C_{\alpha i} C_{\beta i} \mathcal{H}'_{\alpha\beta} =: gM\Delta\sigma_0$$

where $\mathcal{H}'_{\alpha\beta}$ are the elements of the perturbation matrix. The last equality defines the Landé factor, g.

In astrophysical situations the magnetic field generally is not strong enough to split the unperturbed, usually broadened, line: the Zeeman patterns lie within the unperturbed line profile. As a result, a useful measure of the Zeeman effect is obtained by computing the weighted mean displacements, $< \Delta\sigma >$, of the patterns for each ΔM, the weighing factors being the relative intensities of the transitions, and by computing the separation of the weighted mean positions of the σ-patterns ($\Delta M = +1$ and -1): $< \Delta\sigma^+ > - < \Delta\sigma^- >$. The weighted means of the π-patterns ($\Delta M = 0$) are always zero. (See Schadee in Section 2.3 [References], page 5.)

2.3 References

The standard references are the following books:

G. Herzberg, *Molecular Spectra and Molecular Structure. I. Spectra of Diatomic Molecules*, D. van Nostrand, 1950.

J. T. Hougen, *The Calculation of Rotational Energy Levels and Rotational Line Intensities in Diatomic Molecules*, NBS Monograph, 115, 1970 (available on-line at `http://physics.nist.gov/Pubs/Mono115/contents.html`).

K. P. Huber and G. Herzberg, *Molecular Spectra and Molecular Structure. IV. Constants of Diatomic Molecules*, D. van Nostrand, 1979 (update of the reference by Herzberg above). The tables can be found on-line at `http://webbook.nist.gov/chemistry/`.

I. Kovács, *Rotational Structure in the Spectra of Diatomic Molecules*, D. van Nostrand, 1950.

As far as the molecular Zeeman in astrophysical applications is concerned, the reader is referred to the following articles.

A. Schadee, "On the Zeeman effect in electronic transitions of diatomic molecules", *J. Quant. Spectrosc. Radiat. Transfer*, 19, 517 (1978).

S. V. Berdyugina and S. K. Solanki, "The molecular Zeeman effect and diagnostics of solar and stellar magnetic fields", *Astron. Astrophysics*, 385, 701 (2002).

3 Using the library from C/C++

This chapter describes how to compile programs that use MZL, and introduces its conventions.

3.1 An Example Program

The following short program demonstrates the use of the library by computing the value of the Landé factor of the energy term corresponding to $\Omega = 3/2$ of a regular ${}^2\Pi_{3/2}$ doublet for an angular momentum value $J = 3/2$,

```
#include <stdio.h>
#include <stdlib.h>

#include <mzl/mzl.h>

int main(void)
{
  double Av = 79.01;
  double Bv = 4.29;
  int Lambda = 1;
  int Mult = 2;
  double J = 1.5;
  double Omega = 1.5;

  int term;
  double g;
  mzl_state *s;
  mzl_zeeman *z;

  mzl_init();

  s = mzl_state_create(Av, Bv, 0.0, 0.0, Mult, Lambda, "2Pi");
  if (!s)
    exit(EXIT_FAILURE);

  z = mzl_zeeman_alloc(s);
  if (!z)
    exit(EXIT_FAILURE);

  if (!mzl_zeeman_compute(z, J))
    exit(EXIT_FAILURE);

  term = mzl_state_term_number(s, Omega);
  g = mzl_lande_of_term(z, term);

  printf("Lande factor of term #%d: %.5f\n", term, g);

  mzl_zeeman_free(z);
```

```
  mzl_state_destroy(s);

  mzl_cleanup();

  return EXIT_SUCCESS;
}
```

The output is shown below,

```
Lande factor of term #2: 0.88674
```

The steps needed to compile this program are described in the following sections.

3.2 ANSI C Compliance

The library is written in ANSI C and is intended to conform to the ANSI C standard. It should be portable to any system with a working ANSI C compiler.

The library does not rely on any non-ANSI extensions in the interface it exports to the user. Programs you write using MZL can be ANSI compliant.

To avoid namespace conflicts all exported names have the prefix `mzl_`.

3.3 Compiling and Linking

There is only one library header file, '`mzl.h`', and it is installed in its own '`mzl`' directory. You should write the corresponding preprocessor include statement with a '`mzl/`' directory prefix thus,

```
#include <mzl/mzl.h>
```

If the directory is not installed on the standard search path of your compiler you will also need to provide its location to the preprocessor as a command line flag. The default location of the '`mzl`' directory is '`/usr/local/include/mzl`'. A typical compilation command for a source file '`example.c`' with the GNU C compiler `gcc` is,

```
gcc -I/usr/local/include -c example.c
```

This results in an object file '`example.o`'. The default include path for `gcc` searches '`/usr/local/include`' automatically so the `-I` option can be omitted when MZL is installed in its default location.

The library generally is installed as two files, '`libmzl.a`' and '`libmzlcfi.a`'[1] (possibly plus some '`*.so*`' files, if the shared version of the library is installed); the latter contains the C-Fortran interface to MZL proper, and is to be used when a Fortran program needs access to MZL (see Chapter 4 [Using the library from Fortran], page 11). A shared version of the library is also installed on systems that support shared libraries. The default location of these files is '`/usr/local/lib`'. To link C modules against the library you need to specify the main library only. The following example shows how to link an application with the library,

[1] If the library is installed without its C-Fortran interface, the '`libmzlcfi.a`' is not installed.

```
gcc -o example example.o -lmzl -lgsl -lgslcblas -lm
```

As MZL is built on the GNU Scientific Library, the program must also be linked to the GSL.

The program `mzl-config` provides information on the local version of the library and of the other necessary libraries. For example, the following command shows that the library has been installed under the directory '`/usr/local`',

```
bash$ mzl-config --prefix
/usr/local
```

The examples above could have been written,

```
gcc -c example.c `mzl-config --cflags`
gcc -o example example.o `mzl-config --libs`
```

or, since there is here only one source file,

```
gcc -o example example.c `mzl-config --cflags --libs`
```

Please, note the backticks in the examples above.

Further information on `mzl-config` is available using the command `mzl-config --help`.

3.4 Shared Libraries

To run a program linked with the shared version of the library it may be necessary to define the shell variable `LD_LIBRARY_PATH` to include the directory where the library is installed. For example, in the Bourne shell (`/bin/sh` or `/bin/bash`), if the library has been installed in a '`lib`' directory of the user's home directory, the library path can be set with the following commands:

```
LD_LIBRARY_PATH=$HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
./example
```

In the C-shell (`/bin/csh` or `/bin/tcsh`) the equivalent command is,

```
setenv LD_LIBRARY_PATH $HOME/lib:$LD_LIBRARY_PATH
```

To save retyping these commands each session they should be placed in an individual or system-wide login file.

To compile a statically linked version of the program, use the `-static` flag in `gcc`,

```
gcc -o example -static example.o `mzl-config --libs`
```

With another compiler, this option may exist under a different name, or not at all; in the latter case, if you insist on using statically linked executables, you should consider installing the static version of the library only (see option `--disable-shared` of `configure`).

3.5 Compatibility with C++

The library header file automatically define functions to have `extern "C"` linkage when included in C++ programs.

3.6 Thread-safety

The library can be used in multi-threaded programs in as much as the GNU Scientific Library, which MZL requires, is thread-safe (see the corresponding section in GSL's documentation) and that all the functions of MZL are thread-safe, in the sense that they do not use static variables. Memory is always associated with objects and not with functions.

4 Using the library from Fortran

This chapter describes how to compile Fortran programs that use MZL, and introduces its conventions.

4.1 An Example Program

The following short program demonstrates the use of the library by computing the value of the Landé factor of the energy term corresponding to $\Omega = 3/2$ of a regular $^2\Pi_{3/2}$ doublet for an angular momentum value $J = 3/2$,

```
PROGRAM intro_f90

  IMPLICIT NONE

  INCLUDE 'mzl/mzl.inc'

  CHARACTER (LEN=*), PARAMETER :: name = '2Pi'
  INTEGER, PARAMETER :: Mult = 2, Lambda = 1
  DOUBLE PRECISION, PARAMETER :: Av = 79.01, Bv = 4.29, gamma = 0.0, &
   lss = 0.0 , J = 1.5, Omega = 1.5

  INTEGER :: term
  INTEGER :: st, z
  DOUBLE PRECISION :: g

  st = cfi_state_create(Av, Bv, gamma, lss, Mult, Lambda, name)
  IF (st .LT. 0) THEN
   STOP
  END IF

  z = cfi_zeeman_alloc(st)
  IF (z .LT. 0) THEN
   STOP
  END IF

  IF (cfi_zeeman_compute(z, J) .EQ. 0) THEN
   STOP "Error computing Zeeman effect"
  END IF

  term = cfi_state_term_number(st, Omega)
  g = cfi_zeeman_lande_of_term(z, term)

  CALL cfi_zeeman_free(z)
  CALL cfi_state_destroy(st)

  WRITE(*, '(A22,I1,A1,F8.5)') 'Lande factor of term #', term, ':', g

END PROGRAM intro_f90
```

The output is shown below,

```
Lande factor of term #2: 0.88674
```

The steps needed to compile this program are described in the following sections.

4.2 Fortran Dialects

The C-Fortran interface is compatible with Fortran 77, Fortran 90/95 and upward versions of the language such as Fortran 2000 (possibly Fortran 2003). It has been tested on several architectures using several C and Fortran compilers[1].

The interface library proper, '`libmzlcfi.a`', is based on a piece of artwork, '`cfortran.h`', by B. D. Steinmacher-Burow (available at `http://www-zeus.desy.de/~burow/cfortran/`), which enables the Fortran programmer to write reasonably portable, mixed-language C-Fortran code; of course, there may (will) be limitations, but these are essentially due to the use of a Fortran compiler unknown to '`cfortran.h`' (sometimes it is just enough to cheat and tell '`cfortran.h`' that the Fortran compiler we want to use is one of the various it recognizes).

To avoid namespace conflicts all exported function names and variables have the prefix `cfi_`.

4.3 Compiling and Linking

A library file, '`mzl.inc`', which contains the declarations of all functions in the C-Fortran interface library, is installed in the '`mzl`' directory. You should write the corresponding include statement with a '`mzl/`' directory prefix thus,

```
INCLUDE 'mzl/mzl.inc'
```

and provide its location to the compiler as a command line flag. The default location of the '`mzl`' directory is '`/usr/local/include/`'. A typical compilation command for a source file '`example.f90`' with a Fortran 90 compiler `f90` is,

```
f90 -I/usr/local/include -c example.f90
```

This results in an object file '`example.o`'. Generally, the default include path for `f90` searches '`/usr/local/include`' automatically so the `-I` option can be omitted when MZL is installed in its default location, provided the GNU Scientific Library, which is used by MZL, is also installed in the same default location.

If you want to include only the necessary declarations for your functions, just cut and paste what you need from '`mzl.inc`'.

The library is installed as two files, '`libmzl.a`' and '`libmzlcfi.a`'; the latter contains the C-Fortran interface to MZL proper, and is to be used when a Fortran program needs access to MZL (see Chapter 4 [Using the library from Fortran], page 11). A shared version of the library is also installed on systems that support shared libraries. The default location of these files is '`/usr/local/lib`'. To link Fortran modules against the library you need to specify the C-Fortran interface library as well as the main library only. The following example shows how to link an application with the library,

[1] On GNU/Linux with the GNU C compiler `gcc` and either the GNU Fortran compiler `g77` or the Intel Fortran compiler for Linux `ifort`, freely available at `http://www.intel.com/software/products/`; on a Compaq Tru64 UNIX machine with `GCC` or the Compaq C compiler `CC`, and the Compaq Fortran compiler `f77`, `f90` or `f95`.

```
f90 -o example example.o -lmzlcfi -lmzl -lgsl -lgslcblas -lm
```

As MZL uses the GNU Scientific Library, GSL must also be linked to the program.

The program `mzl-config` provides information on the local version of the library. For example, the following command shows that the library has been installed under the directory '`/usr/local`',

```
bash$ mzl-config --prefix
/usr/local
```

The examples above could have been written,

```
f90 -c example.f90 `mzl-config --fflags`
f90 -o example example.o `mzl-config --libs`
```

or, since there is here only one source file,

```
f90 -o example example.f90 `mzl-config --fflags --libs`
```

Please, note the backticks in the examples above.

Further information on `mzl-config` is available using the command `mzl-config --help`.

4.4 Shared Libraries

To run a program linked with the shared version of the library it may be necessary to define the shell variable `LD_LIBRARY_PATH` to include the directory where the library is installed. For example, in the Bourne shell (`/bin/sh` or `/bin/bash`), if the library has been installed in a '`lib`' directory of the user's home directory the library path can be set with the following commands:

```
LD_LIBRARY_PATH=$HOME/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
./example
```

In the C-shell (`/bin/csh` or `/bin/tcsh`) the equivalent command is,

```
setenv LD_LIBRARY_PATH $HOME/lib:$LD_LIBRARY_PATH
```

To save retyping these commands each session they should be placed in an individual or system-wide login file.

To compile a statically linked version of the program, use the `-static` flag in `f90`,

```
f90 -o example -static example.o `mzl-config --libs`
```

With another compiler, this option may exist under a different name, or not at all; in the latter case, if you insist on using statically linked executables, you should consider installing the static version of the library only (see option `--disable-shared` of `configure`).

5 Error handling

This chapter describes the way that MZL functions report and handle errors. By examining the status information returned by every function you can determine whether it succeeded or failed, and if it failed the cause of failure will always be printed on your screen (more precisely, on the device attached to the standard error channel; this usually defaults to the screen).

5.1 Error Reporting in C

Functions that allocate memory return the `NULL` pointer to indicate an error, a valid pointer value otherwise. Functions that compute a `double` value return `HUGE_VAL` to indicate an error[1]; two utility functions, `mzl_is_valid` and `mzl_is_invalid`, are provided to check whether their argument is different from or equal to `HUGE_VAL`, respectively. Functions whose return type is `int` return `0` to indicate an error, a non-zero integer value otherwise. In case of an error, a message explaining the cause thereof is systematically printed on `stderr`.

The routines report an error whenever they cannot perform the task requested of them, and the MZL error handler is invoked. Situations like this are a normal occurrence when using any mathematical library and you should check the return status of the functions that you call.

Normally, before using MZL, you should install the MZL error handler by calling the function,

`void` **mzl_init** (`void`) Function

This function installs the MZL error handler after having safely saved the GSL error handler active at the time of the call. This GSL error handler can be reinstalled with function `mzl_cleanup`.

If the MZL error handler has not been installed, the GSL error handler active at the time of the first call to an MZL function will be invoked. If this error handler is the default GSL error handler, a message will be printed and the program will abort, possibly with a coredump. If the MZL error handler has been installed, its invocation results in an error message being printed, yet without interruption of the program flow; it is therefore up to you to check for the return status of the function and to make the appropriate decision, thus

```
int status;

mzl_init();

status = mzl_state_create_from_file ("foo.dat");
if (status)
{ /* an error occurred */
```

[1] Returning a `NaN` value on error was first considered, then abandoned due to the lack of a uniform default behaviour of the C and Fortran compilers as to the way a floating-point `NaN` is handled; namely, as to whether traps are generated or not when a `NaN` is encountered.

```
    exit(EXIT_FAILURE);
}

...

mzl_cleanup();
```

Assuming that the file '`foo.dat`' doesn't exist or can't be opened for reading, the following message will be printed,

```
mzl: ERROR: can't open datafile [read_file()]
  (generic failure)
mzl: ERROR: error reading datafile [mzl_state_create_from_file()]
  (generic failure)
```

and the program is stopped, since after checking the return status we called function `exit`; otherwise, the program would keep executing, possibly spitting out but rubbish. The names in brackets in the error messages indicate which MZL function detected the error.

If we had not installed MZL error handler, i.e., if we had omitted the call to `mzl_init`, GSL error handler would have been invoked; assuming the GSL error handler which is active is the default one, the following message would have been printed,

```
gsl: state_file.c:122: ERROR: can't open datafile [read_file()]
Default GSL error handler invoked.
Aborted
```

and the program would have aborted (possibly with a coredump).

If your program is using the GSL, you might wish to re-install the GSL error handler after you are finished with MZL function calls; this is achieved through a call to the function

`void` **mzl_cleanup** (`void`) Function

This function reinstalls the GSL error handler that was active just before the call to function `mzl_init`.

5.2 Error Reporting in Fortran

Functions that "create" or "allocate" objects in memory[2] return a negative integer value (actually, `-1`) to indicate an error, a positive integer value otherwise. Functions whose name ends in `_compute` return `0` to indicate an error, `1` otherwise. Functions that compute a `double precision` value return a physically meaningless double precision value to indicate an error; two utility functions, `mzl_is_valid` and `mzl_is_invalid`, are provided to check whether a double precision value is physically meaningful or not. Function `cfi_state_term_number` returns `0` on error, a strictly positive integer value otherwise. In case of an error, a message explaining the cause thereof is systematically printed on the error device, usually the screen.

The functions report an error whenever they cannot perform the task requested of them, and the MZL error handler is invoked. Situations like this are a normal occurrence when using any mathematical library and you should check the return status of the functions that you call.

[2] The memory allocation is actually done by underlying C functions.

Since your are calling MZL functions from Fortran, we assume that you are not using the GNU Scientific Library in your program, and no provisions are made to fiddle with the GSL error handler as explained in the previous section for C programmers. As a result, whenever a subroutine cannot perform the task requested of it, the MZL error handler prints a message, yet without interrupting the program flow; therefore, it is up to you to take the appropriate action (generally, to stop the program), thus

```
integer status

status = cfi_state_create_from_file("foo.dat")
if (status .lt. 0) ! an error occurred
  stop
end if
```

Assuming that the file '`foo.dat`' doesn't exist or can't be opened for reading, the following message will be printed,

```
mzl: ERROR: can't open datafile [read_file()]
  (generic failure)
mzl: ERROR: error reading datafile [mzl_state_create_from_file()]
  (generic failure)
```

and the program is stopped, since after checking the return status we diverted the program flow towards the end; otherwise, the program would keep executing, possibly spitting out but rubbish. The names in brackets in the error messages indicate which MZL function detected the error.

6 C functions

The library functions are divided into three groups:

- State functions for creating and initializing an `mzl_state` object representing a quantum state from data passed as arguments or read from a file, and for destroying such an object.
- Zeeman functions for creating, initializing, and destroying an `mzl_zeeman` object that contains the Landé factors.
- Transition functions for creating and initializing an `mzl_polar` object that describes the Zeeman patterns of a quantum transition, and for destroying such an object.

The access to the data structures and functions below is made through the header '`mzl.h`'.

6.1 Data Types

A molecular state is defined by an `mzl_state` structure, which contains the following compontents: `Av` and `Bv`, the spin-orbit and rotational constants, respectively, for the vibrational level, v, considered; `gamma` the spin-rotation constant for Σ sates; `lss` [short for `lambda` (spin-spin)], the spin-spin constant for $^3\Sigma$ states only (see Chapter 2 [Scientific context], page 4); `spin`, the electronic spin, S; `multiplicity`, the spin multiplicity, $2S+1$; `sigma`, a table of the spin projections, Σ, onto the intranuclear axis (recall that the size of this table is given by `multiplicity`); `lambda`, the absolute value of the projection, Λ, of the orbital momentum onto the intranuclear axis; `omega`, a table of the values of $\Omega = \Lambda + \Sigma$; `omega_size`, the size of this table; `name`, an arbitrary string used to denote the quantum state (e.g., `2Pi_u` to denote a $^2\Pi_u$ state).

The array `sigma` contains the following values: $S, \ldots, -S$.

The array `omega` contains the same values as array `sigma` in the same order if $\Lambda = 0$, and the following values (in this order): $\Lambda, \Lambda+S-1, \ldots, \Lambda-S, -\Lambda+S, -\Lambda+S-1, \ldots, -\Lambda-S$, if $\Lambda \neq 0$.

mzl_state Data type

```
typedef struct
{
  double Av;
  double Bv;
  double gamma;
  double lss;
  int multiplicity;
  double spin;
  double *sigma;
  int lambda;
  double *omega;
  int omega_size;
  char *name;
} mzl_state;
```

The constants `Av`, `Bv`, `gamma`, and `lss` are in cm^{-1} units. (They correspond to the usual molecular constants A_v, B_v, γ and λ, respectively).

Note that the molecular constants have to be provided by the user, who can find some useful information in Huber & Herzberg (see Section 2.3 [References], page 5) or on-line at `http://webbook.nist.gov/chemistry/`.

The Landé factors of each component of a multiplet are stored in the `g` component of a `mzl_zeeman` structure. Such a structure has three other components: `size`, the number of Landé factors; `degeneracy`, a number which equals 1 or 2 according to whether $\Lambda = 0$ or not.

mzl_zeeman Data type

```
typedef struct
{
  int size;
  int degeneracy;
  double *g;
  mzl_hamiltonian *hamiltonian;
} mzl_zeeman;
```

The last component, `hamiltonian`, is not public (i.e., is not meant to be accessed by the end user), and will therefore not be documented here.

The transitions between Zeeman sublevels of two multiplets are described by one `mzl_polar` object for each of the polarisations corresponding to $\Delta M = -1, 0, 1$. The components of an `mzl_polar` object are: `size`, the *allocated* space for each of the two following tables: `dnu`, the frequency shifts in $\mu_B H$ units, where μ_B is the Bohr magneton and H the magnetic field strength (in Gauss); and `Irel`, the corresponding relative intensities.

The useful space in the above tables, i.e. the entries corresponding to allowed transitions, is comprised between the indices `begin` and `end`; more precisely, the useful space lies in the semi-open interval [`begin`, `end`[. If, for some reason, the user needs to now the value of M in the initial state corresponding to entry k such that `begin` $\leq k <$ `end`, this is given by $M = J_i - (\texttt{begin} - k)$ (where J_i is the value of the angular momentum in the initial state).

mzl_polar Data type

```
typedef struct
{
  int size;
  int begin;
  int end;
  double *dnu;
  double *Irel;
} mzl_polar;
```

6.2 State Functions

`mzl_state *` **mzl_state_create** (`double` *Av*, `double` *Bv*, `double` *gamma*, `double` *lss*, `int` *mult*, `int` *lamdba*, `const char *`*name*) Function

This function allocates an `mzl_state` structure, initializing it with the values passed as arguments. If the structure cannot be allocated then the MZL error handler, if it has been installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active a the time of the call is invoked, and a `NULL` pointer is returned.

`mzl_state *` **mzl_state_create_from_file** (`const char *`*filename*) Function

This function allocates an `mzl_state` structure, initializing it with the values from the file *filename*; the file is closed after it has been read. If the file cannot be opened or the structure cannot be allocated then the MZL error handler, if it has been installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active a the time of the call is invoked, and a `NULL` pointer is returned.

The data file *filename* must conform to the following structure,

Structure of a datafile Data file

```
# NH 3Sigma state (v = 0)
Av = 0        # spin-orbit
Bv = 16.33    # rotation
gamma=-0.04   # spin-rotation
lss =0.45     # lambda (spin-spin)
Mult = 3      # Multiplicity
   Lambda = 0 # Projection of electronic angular momentum
Name = "NH 3Sigma state (v = 0)"
```

The data may be entered in any line order. The part of a line from the '`#`' sign up to its end is a comment. White space is ignored, except within double quotes. Empty lines as well as blank lines are ignored. The case of the characters in the symbols at the left of the equal sign is *not* significant. There may be no more than one specification per line. Only the lines for specifying the constants `Av`, `Bv`, `Mult`, and `Lambda` are mandatory. If `gamma` and `lss` are left unspecified then they will be ascribed the value 0.0. If `Name` is left unspecified then a generic name will be derived from the values of `Mult` and `Lambda` (e.g., '`2Pi`' if `Mult = 2` and `Lambda = 1`). As states with $\Lambda > 3$ rarely occur, the generic name in such cases would be '`unknown name`'. Should a constant be mistakenly defined several times in the data file, this results in no error as long as the constant is ascribed the same value; if the value differs, the MZL error handler is invoked, and `NULL` pointer is returned. Should the name of the state be mistakenly defined several times in the data file, the last value will be retained. The name of a state is neither checked nor used, it is there for information purposes only.

`void` **mzl_state_destroy** (`mzl_state *`*state*) Function

This function releases the memory previously allocated for an `mzl_state` object.

`int` **mzl_state_term_number** (`const mzl_state *`*state*, `double` *omega*) Function

This auxiliary function returns the energy term number corresponding to a state *state* with a value *omega* of $\Omega = \Lambda + \Sigma$ (see Herzberg, and Huber & Herzberg in

Section 2.3 [References], page 5 for a correspondance between a value of *omega* and a term number). On success, the value returned is greater or equal to 1; otherwise, the MZL error handler, if it has been installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active a the time of the call is invoked, and -1 is returned.

6.3 Zeeman Functions

`mzl_zeeman *` **mzl_zeeman_alloc** (`const mzl_state *`*state*) Function
`mzl_zeeman *` **mzl_zeeman_hund_a_alloc** (`const mzl_state *`*state*) Function

`mzl_zeeman *` **mzl_zeeman_hund_b_alloc** (`const mzl_state *`*state*) Function

These functions allocate space for an `mzl_zeeman` object that will be used to compute and hold the Landé factors of a state *state*. If the memory allocation fails, the MZL error handler, if installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active at the time of the function call is invoked, and a `NULL` pointer is returned. The first function is used for any state intermediate between Hund (a) and Hund (b) case, whereas the other two are specialized for cases Hund (a) and (b), respectively.

`int` **mzl_zeeman_compute** (`mzl_zeeman *`*z*`, double` *J*) Function
`int` **mzl_zeeman_hund_a_compute** (`mzl_zeeman *`*z*`, double` *J*) Function
`int` **mzl_zeeman_hund_b_compute** (`mzl_zeeman *`*z*`, double` *J*) Function
These functions compute the Landé factors for the state that has been used to create the `mzl_zeeman` structure, for any value of the total angular momentum (exclusive of nuclear spin), *J*. If the computation fails, the MZL error handler, if installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active at the time of the function call is invoked, and the functions return 0; in case of success, they return 1. The first function is used for any state intermediate between Hund (a) and Hund (b) case, whereas the other two are specialized for cases Hund (a) and (b), respectively.

`void` **mzl_zeeman_free** (`mzl_zeeman *`*z*) Function
`void` **mzl_zeeman_hund_a_free** (`mzl_zeeman *`*z*) Function
`void` **mzl_zeeman_hund_b_free** (`mzl_zeeman *`*z*) Function
These functions release the memory allocated for an `mzl_zeeman` object. Actually, the last two functions are mere aliases of the first function.

`double` **mzl_lande_of_term** (`const mzl_zeeman *`*z*`, int` *term*) Function
This auxiliary function returns the Landé factor corresponding to the energy term *term* of a multiplet. Should the term number indicated be incorrect (i.e., less than 1) or out of bounds, the MZL error handler, if installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active at the time of the function call is invoked, and the function returns the value `HUGE_VAL` (the utility function `mzl_is_invalid` tests if its argument is `HUGE_VAL` or not).

`double` **mzl_zeeman_eval** (`const mzl_zeeman *`*z*`, int` *i*) Function

This function returns the *i*-th eigenvalue of the Hamiltonian (0 <= *i* < *z*->`size`). Should the index *i* be out of bound, or should the function be called for an `mzl_zeeman` structure corresponding to pure Hund (a) or (b) cases, the MZL error handler, if installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active at the time of the function call is invoked, and the function returns the value `HUGE_VAL` (the utility function `mzl_is_invalid` tests if its argument is `HUGE_VAL` or not).

`double *` **mzl_zeeman_evec** (`const mzl_zeeman *`*z*`, int` *i*) Function

This function returns the *i*-th eigenvector of the Hamiltonian (0 <= *i* < *z*->`size`) as a dynamically allocated array. Should this function be called for an `mzl_zeeman` structure corresponding to pure Hund (a) or (b) cases, or should the memory allocation fail, the MZL error handler, if installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active at the time of the function call is invoked, and the function returns the `NULL` pointer. It is of the responsibility of the user to free (using the standard C function `free`) the allocated memory space when it is no longer used.

6.4 Transition Functions

`mzl_polar *` **mzl_polar_create** (`double` *Jf*`, double` *gf*`, double` *Ji*`, double` *gi*`, int` *dm*) Function

This function allocates and initializes an `mzl_polar` object for a transition of given *dm* between an initial state of total angular momentum *Ji* and Landé factor *gi*, and a final state of total angular momentum *Jf* and Landé factor *gf*. *dm* corresponds to $\Delta M = M_f - M_i$ and is -1, 0, or 1 [M_f (M_i) is the projection of J_f (J_i) onto the intranuclear axis]. If the allocation fails, the MZL error handler, if installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active at the time of the function call is invoked, and the function returns a '`NULL`' pointer. It should be noted that the size of the allocated tables `dnu` and `Irel` are too large by 2 units when *Jf* is less than *Ji*, which is of no consequence since the corresponding values of the relative intensities in table `Irel` are zero; recall that the useful entries in these tables are between indices `begin` and `end` (exclusive of the latter).

`void` **mzl_polar_destroy** (`mzl_polar *`*p*) Function

This function releases the memory previously allocated to an `mzl_polar` object.

`double` **mzl_average_freq_shift** (`const mzl_polar *`*p*) Function

This function returns the average frequency shift, each frequency shift being weighted by its relative intensity, for a transition described by *p*. This function cannot fail.

`double` **mzl_weighted_sigmas_separation** (`const mzl_polar *`*sigma_plus*`, const mzl_polar *`*sigma_minus*) Function

This function returns the absolute value of the frequency separation, symbolically $| < \sigma^+ > - < \sigma^- > |$, of the weighted mean positions of the two σ-patterns ($\Delta M = \pm 1$) in MHz/G. This function cannot fail.

`double` **mzl_effective_lande** (`double` *Ji*, `double` *gi*, `double` *gf*, `int` *dJ*) Function

This function returns the effective Landé factor as defined by Berdyugina and Solanki (see Section 2.3 [References], page 5). This function cannot fail.

`double` **mzl_zeeman_intensity** (`double` *J*, `double` *M*, `int` *dJ*, `int` *dM*) Function

This auxiliary function computes the relative intensity of a transition between Zeeman sublevels $|JM>$ and $|J+dJ, M+dM>$. In case of failure, the MZL error handler, if installed (see Section 5.1 [Error Reporting in C], page 14), or the GSL error handler active at the time of the function call is invoked, and the function returns the value '`HUGE_VAL`' (the utility function `mzl_is_invalid` tests if its argument is `HUGE_VAL` or not).

6.5 Utility Function

`int` **mzl_is_invalid** (`double val`) Function

This utility function returns 1 if its argument is `HUGE_VAL`, which considered here an invalid double value, 0 otherwise. This function cannot fail.

`int` **mzl_is_valid** (`double val`) Function

This utility function returns 1 if its argument is not equal to `HUGE_VAL`, 0 otherwise. This function is the converse of the previous function. This function cannot fail.

7 C-Fortran interface

The following functions are part of the library '`libmzlcfi.a`', the C-Fortran interface to the MZL library. They are the counterparts of the C functions presented in the preceding section; actually, they are wrappers of these functionss. The dynamically created objects are referred in the Fortran code by integers, in much the same way as the IO logical units. As these objects are actually allocated as C structures, the read access to the members of these structures is provided by a number of auxiliary functions and subroutines[1].

In order to avoir possible name clashes, all functions and subroutines of the C-Fortran interface are prefixed by `cfi_`.

The Fortran functions are divided into three groups:

- State functions for creating and initializing an `mzl_state` object, representing a quantum state, from data passed as arguments or read from a file, and for destroying such an object.
- Zeeman functions for creating, initializing, and destroying an `mzl_zeeman` object that contains the Landé factors.
- Transition functions for creating and initializing an `mzl_polar` object that describes the Zeeman patterns of a quantum transition, and for destroying such an object.

The access to the functions below is made through the header '`mzl.inc`'.

The user should keep in mind that these functions and subroutines are just interface procedures written in C and not in Fortran, and that their description, below, as pseudo-Fortran 90 prototypes is just meant to ease their use. These objects are not meant to be manipulated directly from Fortran.

7.1 Data Types

A molecular state is defined by a `mzl_state` structure, which contains the following compontents: `Av` and `Bv`, the spin-orbit and rotational constants, respectively, for the vibrational level, v, considered; `gamma` the spin-rotation constant for Σ sates; `lss` [short for `lambda` (spin-spin)], the spin-spin constant for $^3\Sigma$ states only (see Chapter 2 [Scientific context], page 4); `spin`, the electronic spin, S; `multiplicity`, the spin multiplicity, $2S + 1$; `sigma`, a table of the spin projections, Σ onto the intranuclear axis (recall that the size of this table is `multiplicity`); `lambda`, the absolute value of the projection, Λ, of the orbital momentum onto the intranuclear axis; `omega`, a table of the values of $\Omega = \Lambda + \Sigma$; `omega_size`, the size of this table; `name`, an arbitrary string used to denote the quantum state (e.g., `2Pi_u` to denote a $^2\Pi_u$ state).

The array `sigma` contains the following values: $S, \ldots, -S$.

The array `omega` contains the same values as array `sigma` in the same order if $\Lambda = 0$, and the following values (in this order): $\Lambda, \Lambda+S-1, \ldots, \Lambda-S, -\Lambda+S, -\Lambda+S-1, \ldots, -\Lambda-S$, if $\Lambda \neq 0$.

An `mzl_state` object can be viewed as the following pseudo-Fortran 90 derived type:

[1] No write access is provided, as this has no meaning in the present implementation.

mzl_state Data type

```
TYPE mzl_state
  DOUBLE PRECISION :: Av
  DOUBLE PRECISION :: Bv
  DOUBLE PRECISION :: gamma
  DOUBLE PRECISION :: lss
  INTEGER :: multiplicity
  DOUBLE PRECISION :: spin
  DOUBLE PRECISION, DIMENSION(...) :: sigma
  INTEGER :: lambda
  DOUBLE PRECISION, DIMENSION(...) :: omega
  INTEGER :: omega_size
  STRING :: name
END TYPE mzl_state
```

where the ellipsis are just to indicate that the size of the two arrays—as well as the length of the character string—is dynamically adjusted when the object is created (recall: by a call to a C function resorting to dynamical memory allocation).

The constants `Av`, `Bv`, `gamma`, and `lss` are in cm^{-1} units. (They correspond to the usual molecular constants A_v, B_v, γ and λ, respectively).

Note that the molecular constants have to be provided by the user, who can find some useful information in Huber & Herzberg (see Section 2.3 [References], page 5) or on-line at `http://webbook.nist.gov/chemistry/`.

The Landé factors of each component of a multiplet are stored in the `g` component of a `mzl_zeeman` object. Such an object has three other components: `size`, the number of Landé factors; `degeneracy`, a number which equals 1 or 2 according to whether $\Lambda = 0$ or not.

An `mzl_zeeman` object can be viewed as the following pseudo-Fortran 90 derived type:

mzl_zeeman Data type

```
TYPE mzl_zeeman
  INTEGER :: size
  INTEGER :: degeneracy
  DOUBLE PRECISION, DIMENSION(...) :: g
  TYPE(mzl_hamiltonian) :: hamiltonian
END TYPE mzl_zeeman
```

where the ellipsis are just to indicate that the size of the array is dynamically adjusted when the object is created (recall: by a call to a C function resorting to dynamical memory allocation).

The last component, `hamiltonian`, is not public (i.e., is not meant to be accessed by the end user), and is therefore not to be documented here.

The transitions between Zeeman sublevels of two multiplets are described by one `mzl_polar` object for each of the polarisations corresponding to $\Delta M = -1, 0, 1$. The components of an `mzl_polar` object are: `size`, the size of the *allocated* space for each the two following

tables: `dnu`, the frequency shifts in $\mu_B H$ units, where μ_B is the Bohr magneton and H the magnetic field strength (in Gauss); and `Irel`, the corresponding relative intensities.

The useful space in the above tables, i.e. the entries corresponding to allowed transitions, is comprised between the indices `begin` and `end`; more precisely, the useful space lies in the interval [`begin`, `end`]. If, for some reason, the user needs to know the value of M in the initial state corresponding to entry k such that `begin` $\leq k \leq$ `end`, this is given by $M = J_i - (\texttt{begin} - k)$ (where J_i is the value of the angular momentum in the initial state).

An `mzl_polar` object can be viewed as the following pseudo-Fortran 90 derived type:

mzl_polar Data type

```
TYPE mzl_polar
  INTEGER :: size
  INTEGER :: begin
  INTEGER :: end
  DOUBLE PRECISION, DIMENSION(...) :: dnu
  DOUBLE PRECISION, DIMENSION(...) :: Irel
END TYPE mzl_polar
```

where the ellipsis are just to indicate that the size of the arrays are dynamically adjusted when the object is created (recall: by a call to a C function resorting to dynamical memory allocation).

7.2 State Functions

`INTEGER` **cfi_state_create** (*av*, *bv*, *gamma*, *lss*, *mult*, *lambda*, *name*) Function

DOUBLE PRECISION, INTENT(IN) :: *av*, *bv*, *gamma*, *lss*

INTEGER, INTENT(IN) :: *mult*, *lambda*

STRING, INTENT(IN) :: *name*

This function allocates an `mzl_state` object, initializing it with the values passed in arguments. If the object cannot be allocated then the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and -1 is returned.

`INTEGER` **cfi_state_create_from_file** (*filename*) Function

STRING, INTENT(IN) :: *filename*

This function allocates an `mzl_state` object, initializing it with the values from the file *filename*; the file is closed after it has been read. If the file cannot be opened or the object cannot be allocated then the MZL error handler is invoked, and -1 is returned.

The data file *filename* must conform to the following structure,

Structure of a datafile Data file

```
# NH 3Sigma state (v = 0)
```

```
Av = 0        # spin-orbit
Bv = 16.33    # rotation
gamma=-0.04   # spin-rotation
lss =0.45     # lambda (spin-spin)
Mult = 3      # Multiplicity
   Lambda = 0 # Projection of electronic angular momentum
Name = "NH 3Sigma state (v = 0)"
```

The data may be entered in any line order. The part of a line from the '`#`' sign up to its end is a comment. White space is ignored, except within double quotes. Empty lines as well as blank lines are ignored. The case of the characters in the symbols at the left of the equal sign is *not* significant. There may not be more than one specification per line. Only the lines for specifying the constants `Av`, `Bv`, `Mult`, and `Lambda` are mandatory. If `gamma` and `lss` are left unspecified then they will be ascribed the value 0.0. If `Name` is left unspecified then a generic name will be derived from the values of `Mult` and `Lambda` (e.g., '`2Pi`' if `Mult = 2` and `Lambda = 1`). As states with $\Lambda > 3$ rarely occur, the generic name in such cases would be '`unknown name`'. Should a constant be mistakenly defined several times in the data file, this results in no error as long as the constant is ascribed the same value; if the value differs, then the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and -1 is returned. Should the name of the state be mistakenly defined several times in the data file, the last value will be retained. The name of a state is neither checked nor used, it is there for information purposes only.

cfi_state_destroy (*state*) Subroutine

INTEGER, INTENT(IN) :: *state*

This function releases the memory previously allocated for an `mzl_state` object.

The following auxiliary functions provide read access to the members of an `mzl_state` object. Each of these functions checks its argument, which must correspond to a valid value returned by `cfi_state_create` or `cfi_state_create_from_file`; if the argument *state* is invalid then a message is printed and the program is exited.

`DOUBLE PRECISION` **cfi_state_Av** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the value of component `Av` of state *state*.

`DOUBLE PRECISION` **cfi_state_Bv** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the value of component `Bv` of state *state*.

`DOUBLE PRECISION` **cfi_state_gamma** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the value of component `gamma` of state *state*.

`DOUBLE PRECISION` **cfi_state_lss** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the value of component `lss` of state *state*.

`DOUBLE PRECISION` **cfi_state_spin** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the value of component `spin` of state *state*.

`INTEGER` **cfi_state_multiplicity** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the value of component `multiplicity` of state *state*.

`INTEGER` **cfi_state_lambda** (*state*) Function

state: INTEGER

This function returns the value of component `lambda` of state *state*.

`INTEGER` **cfi_state_omega_size** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the size of array `omega` of state *state*.

`INTEGER` **cfi_state_term_number** (*state*, *omega*) Function

INTEGER, INTENT(IN) :: *state*

double precision, INTENT(IN) :: *omega*

This auxiliary function returns the energy term number corresponding to a state *state* with a value *omega* of $\Omega = \Lambda + \Sigma$ (see Herzberg, and Huber & Herzberg in Section 2.3 [References], page 5 for a correspondance between a value of *omega* and a term number). On success, the value returned is greater or equal to 1; otherwise, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and -1 is returned.

cfi_state_name (*state, name*) Subroutine

INTEGER, INTENT(IN) :: *state*

STRING, INTENT(OUT) :: *name*

This function returns the value of the component `name` of the state *state* as a string of characters (whose length is predefined to be 80).

`INTEGER` **cfi_state_name_length** (*state*) Function

INTEGER, INTENT(IN) :: *state*

This function returns the length of the character string storing the component `name` of *state*.

cfi_state_sigma (*state, sg*) Subroutine

INTEGER, INTENT(IN) :: *state*

DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: *sg*

This subroutine copies the values of the `sigma` array of state *state* to array *sg* (whose dimension must have been correctly defined before the call to the subroutine).

cfi_state_omega (*state*, *om*) Subroutine

INTEGER, INTENT(IN) :: *state*

DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: *om*

This subroutine copies the values of the `omega` array of state *state* to array *om* (whose dimension must have been correctly defined before the call to the subroutine).

`DOUBLE PRECISION` **cfi_omega_val** (*state*, *i*) Function

INTEGER, INTENT(IN) :: *state*, *i*

This function returns the *i*-th value of the `omega` array of state *state*. *i* is a Fortran index, i.e., 1-based. If *i* is less than 1 or out of bounds, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and a physically meaningless value is returned (the utility function `cfi_is_invalid` tests if its argument is physically meaningless).

`DOUBLE PRECISION` **cfi_sigma_val** (*state*, *i*) Function

INTEGER, INTENT(IN) :: *state*, *i*

This function returns the *i*-th value of the `sigma` array of state *state*. *i* is a Fortran index, i.e., 1-based. If *i* is less than 1 or out of bounds, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and a physically meaningless value is returned (the utility function `cfi_is_invalid` tests if its argument is physically meaningless).

7.3 Zeeman Functions

`INTEGER` **cfi_zeeman_alloc** (*state*) Function
`INTEGER` **cfi_zeeman_hund_a_alloc** (*state*) Function
`INTEGER` **cfi_zeeman_hund_b_alloc** (*state*) Function

INTEGER, INTENT(IN) :: *state*

These functions allocate space for an `mzl_zeeman` object that will be used to compute and hold the Landé factors of a state *state*. If the memory allocation fails, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and -1 is returned. The first function is used for any state intermediate between Hund (a) and Hund (b) case, whereas the other two are specialized for cases Hund (a) and (b), respectively.

`INTEGER` **cfi_zeeman_compute** (*z*, *J*) Function
`INTEGER` **cfi_zeeman_hund_a_compute** (*z*, *J*) Function
`INTEGER` **cfi_zeeman_hund_b_compute** (*z*, *J*) Function

INTEGER, INTENT(IN) :: *z*

DOUBLE PRECISION, INTENT(IN) :: *J*

These functions compute the Landé factors for the state that has been used to create the `mzl_zeeman` object, for any value of the total angular momentum (exclusive of nuclear spin), *J*. If the computation fails, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and the functions return 0; in case

of success, they return 1. The first function is used for any state intermediate between Hund (a) and Hund (b) case, whereas the other two are specialized for cases Hund (a) and (b), respectively.

cfi_zeeman_free (*z*) Subroutine
cfi_zeeman_hund_a_free (*z*) Subroutine
cfi_zeeman_hund_b_free (*z*) Subroutine

INTEGER, INTENT(IN) :: *z*

These functions release the memory allocated for an `mzl_zeeman` object, *z*. Actually, the last two functions are mere aliases of the first function.

`DOUBLE PRECISION` **cfi_lande_of_term** (*z*, *term*) Function

INTEGER, INTENT(IN) :: *z*, *term*

This auxiliary function returns the Landé factor corresponding to the energy term *term* of the multiplet to which the `mzl_zeeman` object *z* corresponds. Should the term number indicated be incorrect (i.e., less than 1) or out of bounds, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and the function returns a physically meaningless value (the utility function `cfi_is_invalid` tests if its argument is physically meaningless).

The following auxiliary functions provide read access to the public members of an `mzl_zeeman` object. Each of these functions checks its argument, which must correspond to a valid value returned by `cfi_zeeman_alloc`, `cfi_zeeman_hund_a_alloc` or `cfi_zeeman_hund_b_alloc`; if the argument *z* is invalid then a message is printed and the program is exited.

`INTEGER` **cfi_zeeman_size** (*z*) Function

INTEGER, INTENT(IN) :: *z*

This function returns the value of component `size` of the `mzl_zeeman` object *z*.

`INTEGER` **cfi_zeeman_degeneracy** (*z*) Function

INTEGER, INTENT(IN) :: *z*

This function returns the value of component `degeneracy` of the `mzl_zeeman` object *z*.

cfi_zeeman_lande (*z*, *lande*) Subroutine

INTEGER, INTENT(IN) :: *z*

DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: *lande*

This subroutine copies the elements of `g` array to indicated array *lande*, whose predeclared dimension must be large enough to accomodate as many values as given by the member `size` of the `mzl_zeeman` object *z*.

7.4 Transition Functions

`INTEGER` **cfi_polar_create** (*Jf*, *gf*, *Ji*, *gi*, *dm*) Function

DOUBLE PRECISION, INTENT(IN) :: *Jf*, *gf*, *Ji*, *gi*

INTEGER, INTENT(IN) :: *dm*

This function allocates and initializes an `mzl_polar` object for a transition of given *dm* between an initial state of total angular momentum *Ji* and Landé factor *gi*, and a final state of total angular momentum *Jf* and Landé factor *gf*. *dm* corresponds to $\Delta M = M_f - M_i$ and is -1, 0, or 1 (M_f (M_i) is the projection of J_f (J_i) onto the intranuclear axis. If the allocation fails, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and the function returns a '`-1`'. (It should be noted that the size of the allocated tables `dnu` and `Irel` are too large by 2 units when *Jf* is less than *Ji*, which is of no consequence since the corresponding values of the relative intensities in table `Irel` are zero.)

cfi_polar_destroy (*p*) Subroutine

INTEGER, INTENT(IN) :: *p*

This function releases the memory previously allocated to an `mzl_polar` object, *p*.

`DOUBLE PRECISION` **cfi_average_freq_shift** (*p*) Function

INTEGER, INTENT(IN) :: *p*

This function returns the average frequency shift, each frequency shift being weighted by its relative intensity, corresponding to the `mzl_polar` object *p*. This function cannot fail (except if its argument *p* is invalid, in which case an error message is printed and the program is exited).

`DOUBLE PRECISION` **cfi_weighted_sigmas_separation** (*sigma_plus*, *sigma_minus*) Function

INTEGER, INTENT(IN) :: *sigma_plus*, *sigma_minus*

This function returns the absolute value of the frequency separation (in MHz/G), symbolically $| < \sigma^+ > - < \sigma^- > |$, of the weighted mean positions of the two σ-patterns ($\Delta M = \pm 1$) described by the `mzl_polar` objects *sigma_plus* and *sigma_minus*. This function cannot fail.

`DOUBLE PRECISION` **cfi_effective_lande** (*Ji*, *gi*, *gf*, *dJ*) Function

DOUBLE PRECISION, INTENT(IN) :: *Ji*, *gi*, *gf*

INTEGER, INTENT(IN) :: *dJ*

This function returns the effective Landé factor as defined by Berdyugina and Solanki (see Section 2.3 [References], page 5). This function cannot fail.

`DOUBLE PRECISION` **cfi_zeeman_intensity** (*J*, *M*, *dJ*, *dM*) Function

DOUBLE PRECISION, INTENT(IN) :: *J*, *M*

INTEGER, INTENT(IN) :: *dJ*, *dM*

This auxiliary function computes the relative intensity of a transition between Zeeman sublevels $|JM>$ and $|J+dJ, M+dM>$. In case of failure, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and the function returns a physically meaningless value (the utility function `cfi_is_invalid` tests if its argument is physically meaningless).

The following auxiliary functions provide read access to the members of an `mzl_polar` object. Each of these functions checks its argument, which must correspond to a valid value returned by `cfi_polar_create`; if the argument p is invalid then a message is printed and the program is exited.

`INTEGER` **cfi_polar_size** (p) Function

INTEGER, INTENT(IN) :: p

This function returns the size of the two arrays, `dnu` and `Irel`, members of an `mzl_polar` object.

`INTEGER` **cfi_polar_begin** (p) Function

INTEGER, INTENT(IN) :: p

This function returns the Fortran (i.e., 1-based) index of the first useful entry in the two arrays, `dnu` and `Irel`, members of an `mzl_polar` object.

`INTEGER` **cfi_polar_end** (p) Function

INTEGER, INTENT(IN) :: p

This function returns the Fortran (i.e., 1-based) index of the last useful entry in the two arrays, `dnu` and `Irel`, members of an `mzl_polar` object.

cfi_polar_dnu (p, *delta_nu*) Subroutine

INTEGER, INTENT(IN) :: p

DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: *delta_nu*

This subroutine copies the values of the `dnu` array of the `mzl_polar` object p to array *delta_nu* (whose dimension must have been correctly defined before the call to the subroutine). If the angular momentum is lower in the final state than in the initial state of the molecular transition, this array has two unused entries, and the useful entries are those whose index is between `begin` and `end`, inclusive.

`DOUBLE PRECISION` **cfi_polar_dnu_val** (p, i) Function

INTEGER, INTENT(IN) :: p, i

This function returns the i-th value of the `dnu` array of transition p. i is a Fortran index, i.e., 1-based, and must be comprised in `begin` and `end`, inclusive. In case of failure, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and the function returns a physically meaningless value (the utility function `cfi_is_invalid` tests if its argument is physically meaningless).

cfi_polar_Irel (*p, Ir*) Subroutine

INTEGER, INTENT(IN) :: *p*

DOUBLE PRECISION, DIMENSION(:), INTENT(OUT) :: *Ir*

This subroutine copies the values of the `Irel` array of the `mzl_polar` object *p* to array *Ir* (whose dimension must have been correctly defined before the call to the subroutine). If the angular momentum is lower in the final state than in the initial state of the molecular transition, this array has two unused entries, and the useful entries are those whose index is between `begin` and `end` inclusive.

`DOUBLE PRECISION` **cfi_polar_Irel_val** (*p*, *i*) Function

INTEGER, INTENT(IN) :: *p*, *i*

This function returns the *i*-th value of the `Irel` array of transition *p*. *i* is a Fortran index, i.e., 1-based, and must be comprised in `begin` and `end`, inclusive. In case of failure, the MZL error handler (see Section 5.2 [Error Reporting in Fortran], page 15) is invoked, and the function returns a physically meaningless value (the utility function `cfi_is_invalid` tests if its argument is physically meaningless).

7.5 Utility Functions

As some functions of the C-Fortran interface which return a double precision number return a physically meaningless value in case of an error, two utility functions `cfi_is_invalid` and `cfi_is_valid` are provided for testing if their argument is physically meaningless or not, respectively.

`INTEGER` **cfi_is_invalid** (*x*) Function

DOUBLE PRECISION, INTENT(IN) :: *x*

This function returns 1 if *x* is physically meaningless, 0 otherwise.

`INTEGER` **cfi_is_valid** (*x*) Function

DOUBLE PRECISION, INTENT(IN) :: *x*

This function returns 1 if *x* is not physically meaningless, 0 otherwise. This function is the converse of the previous one.

Free Software Needs Free Documentation

The following article was written by Richard Stallman, founder of the GNU Project.

The biggest deficiency in the free software community today is not in the software—it is the lack of good free documentation that we can include with the free software. Many of our most important programs do not come with free reference manuals and free introductory texts. Documentation is an essential part of any software package; when an important free software package does not come with a free manual and a free tutorial, that is a major gap. We have many such gaps today.

Consider Perl, for instance. The tutorial manuals that people normally use are non-free. How did this come about? Because the authors of those manuals published them with restrictive terms—no copying, no modification, source files not available—which exclude them from the free software world.

That wasn't the first time this sort of thing happened, and it was far from the last. Many times we have heard a GNU user eagerly describe a manual that he is writing, his intended contribution to the community, only to learn that he had ruined everything by signing a publication contract to make it non-free.

Free documentation, like free software, is a matter of freedom, not price. The problem with the non-free manual is not that publishers charge a price for printed copies—that in itself is fine. (The Free Software Foundation sells printed copies of manuals, too.) The problem is the restrictions on the use of the manual. Free manuals are available in source code form, and give you permission to copy and modify. Non-free manuals do not allow this.

The criteria of freedom for a free manual are roughly the same as for free software. Redistribution (including the normal kinds of commercial redistribution) must be permitted, so that the manual can accompany every copy of the program, both on-line and on paper.

Permission for modification of the technical content is crucial too. When people modify the software, adding or changing features, if they are conscientious they will change the manual too—so they can provide accurate and clear documentation for the modified program. A manual that leaves you no choice but to write a new manual to document a changed version of the program is not really available to our community.

Some kinds of limits on the way modification is handled are acceptable. For example, requirements to preserve the original author's copyright notice, the distribution terms, or the list of authors, are ok. It is also no problem to require modified versions to include notice that they were modified. Even entire sections that may not be deleted or changed are acceptable, as long as they deal with nontechnical topics (like this one). These kinds of restrictions are acceptable because they don't obstruct the community's normal use of the manual.

However, it must be possible to modify all the *technical* content of the manual, and then distribute the result in all the usual media, through all the usual channels. Otherwise, the restrictions obstruct the use of the manual, it is not free, and we need another manual to replace it.

Please spread the word about this issue. Our community continues to lose manuals to proprietary publishing. If we spread the word that free software needs free reference

manuals and free tutorials, perhaps the next person who wants to contribute by writing documentation will realize, before it is too late, that only free manuals contribute to the free software community.

If you are writing documentation, please insist on publishing it under the GNU Free Documentation License or another free documentation license. Remember that this decision requires your approval—you don't have to let the publisher decide. Some commercial publishers will use a free license if you insist, but they will not propose the option; it is up to you to raise the issue and say firmly that this is what you want. If the publisher you are dealing with refuses, please try other publishers. If you're not sure whether a proposed license is free, write to licensing@gnu.org.

You can encourage commercial publishers to sell more free, copylefted manuals and tutorials by buying them, and particularly by buying copies from the publishers that paid for their writing or for major improvements. Meanwhile, try to avoid buying non-free documentation at all. Check the distribution terms of a manual before you buy it, and insist that whoever seeks your business must respect your freedom. Check the history of the book, and try reward the publishers that have paid or pay the authors to work on it.

The Free Software Foundation maintains a list of free documentation published by other publishers, at http://www.fsf.org/doc/other-free-books.html.

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *one line to give the program's name and a brief idea of what it does.*
> Copyright (C) *yyyy name of author*
>
> This program is free software; you can redistribute it
> and/or modify it under the terms of the GNU General Public
> License as published by the Free Software Foundation; either
> version 2 of the License, or (at your option) any later
> version.
>
> This program is distributed in the hope that it will be
> useful, but WITHOUT ANY WARRANTY; without even the implied
> warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
> PURPOSE. See the GNU General Public License for more
> details.
>
> You should have received a copy of the GNU General Public
> License along with this program; if not, write to the Free
> Software Foundation, Inc., 59 Temple Place - Suite 330,
> Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) 19*yy name of author*
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
> type 'show w'. This is free software, and you are welcome
> to redistribute it under certain conditions; type 'show c'
> for details.

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in
> the program 'Gnomovision' (which makes passes at compilers)
> written by James Hacker.
>
> *signature of Ty Coon*, 1 April 1989

```
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgments" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant

Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this

License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) *year your name*. Permission is
> granted to copy, distribute and/or modify this document
> under the terms of the GNU Free Documentation License,
> Version 1.1 or any later version published by the Free
> Software Foundation; with the Invariant Sections being
> *list their titles*, with the Front-Cover Texts being
> *list*, and with the Back-Cover Texts being
> *list*. A copy of the license is included in the
> section entitled ``GNU Free Documentation License''.

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being *list*"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Function Index

C

M

Type Index

M

S

Concept Index

Table of Contents