RTTOV v14 Quick Start Guide

James Hocking

Met Office

This documentation was developed within the context of the EUMETSAT Satellite Application Facility on Numerical Weather Prediction (NWP SAF), under the Cooperation Agreement dated 7 September 2021, between EUMETSAT and the Met Office, UK, by one or more partners within the NWP SAF. The partners in the NWP SAF are the Met Office, ECMWF, DWD and Météo France.

Change record			
Version	Date	Author / changed by	Remarks
0.1	01/03/24	JH	First draft for v14 beta.
1.0	21/08/24	JH	Updates after beta.
1.0.1	18/10/24	JH	Updates after Met Office review
1.0.2	28/11/24	JH	Updates after DRR

Table of Contents

1. *Introduction*

This guide provides a summary of how to get started with RTTOV for new users. It assumes a basic knowledge of Fortran 90. Sections 2 and 3 cover compilation of the package and testing your installation. Section 4 gives an overview of basic RTTOV concepts as it guides you through a simple example of a clear-sky simulation. You should refer to the user guide for full details of all RTTOV capabilities. If you have problems or questions when compiling or running RTTOV please contact the helpdesk:

https://nwp-saf.eumetsat.int/site/help-desk/

You can experiment with RTTOV simulations and 1D-Var retrievals directly online with a Web Based Satellite Sounding Training Application https://sounding.trainhub.eumetsat.int/. A selection of satellite instruments and input profiles from the NWP SAF profile database are used for the simulation and the retrieval tools. You can modify the temperature, humidity, observation and background errors, noise, satellite zenith angle, or surface emissivity, and see how it impacts simulations and retrievals. The tool can display the results of the Jacobian model as well as the result of the direct model (brightness temperatures and radiances), and the differences between the different runs.

2. *Compilation*

First create a directory (e.g., *rttov14/*), navigate into it, and extract the RTTOV tarball:

```
$ tar xvf rttov140.tar.xz
```

Some features require RTTOV to be compiled against the netCDF4 library: these are reading of netCDF coefficient files (the hyperspectral IR sounder coefficient files and other large files are distributed in netCDF format for efficiency), and the land surface emissivity/BRDF atlases. It is recommended to compile against the netCDF library, but it is not mandatory and it is not required in order to follow this quick start guide. To compile with netCDF you must first edit the file *build/Makefile.local* **before** running the *rttov_compile.sh* script (described below): specify the path of your netCDF installation in the *NETCDF_PREFIX* variable, and then uncomment one *FFLAGS_NETCDF* line and one *LDFLAGS_NETCDF* line. You may also need to specify the HDF5 library that your netCDF library was compiled with: see the comments in *Makefile.local*. Full details on building RTTOV are in the RTTOV user guide.

You can compile RTTOV by running the interactive script *build/rttov_compile.sh* from within the *src/* directory:

```
$ cd src
$ ../build/rttov_compile.sh
```

The script asks which compiler flags you wish to use: it prints out a list of the available options. The default is to use gfortran-openmp. If there is an “-openmp” option for your compiler (as in “gfortran-openmp”) this can be used to enable multi-threaded simulations via RTTOV's own “parallel” interface. The “-debug” options are not recommended.

The script then asks for an installation directory: leave this blank to use the default which is your top-level RTTOV directory. (Note: if you do specify a directory, it is a path relative to your top-level RTTOV directory rather than an absolute path).

The script will test whether you have f2py installed and if so will ask whether you want to compile the Python wrapper and the GUI: choose either “y” or “n” (it does not matter which for this guide).

The script asks if you want to specify additional flags: leave this blank.

Finally the script prints a summary of your selected options: enter “y” and return to compile RTTOV.

After compilation the RTTOV executables and libraries can be found in the *bin/* and *lib/* directories in the top-level RTTOV directory respectively (or, if you specified one, within your chosen installation directory).

3. Verifying your installation

If you compiled against the netCDF library, then before running RTTOV you may need to specify the location(s) of the netCDF and/or HDF5 libraries in your $LD_LIBRARY_PATH. For example:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/hdf5/lib
```

Change to the *rttov_test/* directory in the top-level RTTOV directory. You can then verify your installation by running the following script:

```
$ cd ../rttov_test
$ ./test_rttov14.sh ARCH=myarch
```

You should replace *myarch* with whatever compilation flags you chose in answer to the first question from the compilation script (e.g. *gfortran-openmp*).

This script runs RTTOV simulations for various sensors and compares the output against supplied reference data. The output is printed in a table that provides a lot of information about how each test is configured. More information about the test suite itself is given in the test suite guide (*docs/rttov-test.pdf*) although it is not necessary to read that document to use RTTOV. Successful tests are reported as “OK” in green in the right-most column. For example, successful output looks like this:

```
      TEST_ID         C      P  S    L EBRAHG         GASES         PORTSCAPI BRPRLTQLFSNMIS    I    O T    M    N CPTAFIDTAKBTAT  REAL TIME   USER TIME STATUS
     amsre/301        72     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.48        0.16   OK
     amsua/301        90     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.42        0.22   OK
   amsua/301clw       90     6 1  70 X.X..2 clw                     20.31X..4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.52        0.21   OK
     amsub/301        30     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.24        0.09   OK
       msu/301        24     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.28        0.09   OK
     ssmis/301       126     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.43        0.24   OK
    windsat/301       96     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.41        0.21   OK
      hirs/517       114     6 1  70 X.X..2 co2 o3                  20.31XX.4 XX.XXXX.X..321    1    1 0    1    1 .....XXXXX....       0.48        0.27   OK
     modis/401       216     6 1  70 XXX..2 o3                      20.31X..4 XX.XXXX.XX.321    1    1 0    1    1 .....XXXXX....       0.84        0.55   OK
    seviri/504        66     6 1  70 XXX..2 co2 o3                  20.31X..4 XX.XXXX.XX.321    1    1 0    1    1 .....XXXXX....       0.46        0.25   OK
```

If there are differences to the reference data, these are indicated by a “DIFF” in yellow in the right-most column. This can happen in some cases for adjoint or, especially, the Jacobian model outputs (indicated by references to “PROFILES_AD” and “PROFILES_K” respectively). The test suite outputs the test values and the reference values where they differ. It is not unusual to see occasional small compiler-related numerical differences. Acceptable differences for a test look something like this, for example:

```
    windsat/301       96     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 ......XXXX....       0.31        0.15  DIFF

                           | ========== PROFILES_K(  77)%SKIN( 1)%FASTEM ==========
                           |                                                                  4
                           | test_rttov14.1.gfortran-openmp/windsat/301/out/k/profiles_k.txt: -0.158910E-013
                           | test_rttov14.2/windsat/301/out/k/profiles_k.txt : 0.00000
                           |  ========== PROFILES_K(  78)%SKIN( 1)%FASTEM ==========
                           |                                                                  4
                           | test_rttov14.1.gfortran-openmp/windsat/301/out/k/profiles_k.txt: 0.158910E-013
                           | test_rttov14.2/windsat/301/out/k/profiles_k.txt : 0.00000
                           | ========== PROFILES_AD(   5)%SKIN( 1)%FASTEM ==========
                           |                                                                  4
                           | test_rttov14.1.gfortran-openmp/windsat/301/out/ad/profiles_ad.txt: 0.394430E-030
                           | test_rttov14.2/windsat/301/out/ad/profiles_ad.txt : 0.00000
                      +-----------------------------------------------------------------------------------------
```

Failed tests are indicated by a red “FAIL” in the right-most column. In this case an error message is usually printed which may help to diagnose the problem. This is an example of a failed test, in this case because the coefficient file was not found:

```
    windsat/301       96     6 1  70 X.X..2                         20.31X..4 XX.XXXX.X..321    1    1 0    1    1 ......XXXX....                          FAIL

                           | STOP 1
                           |  2024/02/02  11:09:04  fatal error in module
                           | ../../src/coef_io/rttov_coef_io_mod.F90:0220
                           |        File of filetype rtcoef does not exist:
                           | ~/rttov/rtcoef_rttov14/rttov13pred54L/rtcoef_coriolis_1_windsat.dat
                           |  2024/02/02  11:09:04  fatal error in module
                           | ../../src/coef_io/rttov_read_coefs.F90:0273
                           |
                           |  2024/02/02  11:09:04  fatal error in module ../../src/test/rttov_test.F90:1256
                           |
                           |  2024/02/02  11:09:04  fatal error in module ../../src/test/rttov_test.F90:0284

                      +-----------------------------------------------------------------------------------------
```

4. *RTTOV basics and the example code*

Example Fortran code for running the RTTOV forward model is provided in *src/test/example_fwd.F90*. This is a basic program intended to demonstrate how RTTOV is run. It reads data for one or more profiles from an ASCII file, calls RTTOV once, and then writes various outputs to an ASCII file. The intention is that this code can form the basis of your own application. The code contains many comments to help you understand what it does. The program is compiled to an executable in the *bin/* directory when you build RTTOV.

All file paths in this section are relative to the *rttov_test/* directory. You can run the example from within the *rttov_test/* directory using the *run_example_fwd.sh* shell script:

```
$ ./run_example_fwd.sh ARCH=myarch
```

As before you should set “myarch” to the compiler flags you specified when building RTTOV (e.g., *gfortran-openmp*). The input profile data are specified in the ASCII file *test_example.1/prof.dat*: this file also contains comments to describe the contents. There is a section at the top of the *run_example_fwd.sh* shell script which allows you to configure aspects of the simulation, in particular: the coefficient file of the sensor to simulate, the channel list to simulate, and whether to include solar radiation. You must also specify the number of profiles defined in *prof.dat* and the number of pressure half-levels (*nlevels*). By default the script compares the output to provided reference data: if you modify the example code or profile data you can turn off this checking by setting *CHECK_REF=0* in the shell script.

The following sections highlight the important features of the *src/test/example_fwd.F90* program. The Annexes of the user guide provide full details of the interfaces to the various subroutines that are used.

4.1 Variable declarations

RTTOV defines Fortran kinds for real, integer and logical variables in the module *rttov_kinds:* you should import the variables *jprv* (real kind), *jpim* (integer kind), and *jplm* (logical kind) from this module. All variables input to RTTOV should use these kinds where relevant.

The program imports various Fortran derived types from the *rttov_types* module and it declares variables of each type. These are used for passing data in and out of RTTOV. In particular note the following:

Type	***Variable***	***Description***
rttov_options	*opts*	options to configure the simulations
rttov_coefs	*coefs*	structure to hold data from coefficient file
rttov_chanprof	*chanprof(:)*	specify channels/profiles to simulate
rttov_emis_refl	*emis_refl(:)*	input/output emissivities and reflectances
rttov_profile	*profiles(:)*	input atmospheric and surface data
rttov_radiance	*radiance*	output radiances
rttov_transmission	*transmission*	output transmittances

4.2 Specify options

All aspects of an RTTOV simulation can be configured by setting the various options defined in the *rttov_options* derived type. The options are grouped in sub-types according to function. There are many different options and the full list is given in Annex J of the user guide. Some examples that we will encounter in this guide are:

opts%rt_all%solar	activate solar radiation in the simulations
opts%rt_all%o3_data	activate ozone as an input trace gas

It is important to specify the RTTOV options first as the values of some options will define what members are available in RTTOV structures (derived types) that are subsequently allocated.

RTTOV fails with a fatal error if you try to specify mutually incompatible options or if the options are not compatible with the loaded coefficient file (see next section).

In the example code the *o3_data* option is set to false. If you wish to supply ozone profiles you would need to set this to true, modify the section of code which reads profile data to ingest ozone data (see section 4.6 below) and provide ozone data in the input data file. You must also make sure the selected coefficient file supports variable ozone (see next section).

4.3 Read the coefficient file

The RTTOV gas absorption optical depth calculation is a parameterisation that requires pre-computed regression coefficients. These coefficients are specific to each instrument and are stored in coefficient files beginning with "*rtcoef_*". The coefficient files can be found in the sub-directories within the *rtcoef_rttov14/* directory in the top-level RTTOV directory.

The RTTOV package contains optical depth coefficient files for a selection of common sensors. RTTOV supports a large number of sensors and all RTTOV coefficients including the large hyperspectral sounder coefficient files are available from the web site:

https://nwp-saf.eumetsat.int/site/software/rttov/download/coefficients/coefficient-download/

If coefficients are not available for the sensor you wish to simulate then you can request a new coefficient file via the helpdesk (see section 1).

There are various types of gas optical depth coefficient file: the user guide gives details (section 3). In this guide we will focus on the coefficient files contained in the *rtcoef_rttov14/rttov13pred54L/* directory.

The "54L" refers to the fact that RTTOV calculates the gas optical depths on a fixed set of layers defined by 54 pressure levels (see below). Coefficients are available on other sets of levels (in particular 101 levels) for certain sensors. The "13pred" refers the optical depth parameterisation: the files in this directory are based on the parameterisation first implemented in RTTOV v13.

The coefficients are read in your code by calling the *rttov_read_coefs* subroutine. The example code demonstrates a simple call to this subroutine which reads all channels from the file specified by it's full path. Annex C of the user guide gives the full interface to this subroutine. In particular it is possible read only a subset of channels from the file, but this will not be described here.

4.4 Allocate input/output arrays and structures

RTTOV provides the *rttov_alloc_direct* subroutine which can be used to allocate (and later deallocate) any/all input/output arrays and structures for the RTTOV direct model. The example code sets the flag to initialise the newly allocated data structures with default values (typically zeros). See Annex D of the user guide for the full interface to this subroutine.

4.5 Specify profiles and channels to simulate

As section 4.6 below describes, you can pass atmospheric and surface data for one or more profiles into RTTOV. RTTOV allows you to simulate an arbitrary subset of sensor channels for each profile you pass. Usually you will want to simulate the same set of the channels for all profiles. The profiles and channels to simulate are specified in the *rttov_chanprof* structure. You must declare an array of this type with size equal to the total number of channels you are simulating over all profiles. If you are simulating *nchannels* channels for each of *nprofiles* profiles, then the size of the *chanprof(:)* array is *nchanprof = nchannels * nprofiles*.

The members of this structure are *chanprof(:)%prof* and *chanprof(:)%chan* which specify each profile/channel combination to be simulated. Suppose you are simulating channels 1-3 of a sensor for 2 profiles. Then you would populate *chanprof(:)* as follows:

chanprof(1:6)%prof = (/ 1, 1, 1, 2, 2, 2 /)

chanprof(1:6)%chan = (/ 1, 2, 3, 1, 2, 3 /)

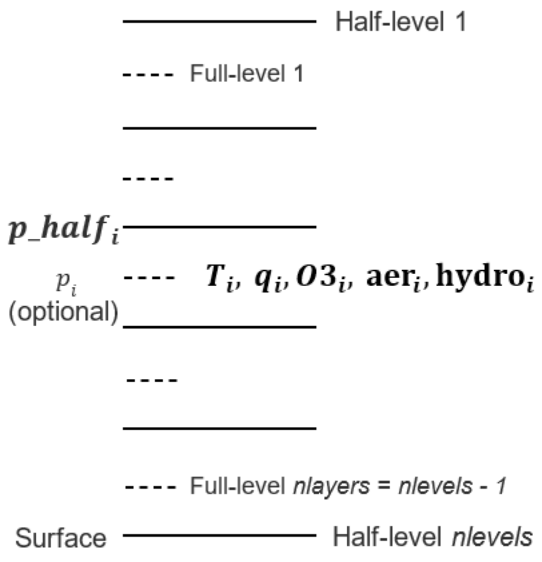
The profiles must be specified in ascending order with all channels to simulate for profile 1 specified first, followed by all channels for profile 2, and so on. The profile numbers are simply the indices into the *profiles(:)* array that you declare to input your profile data to RTTOV (see section 4.6).

The channel numbers are defined by the optical depth coefficient file. Channel numbering in RTTOV *always* starts at 1. You should check the headers of the coefficient file to determine the ordering of the channels. RTTOV generally adheres to each instrument's convention.

The example code populates the *chanprof(:)* array such that the sensor channels specified in the *run_example_fwd.sh* shell script are simulated for each profile passed in.

4.6 Input atmospheric and surface variables

RTTOV is a one-dimensional radiative transfer model: it simulates satellite-seen radiances given a vertical profile of atmospheric temperature and water vapour specified for an arbitrary set of pressure levels. The input atmospheric and surface variables are provided to RTTOV in the *rttov_profile* derived type. The following figure copied from the user guide illustrates the input profile.



You must specify *nlevels* which is the number of pressure half-levels (*p_half*). Adjacent pressure half-levels provide the upper/lower boundaries for *nlayers=nlevels-1* layers. Each layer is associated with a pressure full-level (*p*). The terms “layers” and “full-levels” refer to the same thing in the RTTOV documentation. Similarly “levels” and “half-levels” are used interchangeably. Temperature (*T*) and water vapour concentration (*q*) are provided for each layer (or, equivalently, for each full-level). The pressure half-levels, temperature, and water vapour are mandatory inputs. The pressure full-level values are optional and are not discussed further in this quick start guide. Other mandatory inputs include the surface type, and the selection of units for gas inputs. The full *rttov_profile* structure is given in Annex J of the user guide, and table 7.2 of the user guide gives details about which variables are required under different conditions.

You must declare an array of type *rttov_profile*, say *profiles(:)*. It is possible to pass one or more profiles into RTTOV: RTTOV simulates radiances for the sensor channels you choose for each profile in *profiles(:)* as described in section 4.5.

The pressure half-levels passed into RTTOV will be different for each profile, but the number of pressure half-levels (*nlevels*) must be the same for all profiles passed into RTTOV in a single call. For accurate simulations it is essential that the pressure half-levels encompass the weighting functions of the channels being simulated (i.e., that they cover the full range of atmosphere to which the channels are sensitive). The top-most pressure half level may be at 0 hPa. You should pass profiles from NWP models on their native vertical resolution.

RTTOV always assumes that the surface lies on the bottom pressure half-level. This is consistent with the way that NWP models represent the atmosphere. RTTOV provides the ability to represent multiple surface types falling within the satellite field of view. You must specify the number of surface types (*nsurfaces*) to be associated with each profile. In this quick start guide, we will only consider homogenous surfaces, i.e., where *nsurfaces=1*. As for *nlevels*, *nsurfaces* must be the same for all profiles passed into RTTOV in a single call. The surface skin and near-surface variables in the profile structure (see below) are specified in the *profiles%skin(:)* and *profiles%near_surface(:)* sub-types which are arrays of size *nsurfaces*.

For UV/visible/IR sensors and some MW sensors, RTTOV allows a number of other optional trace gas profiles to be specified such as ozone. The variable gases available depend on the coefficient file (see section 3 of the user guide). If you wish to supply ozone profiles to RTTOV, for example, you must set the *opts %rt_all%o3_data* option to true. If you leave the option as false RTTOV uses a fixed climatological ozone profile.

For clear-sky simulations the following variables must always be specified for each profile:

profiles(:)%p_half(1:nlevels)	pressure half-levels (hPa)
profiles(:)%t(1:nlayers)	temperature on each layer (K)
profiles(:)%q(1:nlayers)	water vapour concentration on each layer (units: see below)
profiles(:)%skin(1)%t	surface skin temperature (K)
profiles(:)%skin(1)%surftype	surface type: *surftype_land* (0), *surftype_sea* (1), *surftype_seaice* (2)
profiles(:)%zenangle	local satellite zenith angle at surface (degrees)
profiles(:)%elevation	surface elevation (km)
profiles(:)%latitude	latitude (degrees)

In your own code it is recommended to use the named constants provided in the *rttov_const* module for integer variables that select between options where possible, rather than using the corresponding integer values explicitly. This applies to *surftype* above and to *gas_units* and *watertype* described below among others. All relevant constants are summarised in Annex K of the user guide.

You can choose the units of all input gas profiles (water vapour, ozone, etc) by setting the *profiles(:)*

%gas_units variable. There are three options:

gas_unit_ppmv (2)	=> ppmv over moist air
gas_unit_kg_per_kg (1)	=> kg/kg over moist air (the default)
gas_unit_ppmvdry (0)	=> ppmv over dry air

Other important profile variables include:

profiles(:)%azangle	local satellite azimuth angle at surface (degrees)
profiles(:)%sunzenangle	local solar zenith angle at surface (degrees)
profiles(:)%sunazangle	local solar azimuth angle at surface (degrees)
profiles(:)%o3(1:nlayers)	ozone concentration on each layer (same units as *q*)
profiles(:)%skin(1)%watertype	for *surftype*=sea: *watertype_fresh_water* (0), *watertype_ocean_water* (1)
profiles(:)%skin(1)%salinity	for *surftype*=sea (PSU), *typical value for sea water: 35*
profiles(:)%near_surface(1)%t2m	near-surface (2m) temperature (K)
profiles(:)%near_surface(1)%q2m	near-surface (2m) water vapour (same units as *q*)
profiles(:)%near_surface(1)%wind_u / v10m	10m u and v wind components (m/s)
profiles(:)%near_surface(1)%wind_fetch	wind fetch (m), *typical value: 100000m*

The user guide details the types of simulations or conditions in which each profile variable is used. When carrying out your own simulations you should check the description of the *rttov_profile* structure in Annex J carefully and specify values for all relevant profile variables.

The example code reads profile data from *example.1/prof.dat* into the *profiles(:)* structure. This ASCII format is intended for demonstration purposes as it is easy for you to read to help in understanding the example code. It is not recommended for real-world applications where you may be simulating many thousands of profiles. Instead you would typically replace this section with code to read the profile data relevant to your simulations directly from whatever file format they are provided in such as netCDF, for example.

4.7 Surface emissivity and BRDF

RTTOV can provide surface emissivities (for IR and MW channels) and BRDFs (bi-directional reflectance distribution functions, for solar-affected channels). The user guide provides details of the models and assumptions RTTOV makes in this case. Alternatively you can input surface emissivity and/or BRDF values for use in the simulation. This is done via the *rttov_emis_refl* structure. You must allocate an array of this type, say *emis_refl(:),* of size *nsurfaces* (1 in this example).

You specify whether RTTOV should supply the emissivity or BRDF for each channel individually in the logical *emis_refl(1)%calc_emis(:)* and *emis_refl(1)%calc_brdf(:)* arrays which are of size *nchanprof.* These correspond to the channels/profiles specified in the *chanprof(:)* structure (see section 4.5). For those channels where *calc_emis(:)* is true RTTOV will provide an emissivity value, and where *calc_emis(:)* is false you should supply an emissivity value in the corresponding elements of *emis_refl(1)%emis_in(:)*. Likewise for *calc_brdf(:)* and BRDFs for simulations with solar radiation, with input values provided in *emis_refl(1)%brdf_in(:)*.

RTTOV provides a number of emissivity atlases and a BRDF atlas which can be used to provide land surface emissivity and BRDF values. These are not described here: see the user guide.

In the example code we get RTTOV to provide values for the surface emissivity (and BRDF if solar simulations have been activated) by setting *emis_refl(1)%calc_emis(:)* and *emis_refl(1)%calc_brdf(:)* to true for all channels. Note that this is done during the allocation/initialisation of the data structures so no additional code is required here. In this case it does not matter what we specify in the *emis_refl(1)%emis_in(:)* and *emis_refl(1)%brdf_in* arrays.

4.8 Call the RTTOV direct model

This is achieved with a single call to *rttov_direct*. If you compiled RTTOV with “-openmp” compiler flags and you have a CPU with multiple cores you can take advantage of multi-threaded simulations by calling the *rttov_parallel_direct* subroutine instead. This has the same interface as *rttov_direct*, but with a final optional argument, *nthreads*, which allows you to specify the number of threads to use. It is recommended to make use of this capability if you are running many simulations. See Annex H in the user guide for the full interface to this subroutine.

Once *rttov_direct* returns you should check the *errorstatus*: if it is zero the simulation was successful, but any non-zero value indicates there was an error. This applies to all RTTOV subroutines which return an error code.

4.9 RTTOV outputs

The TOA (top-of-atmosphere) radiance outputs are stored in the *rttov_radiance* structure. RTTOV computes radiances for all channels, brightness temperatures (BTs) for channels with significant thermal component (wavelengths above 3µm), and reflectances for solar-affected channels (wavelengths below 5µm). These can be found in the following members of the *radiance* structure:

radiance%clear(:)	TOA clear-sky radiances (mW/m^2/sr/cm^{-1})
radiance%total(:)	TOA radiances including clouds/hydrometeors (mW/m^2/sr/cm^{-1})
radiance%bt_clear(:)	TOA BTs corresponding to *clear(:)* (K)
radiance%bt(:)	TOA BTs corresponding to *total(:)* (K)
radiance%refl_clear(:)	TOA reflectances corresponding to *clear(:)* (no unit)
radiance%refl(:)	TOA reflectances corresponding to *total(:)* (no unit)

Each array has size *nchanprof*: there is one element per simulated radiance corresponding to the channels/profiles specified in the *chanprof(:)* structure (section 4.5).

This guide is focusing on clear-sky simulations. If clouds/hydrometeors are not activated in the simulations then the “*clear*” and “*total*” radiance outputs are identical.

RTTOV outputs additional radiance and transmittance quantities: these are not described further here, see the user guide section 7.8 and Annex J for more information.

RTTOV outputs the surface emissivities used in the simulation in *emis_refl(1)%emis_out(:)*. Similarly, for solar simulations the BRDFs used are output in *emis_refl(1)%brdf_out(:)*.

In the example code the output is quite verbose: it prints out the options structure and, for each profile, it prints out the contents of the profile structure. It then prints a selection of radiance, emissivity, BRDF, and transmittance outputs. In a real-world application you would want to replace this section with code to store the outputs you need to some suitable file.

4.10 Deallocate data

In order to prevent memory leaks you should ensure all allocated data is deallocated once your RTTOV simulations are complete. This is easily achieved for the RTTOV arrays and structures using the *rttov_alloc_direct* subroutine again: the call is identical to the allocation call above (section 4.4) except for the second argument which is specified so that the memory is deallocated. Similarly, the loaded coefficients structure is deallocated by calling the *rttov_dealloc_coefs* subroutine.

4.11 Running RTTOV for your application

You have seen how to run the RTTOV direct model for clear-sky simulations. The *src/test/* directory contains similar *example_*.F90* code for various simulation types: these are all structured in a similar way and hopefully you can now understand these with reference to the user guide. You can base your own application on the relevant *example_*.F90*. However you would almost certainly want to make some modifications:

- Profile ingest: the ASCII format used here is used to make the example clear. In practice you would want to read the profile data directly from whatever file format they are provided in. It is not recommended to use this ASCII format in real-world applications.
- Output: the output from the example code is very verbose and is in ASCII format. In practice you would want to remove much of the output code and write the simulated data to some suitable binary format (e.g., netCDF), especially if simulating many profiles.
- Simulating many profiles: this example is set up to call RTTOV once with multiple profiles because this is the simplest way to demonstrate running RTTOV. In practice if you have many thousands of profiles to simulate or if your simulations require a lot of memory (e.g. many channels of a hyperspectral sounder, or scattering simulations) you may prefer to modify the code so that the profiles are simulated in smaller batches. In this case you would allocate your structures with some small value of *nprofiles* (say 10 or 50) and then loop over the input data, reading in data for each batch of *nprofiles* profiles and calling RTTOV on them in each loop. You might want to set up large arrays into which you store the outputs you require from each batch as they are calculated so that you can write these outputs to a file in one go once all profile batches have been simulated.

5. Other capabilities

Having run a clear-sky simulation, you may be interested in exploring other capabilities of RTTOV. Some examples are given below (the list is not exhaustive): please consult the user guide for full details of how to run RTTOV (section 7) and for details of specific types of simulations (section 8).

Computing Jacobians

RTTOV comprises tangent linear (TL), adjoint (AD) and full Jacobian (K) models in addition to the direct model. The K model can be used to compute the sensitivity of the top-of-atmosphere radiances to each input variable. This is used, for example, in 1DVar retrievals of atmospheric profiles.

Scattering simulations with aerosols and/or hydrometeors:

RTTOV can simulate aerosol- and/or hydrometeor-affected radiances. RTTOV provides various options for aerosol and hydrometeor optical properties. Alternatively it is possible to pass profiles of optical properties per channel explicitly.

Principal Components simulations for hyperspectral IR sounders:

PC-RTTOV is a Principal Components (PC) based model for hyperspectral sensors: by carrying out normal RTTOV simulations for a few hundred channels PC-RTTOV can compute PC scores for the whole spectrum and then, optionally, reconstruct radiances for any set of channels of the sensor. This is an efficient way of simulating full spectra of hyperspectral sounders.

Calling RTTOV from Python or C++:

A wrapper has been created for RTTOV which allows you to call much RTTOV capability directly from Python or C++ (or C). This could be useful if you are more familiar with one of these languages than Fortran. The wrapper interface is fully documented in *docs/rttov-wrapper.pdf*, but you must also refer to the user guide to understand how to run RTTOV correctly.

RTTOV GUI:

A graphical user interface (GUI) for RTTOV based on *pyrttov* (the RTTOV Python interface) has been created which allows most types of simulation to be run. Input profiles and output radiances and Jacobians are visualised and it provides a clear way to understand the impact of changing RTTOV options or profile variables. The GUI is described in *docs/rttov_gui_v14.pdf*.