

Last modified 10 minutes ago

FORTRAN CHS lib Project

What's the aim?

The goal is to provide a library of general Fortran modules which are used by the whole CHS department.

[FORTRAN CHS lib Project](#)
[What's the aim?](#)
[What is already there?](#)
[Note on Numerical Recipes](#)
[How to checkout the complete FORTRAN lib?](#)
[How to contribute to the FORTRAN lib?](#)

What is already there?

Alphabetical ordered modules:

Name	One-Line description
mo_constants	Mathematical and physical constants
mo_jday	Converts Julian Day into Day, Month and Year, and vice versa
mo_kind	Definition of number precision
mo_moment	1st to 4th moments, central and mixed central moments
mo_ncread	Reading nc files using the netcdf4 library
mo_nr	Main numerical recipes module containing the interfaces
mo_nrutil	Numerical recipes utilities module
mo_sort	Quicksort arrays or indices
mo_template	Module template demonstrating the coding standard of the library
mo_xor4096	Generating Uniform/ Gaussian Random Numbers using the xor4096 algorithm

Categorical ordered modules:

Category	Name	One-Line description
Date/ Time		
	mo_jday	Converts Julian Day into Day, Month and Year, and vice versa
Input/ Output		
	mo_ncread	Reading nc files using the netcdf4 library
Optimization		
Miscellaneous		
	mo_constants	Mathematical and physical constants
	mo_kind	Definition of number precision
	mo_nr	Main numerical recipes module containing the interfaces
	mo_nrutil	Numerical recipes utilities module
	mo_template	Module template demonstrating the coding standard of the library
Math		
	mo_sort	Quicksort arrays or indices
	mo_xor4096	Generating Uniform/ Gaussian Random Numbers using the xor4096 algorithm
Statistics		
	mo_moment	1st to 4th moments, central and mixed central moments

Please update the two tables above, if you uploaded a tested, working module. If your module is still not documented or in beta-version please do NOT list it in the tables.

Note on Numerical Recipes

- The directory [nr_ori](#) contains the original *Numerical Recipes in Fortran 90* routines. It also includes PDFs of the two books *Numerical recipes in Fortran 77* and *Numerical recipes in Fortran90*. There is also a Windows help file *NR9F206H.HLP*.

- The directory `nr_chs` contains copies of the routines that are already ported to the standard of this library. To use them: copy `mo_kind.f90`, `mo_constant.f90`, `mo_nrutil.f90`, and `mo_nr.f90` to your code. Also copy the routines needed from the `nr_chs` directory, e.g. `gamma.f90`. Then include in your code for the gamma-example:
USE mo_nr, only: gamma
- To add a new subroutine that is not yet in `nr_chs`, do the following:
 - Copy routine from `nr_ori` to `nr_chs` and move it from `nr_ori` to `nr_ori/in_nr_chs`.
 - In new routine
 - a. Change `nrtype` to `mo_kind`.
 - b. Change `nrutil` to `mo_nrutil`.
 - c. Change `nr` to `mo_nr`.
 - d. Change `I?B` to `I?`.
 - e. Add `USE mo_constants` if constants such as `PI` are used.
 - f. Make `sp` and `dp` versions.
 - Look for the interface in `mo_nr.f90` and change it accordingly, for example add the `dp` version.
 - If you change input/output or similar of the routine, the original documentation in the PDF files is not valid anymore.
Please add the documentation structure from `mo_template.f90` or consider making a completely new module (example `mo_sort.f90`).
- Be aware that the code were acquired under *Numerical Recipes Personal Single-User License*. This license lets you personally use Numerical Recipes code ("the code") on any number of computers, but only one computer at a time. You are not permitted to allow anyone else to access or use the code. You may, under this license, transfer precompiled, executable applications incorporating the code to other, unlicensed, persons, providing that (i) the application is noncommercial (i.e., does not involve the selling or licensing of the application for a fee), and (ii) the application was first developed, compiled, and successfully run by you, and (iii) the code is bound into the application in such a manner that it cannot be accessed as individual routines and cannot practicably be unbound and used in other programs. That is, under this license, your application user must not be able to use Numerical Recipes code as part of a program library or "mix and match" workbench.

How to checkout the complete FORTRAN lib?

To checkout the library in a local directory also called `FORTRAN_chs_lib`:

```
svn checkout https://svn.ufz.de/svn/chs-svn/FORTRAN_chs_lib/
```

To checkout into a local folder with the local name "local_name", which will be created if it does not exist yet:

```
svn checkout https://svn.ufz.de/svn/chs-svn/FORTRAN_chs_lib/ local_name/
```

This will check out the whole library with modules and test folders. To checkout only the module files:

```
svn checkout --depth=files https://svn.ufz.de/svn/chs-svn/FORTRAN_chs_lib/
```

How to contribute to the FORTRAN lib?

As we discussed and agreed with the Fortran programmers at our internal meeting, there are certain **programming rules** we all have to follow to share modules. A template which follows all the rules you can find under `mo_template.f90`. Mainly the rules are:

1. Use the `mo_kind.f90` to declare your number precisions.

```
EXAMPLE:
use mo_kind, only : i4, dp
```

```
integer(i4) :: a      ! To declare a 32-bit integer variable
real(dp)    :: b      ! To declare a double precision real variable
```

2. All subroutines you contribute have to have a single and a double precision version which are combined in an INTERFACE.

EXAMPLE:

```
INTERFACE foo
  MODULE PROCEDURE foo_SP, &      ! Single precision subroutine
                           foo_DP  ! Double precision subroutine
END INTERFACE foo
```

3. Just the interface routines are public, all others are private.

EXAMPLE:

```
PRIVATE
PUBLIC :: foo ! 1-line description of foo here
PUBLIC :: bar ! 1-line description of bar here

INTERFACE foo
  MODULE PROCEDURE foo_SP, foo_DP
END INTERFACE foo
INTERFACE bar
  MODULE PROCEDURE bar_SP, bar_DP
END INTERFACE bar
```

4. If you provide a function/ subroutine based on matrices you should allow for an optional MASK argument, if possible.

EXAMPLE:

```
FUNCTION Mean_dp(A,MASK)
  ! Calculate average
  real(dp), dimension(:), intent(in)      :: A
  logical,  dimension(:), intent(in), optional :: MASK
  real(dp), dimension(size(A)) :: Mean_dp
  ...
  Mean_dp = sum(A, mask=MASK) / count(MASK)
  ...
END SUBROUTINE
```

5. Documentation/ Comment the source code
 - a. Give 1-line descriptions after the public definition.
 - b. Documentation:
 - document before the individual routines;
 - do one documentation per interface, i.e. no separate docu. for sp and dp;
 - follow the documentation structure before the function mean_sp in mo_template.f90;
 - break comment lines at column 130 at most.
 - c. Each module has to appear in the **two tables** above. Please include the module name and its one-line description only if the module is fully tested and documented.
6. Coding style
 - a. Sort routines alphabetically in the file and in the public definitions.
 - b. Break lines at column 130 at most.
 - c. Do not use tabs in files.
 - d. Each module should have a test directory : **test_mo_xxx**, which includes a test program. Include a check that states if the test worked or not.
In the subdirectory test_mo_xxx you simply link your modules, i.e. you do on the command prompt in the test directory:
test_mo_xxx> ln -s ../mo_kind.f90
test_mo_xxx> ln -s ../mo_xxx.f90
 - e. The module should be tested with at least two different compilers (of different vendors),

- e.g. ifort and gfortran
- f. Use lowercase characters for the module name.

[Goto MainPage](#)