1. Compiling RTTOV-10

1.1. RTTOV source layout

The RTTOV source code is split in separate directories; if the user wishes to add some new code which does not implement some basic radiative transfer calculations, but instead depends on the existing code, the user shall consider the possibility of creating a new directory where he will add his code.

The code in each subdirectory will be compiled and packed into a library; that is, there are as many libraries as there are subdirectories. For instance, subdirectory `parallel` will yield `librttov10.0_parallel.a`.

Please note that no circular dependency between subdirectories is allowed; that is, for instance, given two subdirectories dir1 and dir2, it is forbidden for the code in dir1 to make calls to routines in dir2 while dir2 does the same from dir1.

1.2. Creating a Makefile

The creation of a set of Makefiles for compiling RTTOV is automated. The Makefiles are created by a script which analyses the dependencies between RTTOV source units. These dependencies are of two kinds:

- Module dependency: file `a.F90` contains a `use b` statement. Hence if module b is recompiled then unit `a.F90` has to be recompiled.
- Interface dependency: file `a.F90` includes `b.interface`. If the interface of `b` changes then it is necessary to recompile `a.F90`.

To create the Makefiles, change to the subdirectory `src` of the RTTOV distribution and type:

```
$ ../build/Makefile.PL
```

The script above will create a top level Makefile and a Makefile in each subdirectory.

Note that this script has to be run every time dependencies change. Adding a new subroutine or a new program in RTTOV `src/` directory implies running the script again, and so does adding a use statement or including an interface. Not updating the Makefiles may lead to the creation of spurious executable code.

1.3. Compiling for a specific architecture

Compiling RTTOV requires identifying the target machine; the directory `build/arch` contains a list of architectures which have been tested by the project team. The user shall choose the one which appears the most appropriate for his machine.

The user shall then change to the `src` directory of his RTTOV distribution, and type:

```
$ make [ARCH=myarch]
```

The architecture can be specified in the ARCH environment variable or via the ARCH parameter. If neither of these are defined, the core code will be built using gfortran. Object files will be kept in the `obj/` subdirectory of the RTTOV distribution, modules files in `mod/`, executables in `bin/`, and interfaces in `include/`. Note that the creation of these interface files are part of the building process (see section « Interface files »).

It is possible to specify a target directory to install RTTOV; this is very useful when compiling RTTOV with different flags or a different compiler on the same machine:

```
$ make ARCH=myarch INSTALLDIR=mydir
```

The command above, when issued from the `src/` directory will install the `obj/`, `bin/`, `include/` and `mod/` directories in the `mydir/` directory in the RTTOV distribution. Note that the following restriction currently holds: the `mydir` directory has to be located in the RTTOV distribution main directory; but once it is compiled and tested, the user is free to move it where he likes.

When RTTOV is compiled for a specific architecture, a `tmp-myarch` is created in the RTTOV top directory; this makes possible to compile RTTOV in parallel (for different architectures) and to keep the listings issued by some compilers.

Note that by default only the core RTTOV code is compiled. To build the full code, including the MW scattering and emissivity atlas routines, the `all` target should be specified (but see section 1.5 below regarding the emissivity atlas).

Some other targets exist in the Makefile:

- `clean` : removes all object files, libraries, executables, module files and interfaces created by the Makefile.
- `dist` : typing « `make ARCH=myarch dist` » will create a gzipped tarball of RTTOV source and test definition directories.
- `distlbl` : typing « `make ARCH=myarch distlbl` » will create a gzipped tarball of RTTOV coefficient generation source and definition directories (note the LBL code is not supplied in the standard RTTOV-10 package).

To make use of the parallel routines, RTTOV-10 must be compiled with OpenMP. This is typically a case of supplying a suitable flag to an appropriate compiler. There is a compiler flag file in `build/arch/` for compiling with OpenMP support with gfortran.

1.4. Partial build of RTTOV

It is possible to build only the parts of the code the user is interested in. For instance:

```
$ make ARCH=g95 INSTALLDIR=install/g95 mw_scatt
```

will build only the code in the `scatt` directory and its dependencies, namely the `main` code. Other targets exist:

```
$ make ARCH=g95 INSTALLDIR=install/g95 mw_scatt/lib
$ make ARCH=g95 INSTALLDIR=install/g95 mw_scatt/clean
```

1.5. Using external libraries

It is possible to use external libraries (NetCDF v3.6 or later is required for building the emissivity atlas code, and HDF5 v1.8.3 or later is required for the coefficient generation software); in this case, it is necessary to modify the file `build/Makefile.local` which contains the flags required by these libraries. This file contains sample lines for NetCDF and for DrHook (note that in the case of DrHook, the dummy file `src/main/yomhook.F90` must be removed).

For example, to build with the NetCDF library `Makefile.local` should contain the following lines:

```
NETCDF_PREFIX  = path-to-netcdf-install
FFLAGS_NETCDF  = -I$(NETCDF_PREFIX)/include
LDFLAGS_NETCDF = -L$(NETCDF_PREFIX)/lib -lnetcdf
```

```
FFLAGS_EXTERN  = $(FFLAGS_NETCDF)
LDFLAGS_EXTERN = $(LDFLAGS_NETCDF)
```

1.6. Interface files

Interface files are created automatically from the source code by the script `build/mkintf.pl`. Given a Fortran unit `a.F90` this script extracts the source code from `a.F90` up to the !INTF marker which shall appear in every Fortran unit which requires an interface (namely subroutines and functions). Hence a Fortran unit which needs an interface to be extracted shall be written as follows:

```
Subroutine a( x1, x2, x3, .... )
! use statements go here
Use m1
Use m2
! argument declarations go here
Real :: x1
Real :: x2
Real :: x3
...
!INTF_END
```

Note the `!INTF_END` mark at the end of arguments declaration. It is also possible to exclude a part of the code the automatically generated interface, using !INTF_ON and !INTF_OFF markers; the typical example is the use statement of a module whose imported entities are not used in the dummy argument declaration:

```
Subroutine b(x, y)
!INTF_OFF
use m1, Only : t1, t2
!INTF_ON
real :: t1, t2
!INTF_END
```

It is possible to create manually the interface of `a.F90` by typing:

```
$ ../build/mkintf.pl a.F90 a.interface
```

Note also that modifying `a.F90` does not imply that `a.interface` will be created again. It will actually be created only if it different from the one which already exists; this is to avoid unnecessary recompilation of the code.

1.7. Creating an architecture configuration file

If the user architecture is not included in the `build/arch` directory bundled with RTTOV (or in case the user would like to customize the installation of RTTOV), it is possible to create a new configuration file.

This configuration file shall be installed in the `build/arch` directory and define the following macros:

- `FC` : the name of the user's Fortran 90 compiler.
- `FC77` : the name of the user's Fortran 77 compiler; this might be the Fortran 90 compiler

with possibly some special options.

- `LDFLAGS_ARCH` : specific flags to pass to the linker.
- `FFLAGS_ARCH` : specific flags for the Fortran compiler.
- `AR` : the command to create a library from object files.

This configuration file may define the following macros:

- `FFLAG_MOD` : this is the flag used by the Fortran 90 compiler to locate module files; it defaults to `-I`, but it is possible to override this setting.
- `CPP` : the name of the pre-processor; defaults to `cpp`.
- Specific flags for some RTTOV units; defining `FFLAGS_ARCH_a` will force the build system to compile unit `a.F90` with these specific flags.

We reproduce below the content of the configuration file for the NEC-SX F90 compiler with optimization:

```
FC=sxf90
FC77=sxf90
LDFLAGS_ARCH=

FFLAGS_HOPT= -Chopt
FFLAGS_SAFE= -Cvsafe
FFLAGS_NEC = -Wf,-pvctl loopcnt=200000 -Wf,-pvctl nomsg -Wf,-O nomove,-O nomsg
-DRTTOV_ARCH_VECTOR

FFLAGS_ARCH= $(FFLAGS_HOPT) $(FFLAGS_NEC)
FFLAGS_ARCH_rttov_alloc_prof      = $(FFLAGS_SAFE) $(FFLAGS_NEC)
FFLAGS_ARCH_rttov_alloc_predictor = $(FFLAGS_SAFE) $(FFLAGS_NEC)
FFLAGS_ARCH_rttov_tl              = $(FFLAGS_SAFE) $(FFLAGS_NEC)
FFLAGS_ARCH_rttov_ad              = $(FFLAGS_SAFE) $(FFLAGS_NEC)
AR=sxar rv
```

The previous configuration file shows that the Fortran 90 compiler on this platform is sxf90, the archive creation command is sxar rv, and that some files require that optimization be disabled (namely `rttov_alloc_prof.F90`, `rttov_alloc_predictor.F90`, `rttov_tl.F90`, `rttov_ad.F90`).