Python/C/C++ wrapper

for RTTOV v13

James Hocking, Pascale Roquet, Pascal Brunel

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 December 2016, 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	30/03/2020	J Hocking	First draft for v13 beta.
1.0	18/09/2020	J Hocking	Updates after beta.
1.1	16/10/2020	J Hocking	Updates after DRR
1.2	07/10/2021	J Hocking	Updates for v13.1
1.3	31/10/2022	J Hocking	Updates for v13.2

Table of contents

1. Introduction

An interface has been created for RTTOV which allows RTTOV simulations using the direct and K models to be run from Python3 (tested with v3.9, compatibility is expected with subsequent v3.x releases), C or C++ . It is possible to use this interface to run RTTOV without writing any Fortran code. C++ classes and a Python package have been created which allow you to interact with RTTOV in an object-oriented style rather than calling the wrapper interface subroutines directly.

The intention behind the design of the interface is to provide access to as much RTTOV functionality as possible while keeping the interface simple.

This document explains how to call RTTOV from Python, C and C++. You should read the RTTOV user guide (at least the sections which pertain to the kinds of simulations you wish to carry out) in order to understand how RTTOV works before reading this document: ***this document cannot be understood without reference to the RTTOV user guide.***

Section 2 of this document describes compilation of RTTOV with the wrapper. There are two ways to use the RTTOV wrapper:

1. You can call the interface subroutines directly as described in section 3. Sections 4 and 5 provide additional information specific to Python and C/C++ respectively.
2. ***Recommended method:*** a collection of C++ classes have been created which enable RTTOV to be called using object-oriented-style programming. A similar Python interface is available via the pyrttov package. These classes are described in section 6.

You do not need to read sections 3-5 to understand section 6, but the earlier sections contain information which may be useful.

Section 7 gives some important technical information about the wrapper implementation related to thread-safety and other issues.

Section 8 outlines the current limitations of the wrapper. Finally, the appendices provide some additional information about the Fortran-Python/C/C++ interface and the object-oriented classes.

Currently the wrapper supports calls to rttov_direct and rttov_k for clear-sky and visible, IR and MW scattering calculations optionally including use of the surface emissivity and BRDF atlases.

The main changes in the interface since RTTOV v12.3 have been made to support the per-channel specularity variable and the input/output of diffuse reflectance (both of which are managed via the same argument used to pass emissivity and BRDF values in and out of the interface), the new “clwde_param” profile variable, and, for RTTOV-SCATT, the flexible hydrometeor and hydro_frac inputs and the new radar capability. In addition, the new geometric_height and cloud transmittance outputs are available via the wrapper interface.

RTTOV v13.2 adds support for all new developments in this release, including the effective skin temperature inputs which changes the interface (the dimensions of the emissivity/reflectance input/output arrays). This means user code calling the wrapper must be updated to work with v13.2.

2. Compilation and example code

The wrapper Fortran source code is contained in the src/wrapper/ directory. You can use the wrapper with no external library dependencies (the Python wrapper requires f2py), but to use the emissivity and/or BRDF atlases you must compile RTTOV against the HDF5 library (see the user guide).

The easiest way to compile RTTOV is to edit the file build/Makefile.local to point to your HDF5 installation (if the atlases are required) and then do:

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

This runs an interactive script for compiling RTTOV. If you want to compile RTTOV manually refer to section 5.2 of the user guide for details.

Compiling C/C++ code which calls RTTOV

Example Python, C and C++ code is contained in the wrapper/ directory in the top-level of the RTTOV installation.

In order to call RTTOV from C or C++ code you need to include the src/wrapper/rttov_c_interface.h header file in your code and compile against the RTTOV libraries. For the object-oriented interface you need to include the relevant class definitions. The example code in the top-level wrapper/ directory demonstrates this.

Running Python code which calls RTTOV

Having compiled RTTOV as directed above the lib/ directory will contain the Fortran-Python interface in the file rttov_wrapper_f2py.so. You should ensure this is in your current directory or your $PYTHONPATH.

To call the interface subroutines directly you can import them from this file, for example in Python:

```
> from rttov_wrapper_f2py import rttov_load_inst,   \
                                 rttov_call_direct, \
                                 rttov_drop_all
```

See the examples in the top-level wrapper/ directory which demonstrate calling RTTOV from Python, e.g. example_python.py.

Alternatively you can use the pyrttov package which provides an object-oriented interface to RTTOV.

Example code and source files
The following files can be found in the wrapper/ directory:

interface_example_c.c	Calling interface directly in C
interface_example_cpp.cpp	Calling C++ interface directly
interface_example_rttovscatt_cpp.cpp	Calling RTTOV-SCATT C++ interface
interface_example_python.py	Calling Python interface directly
interface_example_rttovscatt_python.py	Calling RTTOV-SCATT Python interface
pyrttov_example.py	Use of pyrttov Python package for multiple instruments and use of the emissivity/BRDF atlases
pyrttov_visirscatt_example.py	Use of pyrttov for visible/IR scattering simulations where optical properties are input
pyrttov_rttovscatt_example.py	Use of pyrttov for MW scattering simulations
pyrttov_rttovscatt_radar_example.py	Use of pyrttov for MW radar simulations
Rttov_example.cpp	Use of C++ Rttov class for multiple instruments including use of the emissivity/BRDF atlases
Rttov_visirscatt_example.cpp	Use of C++ Rttov class for visible/IR scattering simulations where optical properties are input
RttovScatt_example.cpp	Use of C++ RttovScatt class for MW scattering simulations
RttovScatt_radar_example.cpp	Use of C++ RttovScatt class for MW radar simulations
RttovSafe_example.cpp	Use of C++ RttovSafe class for multiple instruments including use of the emissivity/BRDF atlases
RttovSafe_visirscatt_example.cpp	Use of C++ RttovSafe class for visible/IR scattering simulations where optical properties are input
RttovScattSafe_example.cpp	Use of C++ RttovScattSafe class for MW scattering simulations
RttovScattSafe_radar_example.cpp	Use of C++ RttovScattSafe class for MW radar simulations
Makefile	Makefile to compile all C and C++ examples

These can be used as examples from which to develop your own code. The Makefile demonstrates how to compile C and C++ code which calls RTTOV. In order to compile the examples you should look at the top of the Makefile to see if you need to modify the compilers, compiler flags, or the location of your RTTOV libraries. After editing the Makefile as necessary you can compile the example code in the wrapper/ directory:

```
$ make
```

The following files define the classes used by the C++ object oriented interface to RTTOV (see section 6); again the Makefile demonstrates how to compile code which uses the object oriented interface:

RttovSafe.h/.cpp	Class allowing you to call RTTOV for an instrument – carries out some checks on the profiles to help prevent errors.
Profile.h/.cpp	Class representing a single profile for use with RttovSafe.
Rttov.h/.cpp	Class allowing you to call RTTOV – limited error checking.
Profiles.h/.cpp	Class representing one or more profiles for use with Rttov.
RttovScattSafe.h/.cpp	Class allowing you to call RTTOV-SCATT for an instrument – carries out some checks on the profiles to help prevent errors.
ProfileScatt.h/.cpp	Class representing a single profile for use with RttovScattSafe.
RttovScatt.h/.cpp	Class allowing you to call RTTOV-SCATT – limited error checking.
ProfilesScatt.h/.cpp	Class representing one or more profiles for use with RttovScatt.
Options.h/.cpp	Class representing RTTOV and wrapper options.
Atlas.h/.cpp	Class representing emissivity or BRDF data for a single atlas, month, and (where relevant) instrument.

The Makefile compiles these classes into a library (librttovcppwrapper) which you can link your own code against: the example code is compiled like this.

The C++ source includes Doxygen markup. To generate HTML and RTF documentation you can run the following from within the wrapper/ directory:

```
$ doxygen doxygen_config_wrapper
```

The output can be found in wrapper/doxygen_doc_wrapper/.

The pyrttov Python package provides an object-oriented interface to RTTOV in Python. The package source files are contained in the pyrttov/ directory. The pyrttov_doc/ directory can be used to generate documentation for pyrttov using Sphinx: from within pyrttov_doc/ run

```
$ make html
```

This requires both the pyrttov package and the RTTOV rttov_wrapper_f2py.so library to be in your $PYTHONPATH: the documentation can be found in _build/html/index.html. Section 6 provides more details on the pyrttov package.

3. General description of interface

Note that the recommended way to call the interface is via the classes which are described in section 6. The details of the underlying interface (described in this section) are hidden from the user so the classes are a more user-friendly way of calling RTTOV. Nevertheless this section may be useful to understand more about how the wrapper works. If you wish to call RTTOV from C you must use the interface described in this section.

This section describes the interface in general terms: the Python and C/C++ interfaces are very similar. To understand the wrapper interface itself you should read this and then refer to the following two sections below which contain information specific to Python and C/C++. Appendix B lists all subroutines in the RTTOV wrapper.

The wrapper allows you to load coefficients for one or more instruments simultaneously, set the options associated with each instrument, make calls to the RTTOV direct and K models, and access the resulting data. There are also subroutine calls to load data from the IR and MW emissivity and BRDF atlases, and to obtain emissivity or BRDF data from the loaded atlases.

Each initialised instrument is entirely independent. It is possible to load the same coefficients multiple times, giving you multiple independent instances of one instrument. For example, you could extract a different channel set for each instance if you wanted to simulate the instrument for different purposes. Alternatively you can initialise a collection of different instruments. Each initialised instrument has its own set of RTTOV options associated with it.

Similarly, each set of atlas data is independent and can be used to obtain emissivities or BRDFs for any compatible loaded instrument.

3.1. Loading an instrument

The `rttov_load_inst` subroutine is used to load an instrument. In this call you provide a string containing the coefficient filename(s) to load (the “rtcoef” file and optionally aerosol or cloud IR scattering files or a MW hydrotable file), any RTTOV options you wish to set and some wrapper-specific options. The format of this string is described below along with the wrapper-specific options.

This subroutine returns an ID which is used in subsequent subroutine calls to identify this instrument. If the returned ID is less than or equal to 0 this indicates that an error occurred and the instrument was not initialised. The interface is as follows:

```
rttov_load_inst( &
    inst_id,     &
    opts_str,    &
    nchannels,   &
    channels)
```

Argument	Type	Intent	Description
inst_id	Integer	out	Returned ID for instrument; if <=0 then error occurred (instrument was not initialised)
opts_str	Character string	in	String containing options and coef filenames (see below).
nchannels	Integer	in	Size of channels array (not required in Python).
channels(:)	Integer	in	Channels to read from coefficient files. If set to (0) (i.e. an array of length one containing a zero) all channels will be read from the coefficient file.

Notes:

To initialise the wrapper for multiple instruments you should make one call to `rttov_load_inst` per instrument.

If you specify a channel list in channels(:) then beware that this will impact the channel numbering when you make calls to RTTOV later. See the user guide section 7.4 for more information. In short: if you have extracted *n* channels when reading the coefficient file they will subsequently be referred to as 1,2,...,*n* rather than by their original channel numbers. If all channels from the coefficient file are read in you can specify a subset of channels to simulate when you call RTTOV. Alternatively you can extract just the required channels into a new coefficient file using rttov_conv_coef.exe (see user guide Annex A) and then read all channels from this new file when loading the coefficients. Note that if running RTTOV-SCATT (i.e. if a hydrotable filename has been specified) the wrapper will ignore any channels(:) argument as all channels must be read in (a warning is printed if you supply the channels argument).

Specifying the options string

The options string consists of multiple space-separated key-value pairs. Each key is a character string related to an option and the value is an integer, real or character string depending on the option being set. It is important that there are ***no spaces*** in the option names (keys).

Example options string in Python:

This string sets up directories as if being called from the top-level wrapper/ directory:

```
opts_str = 'file_coef ' \
  '../rtcoef_rttov13/rttov13pred54L/rtcoef_msg_4_seviri_o3.dat ' \
  'opts%interpolation%addinterp 1 ' \
  'opts%rt_all%o3_data 1 '           \
  'opts%rt_ir%addsolar 1 '          \
  'nthreads 4 '
```

NB The space separation between options is important and there must be no spaces in option names or file/path names!

See the example code in the top-level wrapper/ directory for more examples.

RTTOV coefficient files – *rtcoef file mandatory, others optional*

Specify full paths to the RTTOV coefficient file(s):

Key	Value	Description
file_coef	Full path to rtcoef file	Mandatory, path to rtcoef file.
file_scaer	Full path to visible/IR aerosol coef file	For visible/IR aerosol simulations, path to scaer coef file.
file_sccld	Full path to visible/IR cloud coef file	For visible/IR cloudy simulations, path to sccld coef file.
file_mfasis_cld	Full path to MFASIS LUT file	For visible cloudy simulations using MFASIS-LUT.
file_mfasis_nn	Full path to MFASIS-NN file	For visible cloudy simulations using MFASIS-NN.
file_hydrotable	Full path to MW hydrotable	For RTTOV-SCATT simulations, path to hydrotable file.
file_pol	Full path to polarisation look-up table	For RTTOV-SCATT simulations using ARO-scaling polarisation option, full path to sensor-independent ARO-scaling polatisaton look-up table.

RTTOV options - *optional*

Every option available in the RTTOV options structure (see user guide Annex O) can be set in the options string. The key value is given as in the table in Annex O of the user guide. For logical options the value should be 0 or 1 for false/true respectively. The usual RTTOV default values apply (see user guide). Remember: there must be ***no spaces*** in the option names specified in the string. Some examples are given below:

Key	Value	Description
opts%config%verbose	0 or 1	Set RTTOV verbosity flag.
opts%rt_ir%addsolar	0 or 1	Turn solar radiation off/on.
opts%interpolation%interp_mode	Integer 1-5	Set interpolation mode.

RTTOV-SCATT exposes only a subset of RTTOV options: these are also listed in Annex O of the user guide. The RTTOV-SCATT options can be set using keys prefixed with “opts_scatt”, for example: “opts_scatt%config%verbose”, “opts_scatt%fastem_version” and “opts_scatt %lusercfrac”.

Wrapper-specific options - *optional*

Set options that are related specifically to the wrapper:

Key	Value	Description
verbose_wrapper	0 or 1	Set to 1 for more verbose output from the wrapper (default 0, all output suppressed except fatal error messages).
nthreads	Integer	If <=1 RTTOV is called via the standard interface (e.g. rttov_direct), if >1 RTTOV is called via the parallel interface (e.g. rttov_parallel_direct) using the specified number of threads (default 1).
nprofs_per_call	Integer – greater than 0	Sets the number of profiles passed to each call to rttov_direct or rttov_k

		within the wrapper (default 1).
check_opts	0 or 1	If set to 1 the Fortran `rttov_user_options_checkinput` subroutine (see user guide Annex N) is called to help ensure consistency between the selected options and the loaded coefficient file (default 1).
store_trans	0 or 1	Set to 1 to enable access to transmittance outputs from RTTOV calls (default 0).
store_rad	0 or 1	Set to 1 to enable access to radiance outputs from RTTOV direct model calls (default 0).
store_rad2	0 or 1	Set to 1 to enable access to secondary radiance outputs from RTTOV direct model calls (default 0). If this is set to 1 then store_rad will automatically be set to 1 as well.
store_emis_terms	0 or 1	Set to 1 to enable access to the emissivity retrieval outputs from RTTOV-SCATT direct model calls (default 0). Note that this requires the opts_scatt%lradiance option to be set to 1 (true) as well.

Notes:

To take advantage of multi-threaded execution (by setting nthreads > 1) you must compile RTTOV with OpenMP compiler flags (see user guide).

When calling RTTOV through the wrapper (see below) you can pass any number of profiles. The wrapper will then break these down into chunks and the underlying `rttov_direct`/etc subroutines are called for nprofs_per_call at a time until all profiles have been simulated. You may obtain improved performance (especially with multi-threaded execution) by increasing nprofs_per_call above the default of 1, but if you are simulating a very large number of channels you may run out of memory if this is set too high.

The calls to RTTOV include arguments which return the total TOA radiances and the equivalent brightness temperatures or reflectances (depending on channel wavelength). If you require access to additional RTTOV radiance or transmittance outputs you should set the store_trans, store_rad, store_rad2 and/or store_emis_terms options. You can then use the subroutines listed in Annex B to access this information after calling RTTOV. Note that if store_rad2 is set then store_rad will also be set automatically. See the user guide for more information on RTTOV outputs.

If you are performing visible/IR cloud or aerosol scattering simulations with optical properties from coefficient files (“scaer*”, “sccld*”files) you must ensure the addclouds and/or addaerosl RTTOV options and the paths to the required coefficient file(s) are set in the options string when loading the instrument. If you wish to carry out MFASIS simulations you must set the path to the MFASIS LUT/NN file in the options string in addition to the “sccld” cloud property file. For RTTOV-SCATT calls the path to the hydrotable file must be set in the options string. When using the ARO-scaling polarisation option, the pol_mode must be set and the full path to the polarisation look up table.

3.2. Changing RTTOV options

It is possible to modify the options at any time for an instrument which has been initialised by a call to `rttov_load_inst`.

```
rttov_set_options( &
    err,        &
    inst_id,    &
    opts_str)
```

Argument	Type	Intent	Description
err	Integer	out	Return code: non-zero implies error condition.
inst_id	Integer	in	ID of instrument (as returned by rttov_load_inst) whose options should be updated.
opts_str	Character string	in	String containing options to change.

You can change any options in the options structure and any of the wrapper-specific options in this call. Setting the coefficient file names has no effect in a call to `rttov_set_options` and you should not turn on scattering options which require optical properties from coefficient files if the coefficient files were not read in when `rttov_load_inst` was called. Options that were previously set are retained so you only need to specify options you wish to change.

You can also print the RTTOV and wrapper options by calling `rttov_print_options` (this calls the RTTOV `rttov_print_opts` Fortran subroutine, see user guide Annex N):

```
rttov_print_options(err, inst_id)
```

where err is the output return code and inst_id is the input ID for the instrument whose options you wish to print.

3.3. Using the emissivity and/or BRDF atlases

The emissivity and BRDF atlases can be used to obtain land surface and, in some cases, sea-ice and water emissivity and BRDF values that can be passed into the call to RTTOV. More details about the atlases are given in the user guide.

In order to use the emissivity or BRDF atlases they must first be loaded. There are separate subroutine to set up the BRDF, IR emissivity and MW emissivity atlases. Each subroutine returns a wrapper atlas ID which is used in subsequent subroutine calls to identify this atlas data. If the returned ID is less than or equal to 0 this indicates that an error occurred and the atlas was not initialised. The interfaces are as follows:

```
rttov_load_ir_emis_atlas(atlas_wrap_id, path, month, atlas_id, inst_id,
ang_corr)
rttov_load_mw_emis_atlas(atlas_wrap_id, path, month, atlas_id, inst_id)
rttov_load_brdf_atlas(atlas_wrap_id, path, month, atlas_id, inst_id)
```

Argument	Type	Intent	Description
atlas_wrap_id	Integer	out	Returned wrapper ID for atlas data; if <=0 then error occurred (atlas was not initialised)
path	Character string	in	String containing path to atlas data files.
month	Integer	in	Month (1-12) for which to initialise atlas.
atlas_id	Integer	in	ID of atlas to load, set to -1 for default atlas (see the user guide for the valid IR, MW and BRDF atlas IDs).
inst_id	Integer	in	ID of instrument (as returned by rttov_load_inst) of instrument for which to initialise atlas (may be 0: see below).
ang_corr	Integer	in	IR atlas only: set non-zero to include the zenith angle emissivity correction (see user guide for more information).

Notes:

You can call these subroutines as many times as required (subject to memory limitations) to initialise atlas data from different atlases for multiple months and/or instruments.

For the BRDF atlas, only one atlas is available so you can set the atlas_id to -1.

There are three IR emissivity and two MW emissivity atlases available with IDs as follows:

- UW IR emissivity atlas: atlas_id = 1 (default)
- CAMEL 2007 IR emissivity atlas: atlas_id = 2
- CAMEL climatology IR emissivity atlas: atlas_id = 3
- TELSEM2 MW atlas: atlas_id = 1 (default)
- CNRM MW atlas: atlas_id = 2

The IR emissivity and BRDF atlases can be initialised with an inst_id for a loaded instrument: in this case the atlas data will be specific to that instrument and calls to obtain emissivities/BRDFs will be more rapid, but the loaded data must only be used with that instrument. If you supply a negative inst_id the atlas data can be used with any visible/IR instrument.

The TELSEM2 MW atlas can always be used with any MW instrument so the inst_id argument is ignored in this case.

The CNRM MW atlas is always initialised for a specific instrument and so the inst_id for a loaded instrument must always be supplied in this case.

Obtaining emissivity/BRDF values

A single subroutine is provided to return emissivity/BRDF values from the atlas:

```
rttov_get_emisbrdf( &
    err,            &
    atlas_wrap_id,  &
    latitude,       &
    longitude,      &
    surftype,       &
    watertype,      &
    zenangle,       &
    azangle,        &
    sunzenangle,    &
    sunazangle,     &
    snow_fraction,  &
    inst_id,        &
    channel_list,   &
    emisbrdf,       &
    nchannels,      &
    nprofiles)
```

Argument	Type	Intent	Description
err	Integer	out	Return code: non-zero implies error condition.
atlas_wrap_id	Integer	in	ID of atlas data (as returned by one of the atlas loading subroutines described above) to use.
latitude(nprofiles)	Real	in	Latitude for each profile (used by: all atlases).
longitude(nprofiles)	Real	in	Longitude for each profile (used by: all atlases).
surftype(nprofiles)	Integer	in	skin%surftype for each profile (used by: all atlases).
watertype(nprofiles)	Integer	in	skin%watertype for each profile (used by: BRDF atlas).
zenangle(nprofiles)	Real	in	Satellite zenith angle for each profile (used by: BRDF atlas, MW emissivity atlases, IR atlases only if angular correction is applied).
azangle(nprofiles)	Real	in	Satellite azimuth angle for each profile (used by: BRDF atlas).
sunzenangle(nprofiles)	Real	in	Solar zenith angle for each profile (used by: BRDF atlas, IR emissivity atlases if angular correction applied)

sunazangle(nprofiles)	Real	in	Solar azimuth angle for each profile (used by: BRDF atlas).
snow_fraction(nprofiles)	Real	in	skin%snow_fraction for each profile (used by: optionally by IR emissivity atlas).
inst_id	Real	in	ID of loaded instrument for which to obtain emissivities/ BRDFs. Must be compatible with the atlas data.
channel_list(nchannels)	Integer	in	List of channel numbers for which to obtain emissivities/ BRDFs.
emisbrdf(nprofiles,nchannels)	Real	inout	Output emissivities/BRDFs (depending on atlas type) for each channel and for each profile.
nchannels	Integer	in	Number of channels to simulate (not required in Python).
nprofiles	Integer	in	Number of profiles being passed in (not required in Python).

Notes:

This subroutine can be called with suitable atlas data to obtain the emissivity and/or BRDF values for input to calls to RTTOV (see below).

See Annex O and table 10 in the user guide for information about profile variables (the names in the table above relate to the names in the Fortran profile structure). The RTTOV user guide provides more information about the atlases in respect of, for example, how they each treat different surface types and the input data required by each atlas. All arguments must be supplied to the interface, but if particular variables are not used by the specified atlas the arrays can just be initialised with zeros.

The array index ordering shown above is that which should be used in C/C++: this is opposite to Fortran array index ordering. For Python you should reverse the order of the indices for the 2-dimensional array arguments. It may also be more efficient to ensure that Python stores the arrays in Fortran-contiguous order. See the Python, C and C++ examples which illustrate how to declare the array arguments.

If you extracted a subset of channels from the coefficient file in the rttov_load_inst call for the supplied inst_id then the channel numbers in channel_list(:) are indexes into this list (see user guide section 7.4).

If the specified atlas has no data for the given location it will return a negative value. You may wish to check the output of this subroutine call for negative values and use a different source of emissivity in those cases. However you can pass negative values into RTTOV (see below) and RTTOV will provide surface emissivity/BRDF values for those channels in the simulations.

3.4. Calling the RTTOV direct model

Once a coefficient file has been loaded you can call RTTOV to simulate radiances for an arbitrary number of profiles. Profile data is input via a series of integer and real (float) arrays. The top-of-atmosphere radiances and brightness temperatures (or reflectances) are returned via array arguments. The interface is as follows:

```
rttov_call_direct( &
    err,                &
    inst_id,            &
    channel_list,       &
    datetimes,          &
    angles,             &
    surfgeom,           &
    surftype,           &
    skin,               &
    s2m,                &
    simplecloud,        &
    clwscheme,          &
    icecloud,           &
    zeeman,             &
    p,                  &
    t,                  &
    gas_units,          &
    mmr_cldaer,         &
    gas_id,             &
    gases,              &
    surfemisrefl,       &
    btrefl,             &
    rads,               &
    nchannels,          &
    ngases,             &
    nlevels,            &
    nprofiles)
```

Argument	Type	Intent	Description
err	Integer	out	Return code: non-zero implies error condition.
inst_id	Integer	in	ID of instrument (as returned by rttov_load_inst) of instrument to simulate.
channel_list(nchannels)	Integer	in	Channel numbers to simulate.
datetimes(nprofiles,6)	Integer	in	(year, month, day, hour, minute, second) for each profile.
angles(nprofiles,4)	Real	in	(zenangle, azangle, sunzenangle, sunazangle) for each profile.
surfgeom(nprofiles,3)	Real	in	(latitude, longitude, elevation) for each profile.
surftype(nprofiles,2)	Integer	in	(skin%surftype, skin%watertype) for each profile.
skin(nprofiles,9)	Real	in	(skin%t, skin%salinity, skin%snow_fraction, skin %foam_fraction, skin%fastem(1:5)) for each profile.
s2m(nprofiles,6)	Real	in	(s2m%p, s2m%t, s2m%q, s2m%u, s2m%v, s2m%wfetc) for

			each profile.
simplecloud(nprofiles,2)	Integer	in	(ctp, cfraction) for each profile.
clwscheme(nprofiles,2)	Integer	in	Visible/IR (clw_scheme, clwde_param) for each profile.
icecloud(nprofiles,2)	Integer	in	(ice_scheme, icede_param) for each profile.
zeeman(nprofiles,2)	Real	in	(Be, cosbk) for each profile.
p(nprofiles,nlevels)	Real	in	Pressure levels for each profile.
t(nprofiles,nlevels)	Real	in	Temperature on pressure levels for each profile.
gas_units	Integer	in	Set profile gas_units: 0=>ppmv over dry air; 1=>kg/kg; 2=>ppmv over moist air
mmr_cldaer	Integer	in	Set profile mmr_cldaer flag for cloud/aerosol units: non-zero=>cld/aer kg/kg; 0=>cld: g/m^3, aer: cm^-3
gas_id(ngases)	Integer	in	List of IDs for gases, aerosol and cloud profiles present in the gases array, see below.
gases(ngases,nprofiles,nlevels)	Real	in	Gas, aerosol and cloud concentrations on levels/layers for each profile: must contain at least water vapour profiles, see below.
surfemisrefl(5,nprofiles,nchannels)	Real	inout	Input surface emissivity, BRDF, diffuse reflectance, specularity, and effective Tskin values for each channel; on output contains the emissivity/reflectance values used by RTTOV, see below.
btrefl(nprofiles,nchannels)	Real	inout	Output total TOA brightness temperatures (for all channels at wavelengths > 3µm) or reflectances (wavelengths < 3µm).
rads(nprofiles,nchannels)	Real	inout	Output total TOA radiances.
nchannels	Integer	in	Number of channels to simulate (not required in Python).
ngases	Integer	in	Size of gas_id(:) array, see below (not required in Python).
nlevels	Integer	in	Number of levels in input profiles (not required in Python).
nprofiles	Integer	in	Number of profiles being passed in (not required in Python).

Notes:

If you extracted a subset of channels from the coefficient file in the rttov_load_inst call then the channel numbers in channel_list(:) are indexes into this list (see user guide section 7.4).

The array index ordering shown above is that which should be used in C/C++: this is opposite to Fortran array index ordering. For Python you should reverse the order of the indices for the 2- and 3-dimensional array arguments. It may also be more efficient to ensure that Python stores the arrays in Fortran-contiguous order. See the Python, C and C++ examples which illustrate how to declare the profile data arrays.

See Annex O and table 10 in the user guide for information about profile variables (the names in the table above relate to the names in the Fortran profile structure) and which variables are used in which circumstances. All arguments must be supplied to the interface, but if particular variables are not used in the simulations you are performing the arrays can just be initialised with zeros.

Surface emissivity/reflectance

You should refer to the user guide sections 7.5 and 7.6 to understand how RTTOV treats surface emissivity and reflectance.

The surfemisrefl(0,:,:) and surfemisrefl(1,:,:) arrays are used to control the input or calculation of surface emissivities and BRDFs respectively for all channels for each profile. If you provide non-negative (i.e. >=0) values for any channel then calcemis (or calcrefl) will be set to false for that channel and the supplied value is used for the surface emissivity (or BRDF). If a value in surfemisrefl(0/1,:,:) is negative then calcemis (or calcrefl) will be set to true.

If you wish to use the atlases you can call the rttov_get_emisbrdf subroutine to obtain the emissivity or BRDF values which should be passed into RTTOV via the surfemisrefl(0/1,:,:) arrays.

The surfemisrefl(2,:,:) array is used to provide diffuse reflectance values to RTTOV where relevant: these are only used for channels below 3µm where calcrefl is false (as determined by the corresponding BRDF value in surfemisrefl(1,:,:)) and the diffuse reflectance value is used only if it is >0. The surfemisrefl(3,:,:) array is used to specify the surface specularity which is used with the RTTOV do_lambertian option. Finally, the surfemisrefl(4,:,:) array is used to specify the per-channel effective skin temperatures which are used with the use_tskin_eff option. It is safe to set the entire surfemisrefl(:,:,:) to negative values, in which case calcemis and calcrefl are set to true, and the specularity will be set to zero in the simulations (NB in this case the use_tskin_eff option should be false).

On exit from the subroutine call the surfemisrefl(0/1/2,:,:) arrays are overwritten with the emissivity, BRDF and diffuse reflectance values used by RTTOV.

NB When making multiple calls to the wrapper interface be sure to re-initialise the surfemisrefl array appropriately between calls to avoid inadvertently passing in emissivity and reflectance values from the previous call. This applies to both direct and K model calls.

Specifying gas, aerosol and cloud profiles

RTTOV coefficient files support varying numbers of trace gases (see table 4 in section 3 of the user guide). In addition, IR cloud and aerosol simulations based on “method 1” (see user guide sections 8.5 and 8.6) require one or more profiles of cloud and aerosol concentrations and also a cloud fraction array for cloudy simulations. Any or all of these are supplied to the interface using the gases array.

The list of gas, aerosol and cloud inputs you wish to pass into RTTOV should be listed in the gas_id array. There is one element per input variable which should contain the corresponding ID for that variable (see appendix A of this document for the list of IDs). The gases array should then be populated with the appropriate concentrations in the corresponding order.

The gas_id array must always contain at least the water vapour ID (1) because this is a mandatory input for RTTOV. The order of the variables in gas_id and gases does not matter, but the two arrays must be consistent with one another.

Also note that aerosol and cloud inputs are on *layers* rather than *levels*: profiles of these variables

should be written to the first nlayers values in the array, the final value (at nlevels) is ignored.

As an example, suppose we wish to run an IR cloudy simulation with the STCO and ice cloud types. We must always include water vapour and the cloudy simulations also require cfrac (cloud fraction). Then the gas_id and gases arrays should be specified as follows (pseudo-code):

```
# ngases = 4, for gas IDs see appendix A:
#   1=>q, 20=>cfrac, 21=>STCO (cloud type 1), 30=>ice cloud (cloud type 6)
gas_id[:] = [1, 20, 21, 30]

# water vapour - on levels
gases[0:nprofiles, 0:nlevels, 0] = q[0:nprofiles, 0:nlevels]

# cfrac - on layers
gases[0:nprofiles, 0:nlevels-1, 1] = cfrac[0:nprofiles, 0:nlevels-1]

# STCO - on layers
gases[0:nprofiles, 0:nlevels-1, 2] = strat_cont[0:nprofiles, 0:nlevels-1]

# ice cloud - on layers
gases[0:nprofiles, 0:nlevels-1, 3] = ice_cloud[0:nprofiles, 0:nlevels-1]
```

Outputs

The output radiances and brightness temperatures (or reflectances for VIS/NIR channels) are written to the rads and btrefl arrays. These correspond to the radiance%total, radiance%bt and radiance%refl output arrays: the latter two are “merged” into the btrefl array such that for channels with wavelengths above 3µm BTs are stored while for other channels reflectances are stored. Additional subroutine calls are available which give access to all of the RTTOV radiance and transmittance outputs, assuming the relevant wrapper options were set (store_rad, store_rad2, store_trans): see section 3.1 and appendix B.

3.5. Calling the RTTOV K model

The RTTOV K model interface is similar in many ways to the direct model interface: arguments with the same name behave in exactly the same way as described in the previous section. The K call has some additional arguments to hold the input BT and/or radiance perturbations and the output profile variable Jacobians. The interface is described below with details given only for the K arguments not present in the interface for `rttov_call_direct`:

```
rttov_call_k( &
    err,                &
    inst_id,            &
    channel_list,       &
    datetimes,          &
    angles,             &
    surfgeom,           &
    surftype,           &
    skin,               &
    skin_k,             &
    s2m,                &
    s2m_k,              &
    simplecloud,        &
    simplecloud_k,      &
    clwscheme,          &
    icecloud,           &
    zeeman,             &
    p,                  &
    p_k,                &
    t,                  &
    t_k,                &
    gas_units,          &
    mmr_cldaer,         &
    gas_id,             &
    gases,              &
    gases_k,            &
    surfemisrefl,       &
    surfemisrefl_k,     &
    btrefl,             &
    rads,               &
    bt_k,               &
    rads_k,             &
    nchannels,          &
    ngases,             &
    nlevels,            &
    nprofiles)
```

Argument	Type	Intent	Description
skin_k(nprofiles,nchannels,9)	Real	inout	Calculated Jacobians for (skin%t, skin%salinity, skin %snow_fraction*, skin%foam_fraction, skin%fastem(1:5)). ** snow_fraction is not active in the RTTOV K model so the corresponding Jacobian is always zero.*
s2m_k(nprofiles,nchannels,6)	Real	inout	Calculated Jacobians for (s2m%p, s2m%t, s2m%q, s2m%u, s2m%v, s2m%wfetc).
simplecloud_k(nprofiles,nchannels,2)	Integer	inout	Calculated Jacobians for (ctp, cfraction).
p_k(nprofiles,nchannels,nlevels)	Real	inout	Calculated Jacobians for pressure.
t_k(nprofiles,nchannels,nlevels)	Real	inout	Calculated Jacobians for temperature.
gases_k(ngases,nprofiles,nchannels, nlevels)	Real	inout	Calculated Jacobians for gas, aerosol and cloud, variable order matches the input gas_id and gases arrays, see above.
surfemisrefl_k(5,nprofiles,nchannels)	Real	inout	Calculated Jacobians for surface emissivity, BRDF, diffuse reflectance, specularity, and effective Tskin.
bt_k(nprofiles,nchannels)	Real	in	Input BT perturbations (only for channels at wavelengths > 3µm).
rads_k(nprofiles,nchannels)	Real	in	Input radiance perturbations.

Notes:

The user guide provides more detailed information on calling the RTTOV K model. The input perturbations are supplied in brightness temperature (bt_k) for channels at wavelengths greater than 3µm if opts%rt_all%switchrad is set true in the options. Otherwise perturbations are supplied in radiance (rads_k). It is safe to set input perturbations in both bt_k and rads_k for all channels: RTTOV will use the appropriate perturbation for each channel based on the setting of the switchrad option.

The user guide notes that most Jacobian variables/structures should be initialised to zero before calling the K model: the wrapper takes care of this, so you only need to specify the non-zero perturbations as described above.

3.6. Calling the RTTOV direct model with explicit optical properties

This applies only to visible/IR sensors. You should read sections 8.5 and 8.6 of the user guide to understand the scattering options and inputs: this corresponds to “method 2”. For “method 1” where optical properties are taken from the cloud/aerosol coefficient files see section 3.4 above. When calling this interface either opts%rt_ir%addclouds or opts%rt_ir%addaerosl (or both) must be true and the corresponding opts%rt_ir%user_cld_opt_param or opts%rt_ir%user_aer_opt_param (or both) must be true. You can use optical properties from the relevant coefficient file for clouds or aerosols and supply explicit optical properties for the other via this interface: follow the procedure described in section 3.4 above for the pre-defined cloud/aerosol optical properties. The interface is as follows:

```
rttov_visir_scatt_call_direct( &
    err,                &
    inst_id,            &
    channel_list,       &
    datetimes,          &
    angles,             &
    surfgeom,           &
    surftype,           &
    skin,               &
    s2m,                &
    clwscheme,          &
    icecloud,           &
    p,                  &
    t,                  &
    gas_units,          &
    mmr_cldaer,         &
    gas_id,             &
    gases,              &
    aer_phangle,        &
    aer_asb,            &
    aer_legcoef,        &
    aer_pha,            &
    cld_phangle,        &
    cld_asb,            &
    cld_legcoef,        &
    cld_pha,            &
    surfemisrefl,       &
    btrefl,             &
    rads,               &
    nchannels,          &
    ngases,             &
    nlevels,            &
    nprofiles,          &
    aer_nphangle,       &
    aer_nmom,           &
    cld_nphangle,       &
    cld_nmom)
```

This subroutine call is rather similar to `rttov_call_direct` except for the additional optical

property inputs. Note that the simple_cloud and zeeman inputs are not present because these do not pertain to visible/IR scattering simulations. However the other inputs such as skin and s2m are identical even though some of the variables contained therein only apply to MW simulations.

There are additional optical parameter inputs: these are provided separately for aerosols and clouds. Optical property profiles are provided for each ***layer***, for each ***channel being simulated***, for each profile. You can call this subroutine for any subset of channels read from the coefficient file, but your optical property arrays must correspond to this channel_list argument. In contrast to the contents of the gases input array, the optical property arrays are all sized by nlayers (i.e. nlevels minus one). The inputs are described in the table below are for clouds: the aerosol ones are identical.

Argument	**Type**	**Description**
cld_asb(3,nprofiles,nchannels,nlayers)	Real	Absorption coefficients (cld_asb(1,:,:,:)), scattering coefficients (cld_asb(2,:,:,:)) and bpr parameters (cld_asb(3,:,:,:)). The absorption and scattering coefficients are required in all cases, units km^{-1}. The bpr values are only required for IR channels when Chou-scaling is used: they can be zero otherwise. See below for how to calculate bpr values.
cld_nphangle	Integer	Number of angles on which phase functions are defined. If solar radiation is not active this can be 1. (not required in Python).
cld_phangle(cld_nphangle)	Real	Angle grid on which phase functions are defined (degrees). First value must be 0° and final value must be 180°.
cld_pha(nprofiles,nchannels,nlayers, cld_nphangle)	Real	Azimuthally-averaged phase functions normalised such that the integral over all scattering angles is 4π. Phase functions are only required for solar-affected channels when opts%rt_ir %addsolar is true (i.e. when solar radiation is included).
cld_nmom	Integer	Number of Legendre coefficients provided for each phase function. If the DOM solver is not being used this can be zero. For DOM calculations this should be at least as large as the number of DOM streams being used (not required in Python).
cld_legcoef(nprofiles,nchannels,nlayers, cld_nmom+1)	Real	Legendre coefficients corresponding to each phase function. Note the final dimension is cld_nmom+1: this is consistent with the RTTOV internal structures: the “zeroth” coefficient is always 1. Legendre coefficients are only required for all channels for which the DOM solver is being used. See below for how to calculate Legendre coefficients.

Notes:

For cloud simulations you must always supply a cloud fraction profile: this is done via the “gases” input array as described in section 3.4.

The “store_rad2” option has no effect in this case as the secondary radiance outputs are not calculated for visible/IR scattering simulations.

For layers containing no cloud/aerosol the phase function values and Legendre coefficients can be zero.

If clouds or aerosols are not active in the simulation (i.e. addclouds or addaerosl is false) you can provide minimal arrays of zeros for the corresponding cloud/aerosol inputs. This can be achieved by setting the nphangle dimension to 1 and the nmom dimension to zero (recalling that the legcoef input has dimension nmom+1). Cloud and aerosol nphangle and nmom dimensions are independent.

Wrappers are provided for the RTTOV subroutines which calculate bpr values and Legendre coefficients from phase functions. The bpr calculation in particular is relatively expensive and as such is probably not suitable for calling within an operational system. In practice you may want to calculate the required bpr values off-line and store them for use in simulations.

`rttov_bpr(err, phangle, pha, bpr, nthreads, nphangle)`

Argument	Type	Intent	Description
err	Integer	out	Return code, non-zero value implies error.
phangle(nphangle)	Real	in	Angle grid on which phase functions are defined (degrees). First value must be 0° and final value must be 180°.
pha(nphangle)	Real	in	Azimuthally-averaged phase functions normalised such that the integral over all scattering angles is 4π.
bpr	Real	out	Calculated bpr value.
nthreads	Integer	in	Number of OpenMP threads to use in the calculation (has no effect unless RTTOV is compiled with OpenMP).
nphangle	Integer	in	Number of angles on which phase functions are defined (not required in Python).

`rttov_legcoef(err, phangle, pha, legcoef, ngauss, nphangle, nmom)`

Argument	Type	Intent	Description
err	Integer	out	Return code, non-zero value implies error.
phangle(nphangle)	Real	in	Angle grid on which phase functions are defined (degrees). First value must be 0° and final value must be 180°.
pha(nphangle)	Real	in	Azimuthally-averaged phase functions normalised such that the integral over all scattering angles is 4π.
legcoef(nmom+1)	Real	inout	Calculated Legendre coefficients.
ngauss	Integer	in	Legendre coefficients are calculated using Gaussian quadrature. By default the quadrature size is 1000 points. You can specify a different quadrature size using this parameter. Note that the input value must be greater than or equal to nmom otherwise it is ignored.
nphangle	Integer	in	Number of angles on which phase functions are defined (not required in Python).
nmom	Integer	in	Number of Legendre coefficients to calculate. For DOM calculations this should be at least as large as the number of DOM streams being used (not required in Python).

3.7. Calling the RTTOV K model with explicit optical properties

This is very similar to the direct model interface described in the previous section and in terms of the Jacobian calculations it is very similar to the K model interface described in section 3.5 above.

```
rttov_visir_scatt_call_k( &
    err,                &
    inst_id,            &
    channel_list,       &
    datetimes,          &
    angles,             &
    surfgeom,           &
    surftype,           &
    skin,               &
    skin_k,             &
    s2m,                &
    s2m_k,              &
    clwscheme,          &
    icecloud,           &
    p,                  &
    p_k,                &
    t,                  &
    t_k,                &
    gas_units,          &
    mmr_cldaer,         &
    gas_id,             &
    gases,              &
    gases_k,            &
    aer_phangle,        &
    aer_asb,            &
    aer_legcoef,        &
    aer_pha,            &
    cld_phangle,        &
    cld_asb,            &
    cld_legcoef,        &
    cld_pha,            &
    surfemisrefl,       &
    surfemisrefl_k,     &
    btrefl,             &
    rads,               &
    bt_k,               &
    rads_k,             &
    nchannels,          &
    ngases,             &
    nlevels,            &
    nprofiles,          &
    aer_nphangle,       &
    aer_nmom,           &
    cld_nphangle,       &
    cld_nmom)
```

The K variables are exactly the same as those described in section 3.5 above. Note that the explicit

optical properties have not been implemented as “active” variables in the K model wrapper so Jacobians are not calculated for them.

3.8. Calling the RTTOV-SCATT direct model

This applies only to MW sensors. You should see section 8.7 of the user guide which describes RTTOV-SCATT and also Annex O which describes the options and additional input data relevant to RTTOV-SCATT. A hydrotable file must have been specified and loaded alongside the optical depth coefficient file. RTTOV-SCATT **requires that all channels are read from the coefficient file** when the instrument is loaded. If a hydrotable file has been specified the wrapper enforces this and will print a warning if you supplied a channel_list to `rttov_load_inst`.

This interface is similar in many ways to the direct model interface described in section 3.4. However as this is specifically for MW simulations some irrelevant profile variables are omitted.

```
rttov_scatt_call_direct( &
    err,                &
    inst_id,            &
    channel_list,       &
    datetimes,          &
    angles,             &
    surfgeom,           &
    surftype,           &
    skin,               &
    s2m,                &
    zeeman,             &
    p,                  &
    t,                  &
    gas_units,          &
    gas_id,             &
    gases,              &
    ph,                 &
    cfrac,              &
    multi_hydro_frac,   &
    calc_zef,           &
    surfemis,           &
    bt,                 &
    nchannels,          &
    ngases,             &
    nlevels,            &
    nprofiles)
```

The following table details only those inputs which differ to the direct model call described in section 3.4. See the user guide for more information about RTTOV-SCATT inputs.

Argument	Type	Intent	Description
angles(nprofiles,2)	Real	in	(zenangle, azangle) for each profile.
surftype(nprofiles)	Integer	in	skin%surftype for each profile.
skin(nprofiles,8)	Real	in	(skin%t, skin%salinity, skin%foam_fraction, skin %fastem(1:5)) for each profile.

s2m(nprofiles,5)	Real	in	(s2m%p, s2m%t, s2m%q, s2m%u, s2m%v) for each profile.
zeeman(nprofiles,2)	Real	in	(Be, cosbk) for each profile.
gas_id(ngases)	Integer	in	List of IDs for water vapour, cloud and hydrometeors present in the gases array, see below.
gases(ngases,nprofiles,nlevels)	Real	in	Water vapour, cloud and hydrometeor concentrations on levels for each profile: must contain at least water vapour profiles, see below.
ph(nprofiles,nlevels+1)	Real	in	Pressure half-levels (see user guide).
cfrac(nprofiles)	Real	inout	User-specified cloud fraction, only used if opts_scatt %lusercfrac is true, otherwise contains the values calculated by RTTOV-SCATT on exit (see user guide).
multi_hydro_frac	Logical	in	False => a single cloud fraction profile is input; True => one cloud fraction per hydrometeor is input (see user guide).
calc_zef	Logical	in	False => standard RTTOV-SCATT passive simulation; True => RTTOV-SCATT radar simulation, requires compatible hydrotable file (see user guide).
surfemis(2,nprofiles,nchannels)	Real	inout	Input surface emissivity and effective Tskin values for each channel; on output contains the values used by RTTOV.
bt(nprofiles,nchannels)	Real	inout	Output total TOA brightness temperatures.

Notes:

RTTOV-SCATT does not produce transmittance outputs or radiance outputs and as such the “store_trans” and “store_rad2” options have no effect. If the “store_rad” option is true you can access only the bt, bt_clear, geometric_height and quality outputs. If the “store_emis_terms” option is true you can access the additional emissivity retrieval radiance and transmittance outputs.

Radar simulations can be run for sensors with radar-enabled hydrotable files by setting calc_zef to true. The reflectivity outputs (zef, azef) are available in the same way as radiance/BT outputs (see appendix B).

As surface reflectances and the Lambertian surface option (and hence surface specularity) are not relevant to MW simulations, only emissivity and effective skin temperatures (when use_tskin_eff is true) are inputs. Aside from the difference in the size of the array, this operates in exactly the same way as for the standard RTTOV calls and you can use the MW emissivity atlases with RTTOV-SCATT in the same way.

The gas_id and gases arrays are populated as described in section 3.4. For RTTOV-SCATT only water vapour (mandatory), ozone (optional for relevant sensors), and the RTTOV-SCATT cloud and hydrometeor gas IDs (see appendix A) will be used: any other inputs present in the gases array will be ignored. Gas IDs are provided for the five hydrometeor types in the default hydrotables and for a single hydrometeor cloud fraction (multi_hydro_frac = false). Separate IDs are available for arbitrary numbers of hydrometeors (in custom hydrotable files) and for per-hydrometeor cloud fractions, up to a maximum of 30 particle types.

3.9. Calling the RTTOV-SCATT K model

This is very similar to the K model interface described above in section 3.5 and shares many arguments with the RTTOV-SCATT direct model interface described in the previous section.

```
rttov_scatt_call_k( &
    err,                &
    inst_id,            &
    channel_list,       &
    datetimes,          &
    angles,             &
    surfgeom,           &
    surftype,           &
    skin,               &
    skin_k,             &
    s2m,                &
    s2m_k,              &
    zeeman,             &
    p,                  &
    p_k,                &
    t,                  &
    t_k,                &
    gas_units,          &
    gas_id,             &
    gases,              &
    gases_k,            &
    ph,                 &
    ph_k,               &
    cfrac,              &
    cfrac_k,            &
    multi_hydro_frac,   &
    calc_zef,           &
    surfemis,           &
    surfemis_k,         &
    bt,                 &
    bt_k,               &
    zef_k,              &
    nchannels,          &
    ngases,             &
    nlevels,            &
    nprofiles)
```

The following table lists only those inputs which are not present in the interface to the RTTOV-SCATT direct model interface:

Argument	Type	Intent	Description
skin_k(nprofiles,nchannels,8)	Real	inout	Calculated Jacobians for (skin%t, skin%salinity, skin %foam_fraction, skin%fastem(1:5)).
s2m_k(nprofiles,nchannels,5)	Real	inout	Calculated Jacobians for (s2m%p, s2m%t, s2m%q, s2m%u, s2m%v).
p_k(nprofiles,nchannels,nlevels)	Real	inout	Calculated Jacobians for pressure.

t_k(nprofiles,nchannels,nlevels)	Real	inout	Calculated Jacobians for temperature.
gases_k(ngases,nprofiles,nchannels,nlevels)	Real	inout	Calculated Jacobians for water vapour, cloud, and hydrometeors, variable order matches the input gas_id and gases arrays, see above.
ph_k(nprofiles,nchannels,nlevels)	Real	inout	Calculated Jacobians for pressure half-levels.
cfrac_k(nprofiles,nchannels)	Real	inout	Calculated Jacobians for user-specified cloud fraction.
surfemis_k(2,nprofiles,nchannels)	Real	inout	Calculated Jacobians for surface emissivity and effective Tskin.
bt_k(nprofiles,nchannels)	Real	in	Input BT perturbations for standard (passive) simulations.
zef_k(nprofiles,nchannels,nlevels)	Real	in	Input reflectivity Zef perturbations for radar simulations.

3.10. Deallocating memory

When you have finished calling RTTOV you should make a call to release the memory allocated by the wrapper.

If you simply wish to free all memory allocated by the wrapper for all loaded instruments and atlases you can call:

```
rttov_drop_all(err)
```

Here err is the usual intent(out) return code (non-zero implies an error condition).

Alternatively you can deallocate memory for specific instruments or atlases.

You can deallocate the memory for a single instrument using:

```
rttov_drop_inst(err, inst_id)
```

Again err is the return code and inst_id is the ID of the instrument to deallocate.

You can deallocate memory for a specific atlas using:

```
rttov_drop_atlas(err, atlas_wrap_id)
```

The atlas_wrap_id argument is the wrapper ID for previously loaded atlas data and err is the return code.

4. Specific information for Python

By default integers are 32-bit (e.g. numpy.int32) and reals are 64-bit (e.g. numpy.float64).

The error return code arguments (err) which are INTENT(OUT) appear as return values to the Python function call and as such do not appear among the function arguments. This also applies to inst_id in calls to `rttov_load_inst`.

In addition the array size arguments listed in section 3 are implicit in the Python interface: they are calculated from the dimensions of the input arrays and do not appear among the function arguments.

For example in Python the wrapper initialisation call looks like this:

```
> inst_id = rttov_load_inst(opts_str, channels)
```

Note inst_id is the return value and nchannels is implicitly determined from len(channels) by the interface and is not present as an argument.

You should declare all Python arrays with array indices in the opposite order to those listed above. You may also want to ensure they are in Fortran-contiguous order in memory by supplying the order='F' argument to the Numpy array initialisation calls. The example code provides illustrations of how to declare array arguments.

5. Specific information for C/C++

By default integers are 32-bit (e.g. C int) and reals are 64-bit (e.g. C double).

When passing a character string argument to Fortran from C/C++ it is necessary to include the string length as an additional argument. Usually this is appended as the final argument in the call, but for some compilers it may need to be supplied directly following the string argument. See the example C and C++ code: this applies to `rttov_load_inst`, `rttov_set_options` and the atlas initialisation subroutines.

The C-style array index ordering is opposite to that used in Fortran. You should allocate arrays with dimensions as shown in this document to ensure data is passed correctly between your C or C++ code and the RTTOV Fortran code.

All interface subroutine names should have an underscore appended '_' as in src/wrapper/rttov_c_interface.h. See this header file for interfaces to all wrapper subroutines.

6. RTTOV classes

C++ object-oriented interface

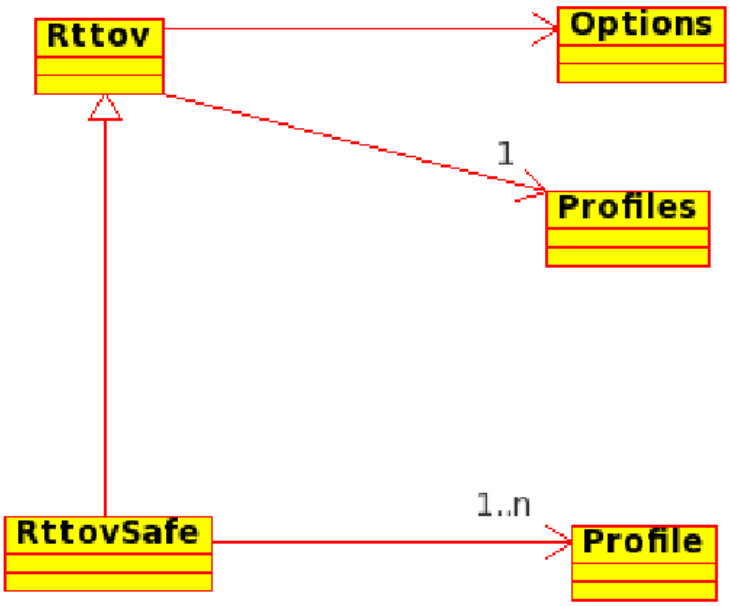
A number of C++ classes have been created in order to provide an object-oriented interface to RTTOV: **Rttov**, **RttovSafe**, **RttovScatt, RttovScattSafe, Options**, **Profiles, Profile, ProfilesScatt, ProfileScatt** and **Atlas**.

RttovSafe and **Rttov** are the primary classes used to call RTTOV: one instance of either class is associated with one instrument.

The **Rttov** object is a fast way to call RTTOV and would usually be associated with a **Profiles** instance which represent one or more RTTOV profiles structures in the form of a collection of arrays.

The **RttovSafe** object provides a safer way to call RTTOV because it carries out some checks on the input profiles before passing them to the RTTOV interface (see below). This is a more user-friendly, but (very slightly) less efficient way to call RTTOV. It is associated with a C++ vector of one or more instances of the **Profile** object each of which represents a single RTTOV profile structure.

The following diagram illustrates the relationship between the classes:



The **Profile** object is designed to handle one vertical profile which is the smallest possible input on which to run RTTOV. The private members of the Profile objects are **vectors** which are safer to use than pointers, and the methods allow the user to populate the **Profile** instance in a friendly way with vectors as entries, or separate values (like with the **setAngles** method). This is in contrast to the **Profiles** object used with the **Rttov** class which uses pointers to manage profile data.

The association between the **RttovSafe** instance and the **Profile** object instance is made with the **RttovSafe.setTheProfiles** method. This methods takes as argument a vector of instances of the **Profile** object. The other methods of the **RttovSafe** class are inherited from the **Rttov** class.

The **RttovSafe.setTheProfiles** method makes the following checks**:**

- ensures the input is a vector of **Profile** objects
- ensures the vector is not empty
- ensure all the profiles have the same number of levels
- if pressure is not filled for the first profile:
  - ensure the number of levels of the profile is the same of the number of levels of the coefficient file: in this case the pressures levels of the coefficient file are used.
- check if all **Profile** objects in the input vector have the same content (gas, aerosols, and clouds), gas_units, mm_cld_clear
- for each profile of the input vector call the check method of the **Profile** object.

The **Profile.check** method makes the following checks:

- ensures all mandatory fields are provided, but does not perform a check upon the values (this is done within RTTOV itself)
- if simplecoud, clwscheme, icecloud or zeeman have not been set initialise them with default values.

Each **Rttov** and **RttovSafe** object is associated with an instance of the **Options** class which represents the RTTOV options structure and also some additional options specific to the wrapper.

It is also possible to use the RTTOV land surface emissivity and BRDF atlases through the **Atlas** object: this is used to obtain emissivity and BRDF values which can be passed to an **Rttov** or **RttovSafe** object.

The **RttovScatt** and **RttovScattSafe** classes are used when calling RTTOV-SCATT for MW scattering simulations. These are quite similar to **Rttov** and **RttovSafe** and the descriptions which follow apply equally to all four classes except where it explicitly states otherwise. The **ProfilesScatt** and **ProfileScatt** classes are used for defining profile data which can be associated with the **RttovScatt** and **RttovScattSafe** classes.

In reading the descriptions of the classes below you should refer to the user guide to understand the RTTOV input and output structures including the options and profiles structures and other aspects of RTTOV such as the treatment of surface emissivity and BRDF. You should also refer to the example code in the wrapper/ directory which provides examples of using these classes.

All classes and associated enumerations are defined within the **rttov::** namespace.

The following documentation for these classes assumes you are familiar with C++ programming.

Python pyrttov package

The Python implementation of the object-oriented interface follows the C++ version closely, but there are some important differences:

- to use the package it needs to be in your $PYTHONPATH (or the current directory) and you can just use `import pyrttov`.
- the **pyrttov** package includes only **Options, Profiles, ProfilesScatt, Rttov, RttovScatt** and **Atlas** classes. The classes carry out a lot of checks so there is no need for the "safe" versions as in the C++ interface.
- there are no get/set methods to return or specify options, profile variables and outputs. Instead you can refer to the members directly. The member names are identical to those for the C++ classes with the "get"/"set" omitted (see the following sections for examples and also the example code provided).
- You can use the Python **help()** functionality to obtain documentation about any **pyrttov** object or object method. For the objects, this displays searchable information about all methods and members. For example:

```
myrttov = pyrttov.Rttov()
help(myrttov)
myprofiles = pyrttov.Profiles(1, 54)
help(myprofiles)
```

Note that for the **pyrttov** package the array index ordering is **the same as** the C/C++ ordering (which is contrary to the order required by the Python interface described in sections 3 and 4 above). Therefore the array ordering is the same for the C++ and Python classes.

The following sections describe both the C++ and Python classes. Where the documentation mentions the "**Rttov** or **RttovSafe**" classes, in Python this means just the **Rttov** class. Where there are important differences between the Python and C++ these are highlighted, but note that where the documentation refers to get/set methods these apply to the C++ classes and in the Python you use the member variable directly (same name omitting "get"/"set") to return data ("get") or to assign values ("set"). Where the **RttovScatt** or **RttovScattSafe** classes differ to **Rttov**/**RttovSafe**, this is highlighted, otherwise the descriptions also apply to the RTTOV-SCATT classes.

6.1. General method for calling RTTOV

An instance, say "myRttov", of either the **Rttov** or **RttovSafe** classes (C++) or the **Rttov** class (Python) should be declared. Each such instance represents a single instrument to simulate. The methods of the **RttovSafe** and **Rttov** C++ classes are given in Appendix C: the majority of methods are common to both classes. The difference is in the way the profile data are associated with instances of each class. The methods and members of the Python **Rttov** class are also given in Appendix C. The **RttovScatt** and **RttovScattSafe** methods and members are given in Appendix D.

The general steps for calling RTTOV via the object-oriented interface are similar to those described in the user guide. This typically involves:

- setting the RTTOV options
- loading an instrument
- optionally initialising the emissivity and/or BRDF atlases
- specifying the surface emissivities and reflectances
- specifying the profile data to simulate
- calling RTTOV
- accessing the simulation outputs
- deallocating memory

Each of these steps is described in more detail below.

6.2. Setting RTTOV options

This myRttov object has a member named “options” (C++) or “Options” (Python) which is an instance of the **Options** class. This is used to specify the RTTOV and wrapper-specific options. The methods (C++) and members (Python) of this class are listed in Appendix I. The user guide describes the RTTOV options (see Annex O). See section 3.1 above for a description of the wrapper-specific options. RTTOV-SCATT exposes only a subset of options to the user (see Annex O of the user guide). These are also available through the **Options** class (appendix I).

In C++: to change an option associated with an **Rttov/RttovSafe** instance named “myRttov” you should use, for example:

```
myRttov.options.setApplyRegLimits(true);
```

In Python the equivalent statement is:

```
myRttov.Options.ApplyRegLimits = True
```

6.3. Loading an instrument

The name of the optical depth (“rtcoef_”) coefficient file should be specified by calling the **myRttov.setFileCoef** method (C++) or assigning to **myRttov.FileCoef** (Python). If required the VIS/IR cloud and/or aerosol coefficient file names should also be specified using the **setFileSccld** and **setFileScaer** methods respectively. For MFASIS simulations the MFASIS LUT/NN should be specified using **setFileMfasisCld/setFileMfasisNN**. For RTTOV-SCATT simulations the hydrotable filename must be specified using the **setFileHydrotable** method: this is compulsory with **RttovScatt/RttovScattSafe** objects. When using the ARO-scaling polarisation option, set the **polMode** option and specify the location of the ARO-scaling look-up table file using **setFilePol** before reading the coefficients.

The coefficients are read in by calling the **myRttov.loadInst** method. If called without arguments all channels are read from the coefficient file. Alternatively a C++ vector/numpy array of channel

numbers may be specified in order to read coefficients for a subset of channels. Note that if a subset of *n* channels is read, they are referenced by numbers 1...*n* subsequently rather than by their original channel numbers as described in the RTTOV user guide. For RTTOV-SCATT all channels must be read so there is no channel list argument available to the **loadInst** method of **RttovScatt/RttovScattSafe**.

After an instrument has been loaded the options can be changed. If you call the **myRttov.updateOptions** method and the wrapper "check_opts" option is set to true this will force a consistency check on the options and loaded coefficients and will report any errors which can be useful for debugging simulations. The **myRttov.printOptions** method will print out the options structure (this calls the rttov_print_opts or rttov_print_opts_scatt Fortran subroutines). Note that changing the coefficient filename(s) after loading the instrument will have no effect.

6.4. Specifying surface emissivities and reflectances

You can pass your own values for surface emissivity and/or reflectance into RTTOV or RTTOV can provide suitable values. The user guide provides full details of the treatment of surface emissivity and reflectance. You should declare an array **surfemisrefl** with dimensions *[5][nprofiles][nchannels]*.This should be initialised before every call to RTTOV. The first dimension of this array provides access to emissivity (index 0), BRDF (index 1), diffuse reflectance (index 2), specularity (index 3), per-channel effective Tskin (index 4) for all channels and profiles being simulated.

Where emissivity/BRDF values in this input array are greater than or equal to zero the corresponding elements of the RTTOV calcemis/calcrefl arrays will be set to false respectively, and these input values of the surface parameters will be used for the simulations. Where the emissivity/BRDF values in **surfemisrefl** are less than zero the corresponding elements of the RTTOV calcemis/calcrefl arrays respectively will be set to true and RTTOV will provide values using its internal models (see the user guide for more details). The emissivity and BRDF atlases can be used to provide input values for emissivity and BRDF: this is described in the next section.

For relevant channels, if the input diffuse reflectance values are greater than zero they will be used if calcrefl is false for the corresponding channel.

The surface specularity values are used when the Lambertian surface option is activated: if the input values are less than zero, then the wrapper will set them to zero when calling RTTOV.

The effective Tskin values are used with the **useTskinEff** option is true.

The **surfemisrefl** array is associated with the **myRttov** instance using the **setSurfEmisRefl** method (C++) or assigning to the **SurfEmisRefl** member (Python).

After RTTOV has been called the surfemisrefl array contains the emissivity, BRDF and diffuse reflectance values that were used by RTTOV. This can be accessed via the **getSurfEmisRefl** method (C++) or via the **SurfEmisRefl** member (Python).

NB When making multiple calls to RTTOV be sure to re-initialise the surfemisrefl array appropriately between calls to avoid inadvertently passing in emissivity and BRDF values from the previous call. This applies to both direct and K model calls.

It is not mandatory to specify/set the **surfemisrefl** array. If you do not then it is equivalent to setting

calcemis and calcrefl to true for all channels and setting the specularity to zero. In this case the **useTskinEff** option must be false. After calling RTTOV you can obtain the emissivities/reflectances as described above. In this case you do not need to worry about reinitialising the emissivities/reflectances between multiple calls as the wrapper takes care of that. In **pyrttov** if you have assigned an array to **SurfEmisRefl** and you wish to delete this before making another call to RTTOV you can use

```
del myRttov.SurfEmisRefl
```

For RTTOV-SCATT simulations, surface reflectance and specularity are not required. Emissivity and effective Tskin may be specified. For the **RttovScatt**/ **RttovScattSafe** classes the corresponding methods are setSurfEmis, getSurfEmis (C++) and SurfEmis (Python). In this case the arrays have dimensions [2][nprofiles][nchannels] (index 0 for emissivity, 1 for effective Tskin). In all other respects the surface emissivity inputs behave the same as in the **Rttov**/**RttovSafe** classes including use of the emissivity atlases (see below).

6.5. Using the emissivity and BRDF atlases

An instance, say “myAtlas”, of the **Atlas** class can be declared. Each such instance is used to contain data from one of RTTOV's atlases for a specific month and, where relevant, for a specific instrument. Unlike previous versions of RTTOV, any combination of atlases and months can be used: each **Atlas** object is independent. The methods and members of the **Atlas** class are described in Appendix J. You should also read the relevant section of the user guide to understand what atlases are available and how they work.

Loading atlas data

The path to the atlas data to be loaded must first be specified via the **setAtlasPath** method (C++) or the **AtlasPath** member (Python).

The atlas data are then read via one of three methods: **loadBrdfAtlas**, **loadIrEmisAtlas** or **loadMwEmisAtlas**. In each case the month of the data to be loaded is specified. The atlas_id argument is used to specify which of the available atlases of the relevant type is to be loaded. The load methods return a Boolean value indicating success (true) or failure (false).

The BRDF and IR emissivity atlases can optionally be loaded for a specific instrument (in which case access to the atlases is significantly faster) and the CNRM MW emissivity atlas ***must*** be loaded for a specific instrument. The instrument is specified by passing an **Rttov/RttovSafe** object to the relevant load method. The instrument itself must have been loaded before the **Atlas** object is initialised.

If you wish to use the BRDF or IR emissivity atlas data with any compatible instrument then do not pass an **Rttov/RttovSafe** object to the Atlas load method. The TELSEM2 MW atlas is never initialised for use with a specific instrument and in this case any **Rttov/RttovSafe** object passed to the load method is ignored.

Obtaining emissivitiy/BRDF values

The process for returning emissivity/BRDF differs between C++ and Python:

In C++ the **fillEmisBrdf** method is used: this requires you to allocate a suitable array (for example the **surfemisrefl** array used by the **Rttov** and **RttovSafe** objects). A pointer to this array is passed to the subroutine and the array is filled with values from the atlas.

In Python the **getEmisBrdf** method is used: this returns a two-dimensional array of size [nprofiles][nchannels] containing the emissivity or BRDF values.

In both cases you must also pass an **Rttov**/**RttovSafe** object to the **getEmisBrdf** method: the instrument must have been loaded and it must have one or more profiles associated with it. The profile data are used when retrieving emissivities/BRDFs from the atlas: see the user guide for information on which profile variables are used by each atlas. You can also optionally specify a channel list (in C++ this is a vector of ints): this should usually match the channel list you will pass into the call to RTTOV (see below). If the channel list is omitted, emissivity/BRDF values are returned for all channels of the loaded instrument.

The various atlases behave differently for profiles with different surface types (specified in profiles(:)%skin%surftype in the Fortran). This is described in the user guide. To provide more control over the atlases, the **Atlas** object has three flags: **IncLand**, **IncSea** and **IncSeaIce** which can be accessed via get/set methods in C++ or accessed directly in Python as usual. When one or more of these flags is true the atlas will be called for profiles with the corresponding surface type and any returned values will be output in the emissivity/BRDF array. If the flag is false then emissivities/BRDFs for profiles of that surface type will be left as they are by the call to **fillEmisBrdf** (C++) or will be filled with negative values in the array returned by **getEmisBrdf** (Python). By default all three flags are true so the atlases are called for all profiles.

Deallocating atlas data

When the **Atlas** destructor is called any associated data is deallocated so you do not have to worry about deallocating data manually. However you can deallocate the data in an **Atlas** object so that it can be re-used by calling the **dropAtlas** method.

6.6. Profile data for an RttovSafe object (C++ only)

The **Profile** class represents a single RTTOV profile structure. It is used to provide the atmospheric and surface variables to the **RttovSafe** instance in the form of a C++ vector of **Profile** objects. The methods of the **Profile** class are given in Appendix E.

A **Profile** object is instantiated as follows, where *nlevels* is the number of levels for the profile:

```
rttov::Profile myProfile(nlevels);
```

You can then use the methods listed in Appendix E to specify the profile variables. Many of these methods are self-explanatory: for example, the **setT** method is used to specify the temperature profile.

When doing visible/IR cloud and/or aerosol simulations the cloud, cfrac and aerosol profiles input

to RTTOV are defined on atmospheric layers. However they must be supplied to the **Profile** object as an array of *nlevels* elements: the final element is ignored.

If you are running aerosol simulations with a standard OPAC or CAMS *scaercoef* aerosol optical property file there are specific methods to set each individual aerosol species (e.g. **setInso** or **setBcar**). If you are using a custom *scaercoef* file then the individual aerosol profiles are specified using the **setUserAerN** method. The *scaercoef* file must not contain more than 30 aerosol species. Note that you can use this latter method to specify OPAC or CAMS aerosols, but in this case you must not use the individual methods (**setInso**, etc) and Jacobians are accessed via **getUserAerNK** (see section 6.12).

The **setGasUnits** method takes an argument of type **rttov::gasUnitType** which is defined in wrapper/rttov_common.h. The constants of this enumeration are listed in Appendix K. If unspecified the default is ppmv over moist air, but a warning is printed if you do not set this explicitly.

The **setAngles**, **setS2m**, **setSkin, setSurfType**, **setSurfGeom** and **setDateTimes** methods must all be called for every **Profile** instance. Each of these methods sets a collection of related profile variables: the RTTOV user guide provides more information on which variables are required for particular types of simulations. If an argument to one of these subroutines corresponds to a variable which is not relevant to your simulations you can set it to zero. The table at the end of section 6.8 lists the variables that must be specified in each array (the order of the variables is important).

The **setSimpleCloud**, **setClwScheme**, **setIceCloud** and **setZeeman** methods do not need to be called unless you require the corresponding variables to be specified in your simulations. If unspecified the **Profile** object will set the values of the corresponding profile variables to suitable defaults or to zero.

If you are not using the RTTOV interpolator you do not need to specify the pressure levels. Instantiate the **Profile** object with the same number of levels as the coefficient file is based on (usually 54 or 101) and the pressure profile from the coefficient file will be used by default unless you specify a different set of pressure levels using the **setP** method.

Once a **Profile** object has been populated with profile data it can be stored in a C++ vector of **Profile** objects. For example:

```
std::vector <rttov::Profile> profiles;
profiles.push_back(myProfile);
```

This can be repeated for every profile to be simulated. Once the collection of **Profile** instances is fully populated it is associated with the **RttovSafe** instance by calling the **myRttov.setTheProfiles** method. This performs some checks on the profiles before RTTOV is called which helps to prevent errors. It is very important that ***all*** profile data are associated with the **Profile** object ***before*** it is associated with the **Rttov/RttovSafe** instance.

6.7. Profile data for an RttovScattSafe object (C++ only)

The **ProfileScatt** class represents a single profile structure for input to RTTOV-SCATT. It is used to provide the atmospheric and surface variables to the **RttovScattSafe** instance in the form of a C++ vector of **ProfileScatt** objects. The methods of the **ProfileScatt** class are given in Appendix G.

The **ProfileScatt** class is similar in many ways to the **Profile** class so most of the description in the previous section applies here. However since **ProfileScatt** is used specifically for MW scattering simulations, not all RTTOV profile variables are relevant, some arrays have slightly different dimensions and some additional profile variables may be specified. In particular the input hydrometeor arrays are defined on nlevels (unlike the case for visible/IR scattering where they are on nlayers) and the pressure half-levels profile has size (nlevels+1). You must always specify the pressure levels for RTTOV-SCATT: there is no option to use the optical depth coefficient levels. Similarly some of the arrays which group profile variables together are different to those in the **Profile** class: the table at the end of section 6.9 lists the variables that must be specified in each array.

Concentration profiles for the five default hydrometeor types can be specified using individually named methods (e.g. **setRain**), and a single hydrometeor cloud fraction (multi_hydro_frac = false) can be specified via **setHydroFrac**. To specify individual cloud fraction profiles per hydrometeor instead use **setHydroFracN**. Similarly, if you are using a custom hydrotable file, you can use the **setHydroN** method to specify the different hydrometeor profiles for up to 30 hydrometeor types.

The **setZeeman** method does not have to be called unless you require the corresponding variables to be specified in your simulations.

Just as for **Profile** objects, once the collection of **ProfileScatt** instances is fully populated it is associated with the **RttovScattSafe** instance by calling the **myRttov.setTheProfiles** method. It is very important that ***all*** profile data are associated with the **ProfileScatt** object ***before*** it is associated with the **RttovScatt/RttovScattSafe** instance.

6.8. Profile data for an Rttov object (C++ and Python)

The **Profiles** class represents one or more RTTOV profile structures. The atmospheric profiles and other variables are specified as a series of arrays. An instance of the **Profiles** class is then provided to the **Rttov** instance. The methods (C++) and members (Python) of the **Profiles** class are given in Appendix F.

A **Profiles** object is instantiated as follows, where *nprofiles* is the number of profiles and *nlevels* is the number of levels in each profile.

In C++:

```
rttov::Profiles myProfiles(nprofiles, nlevels);
```

In Python:

```
myProfiles = pyrttov.Profiles(nprofiles, nlevels)
```

In C++ the data for each profile variable is provided to the Profiles instance as a pointer to an array containing the data for every profile using the relevant method. For example, the **setT** method assigns the temperature profiles to the **Profiles** instance. There are methods for setting profile data for each trace gas and the pressure levels.

In Python numpy arrays are assigned directly to the member variables of the **myProfiles** object (e.g. `myProfiles.T = temperature_array` for the temperature profiles). Profiles for each trace gas and the pressure levels can be set in the same way.

For atmospheric profile variables like temperature and gas abundances you must create an array of size *[nprofiles][nlevels]* and populate it with the atmospheric profile values for every profile.

When doing visible/IR cloud and/or aerosol simulations the cloud, cfrac and aerosol profiles input to RTTOV are defined on atmospheric layers. However they must be supplied to the **Profiles** object as arrays of *[nprofiles][nlevels]* elements (as for temperature and gases): the final element of each profile is ignored.

In C++ to supply the cloud and aerosol profiles you must use the **setGasItem** method which takes the profile as input and an ID for the profile variable being set. This second argument is of type **rttov::itemIdType**: this enumeration is defined in wrapper/rttov_common.h and a complete list of the associated constants is given in Appendix K. (You can also set the gas profiles using this method, but it is clearer to use the methods like **setQ** which are particular to each gas).

In Python there is no equivalent to **setGasItem**: the individual cloud and aerosol profile variables can be assigned directly by name. For example, **myProfiles.Cfrac = cfrac** (cloud fraction), **myProfiles.Cirr = ciw** (cloud ice water), **myProfiles.Inso = aer_inso** (insoluble aerosol). For aerosols this applies to both OPAC and CAMS *scaercoef* aerosol optical property files. If you are running simulations with a custom *scaercoef* file you can either use the **AerN** (N=1,2,...,30) members of **Profiles** or the **setUserAerN** method. The *scaercoef* file must not contain more than 30 aerosol species. Note that you can use this latter approach (**AerN**/**setUserAerN**) to specify OPAC or CAMS aerosols, but in this case you must not use the individual members (**Inso**, etc) and Jacobians are accessed via **getUserAerNK** or **AerNK** (see section 6.12).

The **setGasUnits** method takes an integer argument: see the RTTOV user guide for valid values. If unspecified the default is ppmv over moist air.

In C++ the **setAngles**, **setS2m**, **setSkin, setSurfType**, **setSurfGeom** and **setDateTimes** methods must all be called for each **Profiles** instance in C++. Each of these methods sets a collection of related profile variables. The argument to each method is a two dimensional array (see Appendix F). The first dimension is *nprofiles*, and the second dimension depends on the number of variables being set by each method (see table below). The RTTOV user guide provides more information on which variables are required for particular types of simulations: if an element of an array argument to one of these subroutines corresponds to a variable which is not relevant to your simulations you can set it to zero.

The **setSimpleCloud**, **setClwScheme**, **setIceCloud** and **setZeeman** methods do not need to be called unless you require the corresponding variables to be specified in your simulations. If unspecified the **Profiles** object will set the values of the corresponding profile variables to zero (or to suitable defaults).

In Python the same applies except that the equivalent member arrays (**Angles**, **S2m**, **SimpleCloud**, etc) are assigned for each **Profiles** instance rather than via a method call.

If you are not using the RTTOV interpolator you do not need to specify the pressure levels. Instantiate the **Profiles** object with the same number of levels as the coefficient file is based on (usually 54 or 101) and the pressure profile from the coefficient file will be used by default unless you specify an array containing different pressure levels using the **setP** method (C++) or assign pressure levels to the **P** member (Python).

Once all the necessary profile data have been specified in the Profiles instance it can be associated with the **RttovSafe** or **Rttov** instance. In C++ this is done using the **myRttov.setProfiles** method. No checks are made on the the profile data before RTTOV is called so you must ensure that it conforms to the requirements of RTTOV and the wrapper interface. In Python you can simply assign the **myProfiles** object to the **myRttov.Profiles** member: in contrast to the C++ classes, **pyrttov** does carry out checks on the profile (and other) data as you assign values. For C++ only it is very important that ***all*** profile data are associated with the **Profiles** object ***before*** it is associated with the **Rttov/RttovSafe** instance.

In C++ once you have called RTTOV for the profiles it is up to you to deallocate the arrays which you associated with the **Profiles** instance using the "set" methods: these are not deallocated by the **Profiles** destructor. This is not an issue in Python as the garbage collection handles this automatically.

The following table gives the dimensions and profile variable list which should be specified in each input array. See the user guide for more information on which profile variables are used for each type of simulation (e.g. MW, IR, solar-affected, scattering, etc) Unused variables can be set to zero.

Array	**Type**	**Dimensions***	**Mandatory/ Optional**	**Variable list**
DateTimes	Integer	[nprofiles][6]	Mandatory	(year, month, day, hour, minute, second) per profile *(The full date will be used to calculate the TOA solar irradiance for solar-affected simulations. The time is not currently used by RTTOV so can be zero).*
Angles	Real	[nprofiles][4]	Mandatory	(zenangle, azangle, sunzenangle, sunazangle) per profile
SurfGeom	Real	[nprofiles][3]	Mandatory	(latitude, longitude, elevation) per profile
SurfType	Integer	[nprofiles][2]	Mandatory	(skin%surftype, skin%watertype) per profile
Skin	Real	[nprofiles][9]	Mandatory	(skin%t, skin%salinity, skin%snow_fraction, skin %foam_fraction, skin%fastem(1:5)) per profile
S2m	Real	[nprofiles][6]	Mandatory	(s2m%p, s2m%t, s2m%q, s2m%u, s2m%v, s2m%wfetc) per profile
SimpleCloud	Real	[nprofiles][2]	Optional	(ctp, cfraction) per profile
ClwScheme	Integer	[nprofiles][2]	Optional	Visible/IR (clw_scheme, clwde_param) per profile
IceCloud	Integer	[nprofiles][2]	Optional	(ice_scheme, icede_param) per profile
Zeeman	Real	[nprofiles][2]	Optional	(Be, cosbk) per profile

**For the C++ Profile class the arrays are specified for each profile separately so there is no [nprofiles] dimension. For the C++ and Python Profiles classes the data are specified for all profiles together in a single array.*

6.9. Profile data for an RttovScatt object (C++ and Python)

The **ProfilesScatt** class represents one or more profile structures for input to RTTOV-SCATT. The

atmospheric profiles and other variables are specified as a series of arrays. An instance of the **ProfilesScatt** class is then provided to the **RttovScatt** instance. The methods (C++) and members (Python) of the **ProfilesScatt** class are given in Appendix H.

The **ProfilesScatt** class is similar in many ways to the **Profiles** class so most of the description in the previous section applies here. However since **ProfilesScatt** is used specifically for MW scattering simulations, not all RTTOV profile variables are relevant, some arrays have slightly different dimensions and some additional profile variables may be specified. In particular the input cloud and hydrometeor arrays are defined on nlevels (unlike the case for visible/IR scattering where they are on nlayers) and the pressure half-levels profile has size (nlevels+1). You must always specify the pressure levels for RTTOV-SCATT: there is no option to use the optical depth coefficient levels. Similarly some of the arrays which group profile variables together are different to those in the Profiles class: the table below lists these arrays.

Concentration profiles for the five default hydrometeor types can be specified using individually named methods (e.g. **setRain**), and a single hydrometeor cloud fraction (multi_hydro_frac = false) can be specified via **setHydroFrac**. In C++, to specify separate cloud fraction profiles per hydrometeor type or to specify hydrometeor profiles with custom hydrotable files use the **setGasItem** as for the **Profiles** object. In Python you can specify arbitrary cloud fraction and cloud concentration profiles via the **HydroN** and **HydroFracN** members.

Just as for **Profiles** objects, you associate a populated **ProfilesScatt** instance with an **RttovScatt/RttovScattSafe** instance using the **setProfiles** method (C++) or by directly assigning to the **Profiles** member (Python). For C++ only it is very important that ***all*** profile data are associated with the **ProfilesScatt** object ***before*** it is associated with the **RttovScatt/RttovScattSafe** instance.

The following table gives the dimensions and profile variable list which should be specified in each input array.

Array	**Type**	**Dimensions***	**Mandatory/ Optional**	**Variable list**
DateTimes	Integer	[nprofiles][6]	Mandatory	(year, month, day, hour, minute, second) per profile *(This is not currently used in the interface and can be zero: it is included as it may be used in the future).*
Angles	Real	[nprofiles][2]	Mandatory	(zenangle, azangle) per profile
SurfGeom	Real	[nprofiles][3]	Mandatory	(latitude, longitude, elevation) per profile
SurfType	Integer	[nprofiles]	Mandatory	skin%surftype per profile
Skin	Real	[nprofiles][8]	Mandatory	(skin%t, skin%salinity, skin%foam_fraction, skin%fastem(1:5)) per profile
S2m	Real	[nprofiles][5]	Mandatory	(s2m%p, s2m%t, s2m%q, s2m%u, s2m%v) per profile
Zeeman	Real	[nprofiles][2]	Optional	(Be, cosbk) per profile

**For the C++ ProfileScatt class the arrays are specified for each profile separately so there is no [nprofiles] dimension. For the C++ and Python ProfilesScatt classes the data are specified for all profiles together in a single array.*

6.10. Specifying explicit cloud/aerosol optical properties for visible/IR scattering simulations

This section applies to visible/IR aerosol/cloud scattering simulations using "method 2" as described in sections 8.5 and 8.6 of the user guide: you should read these sections in order to understand the RTTOV scattering options and inputs.

These simulations are run using **Rttov/RttovSafe** objects (this does not apply to **RttovScatt/RttovScattSafe** objects). They are activated by setting the AddClouds or AddAerosl (or both) options to true and the corresponding UserCldOptParam or UserAerOptParam (or both) options to true.

Separate optical property inputs are available for clouds and aerosols. The optical properties are provided in the same way for both. The only difference is that for cloudy simulations you must specify a profile of cloud fractions (cfrac) in the **Profile** or **Profiles** object associated with the **Rttov/RttovSafe** object whereas this is not required for aerosols.

If aerosols are not active you do not need to specify any aerosol optical property inputs, and likewise for clouds. Also note that you can specify optical properties for clouds and use the pre-defined aerosol particle types from the coefficient file (as described above) or vice versa.

Optical properties are specified for every ***layer*** for every ***channel being simulated*** for every profile. It is important that in the arguments described below the optical properties are defined for the same channels being simulated in the call to RTTOV (see the next section).

The optical property parameters are listed in the following table.

Argument	**Type**	**Description**
asb[3][nprofiles][nchannels][nlayers]	Real	Absorption coefficients (cld_asb(1,:,:,:)), scattering coefficients (cld_asb(2,:,:,:)) and bpr parameters (cld_asb(3,:,:,:)). The absorption and scattering coefficients are required in all cases, units km^{-1}. The bpr values are only required for IR channels when Chou-scaling is used: they can be zero otherwise. See below for how to calculate bpr values.
phangle[nphangle]	Real	Angle grid on which phase functions are defined (degrees). First value must be 0° and final value must be 180°. Only required for solar-affected channels when opts%rt_ir %addsolar is true (i.e. when solar radiation is included).
pha[nprofiles][nchannels][nlayers] [nphangle]	Real	Azimuthally-averaged phase functions normalised such that the integral over all scattering angles is 4π. Phase functions are only required for solar-affected channels when opts%rt_ir %addsolar is true (i.e. when solar radiation is included).
legcoef[nprofiles][nchannels][nlayers] [nmom+1]	Real	Legendre coefficients corresponding to each phase function. Note the final dimension is nmom+1: this is consistent with the RTTOV internal structures: the "zeroth" coefficient is always 1. Legendre coefficients are only required for all channels for which the DOM solver is being used. See below for how to calculate Legendre coefficients.

The relevant methods of the **Rttov**/**RttovSafe** objects for specifying optical properties are listed in Appendix C. The only mandatory input is the asb array containing the absorption and scattering coefficients. This is assigned to the **Rttov**/**RttovSafe** object using the **setCldAsb**/**setAerAsb** methods (C++) or directly assigning to the **CldAsb**/**AerAsb** members (Python). The absorption and scattering coefficients must be supplied for all layers, channels and profiles. For any channels for which Chou-scaling is not being used the bpr values may be zero. In the case where Chou-scaling is being used and solar radiation is not included no other optical property inputs need to be specified.

If solar radiation is enabled you must specify phase functions for solar affected channels in all layers containing scattering particles. In addition the grid of angles on which the phase functions are defined must also be specified. In C++ these are set together using the **setCldPha**/**setAerPha** method. In Python the phase angles and phase functions are assigned directly to the **CldPhangle**/**AerPhangle** and **CldPha**/**AerPha** members.

If the DOM solver is being used you must specify the Legendre coeffiicients corresponding to the phase functions: this applies to all channels (not only solar-affected ones). In C++ the **setCldLegcoef/setAerLegcoef** method is used and in Python the coefficients are assigned directly to the **CldLegcoef/AerLegcoef** members. Notice that the final dimension of the Legendre coefficient array is (nmom+1). The value of nmom must equal or exceed the number of DOM streams you are using in the simulations (there is no advantage to providing *more* coefficients than this unless you are changing the number of DOM streams). For layers containing no cloud/aerosol the phase function values and Legendre coefficients can be zero.

RTTOV provides subroutines to calculate bpr values and Legendre coefficients from phase functions: this is achieved via the **calcBpr** and **calcLegcoef** methods whose interfaces are described in Appendix C. The subroutine to calculate the bpr values in particular is relatively slow and you may wish to run this off-line and store the bpr values required for your simulations. The subroutine in RTTOV is OpenMP-enabled: if you compiled RTTOV with OpenMP then the number of threads specified in the wrapper options will be used when calling **calcBpr**.

6.11. Calling RTTOV

The RTTOV direct model is run by calling the **myRttov.runDirect** method. There are two interfaces for this method: if called without arguments all channels that were loaded will be simulated. Otherwise a list of channel numbers to simulate may be supplied.

The RTTOV K (Jacobian) model is run by calling the **myRttov.runK** method. As for the direct model this can be called for all channels (no arguments) or for a subset of loaded channels (by specifying the list of channel numbers). The input perturbation is set to 1 for brightness temperatures and radiances in all channels (see the user guide for details about the K model).

NB For radar simulations, currently the zef_k input perturbation is set to 1 for all levels for all channels.

The user guide notes that most Jacobian variables/structures should be initialised to zero before calling the K model: the wrapper takes care of this.

Note that there is no difference in how you set up the input data for the direct and K models: they require the same inputs. The only difference is that after running the K model, the additional

Jacobian outputs are available.

You can specify a large number of profiles in an **Rttov/RttovSafe** instance. When RTTOV is called on the profiles, the number of profiles passed into RTTOV per call is defined in the wrapper option "nprofs_per_call" which is specified by the **setNprofsPerCal**l method of the **Options** class (C++) or the **NprofsPerCall** member of the **Options** class (Python). The total number of profiles is divided into batches of this size and RTTOV is called repeatedly by the wrapper until all profiles have been simulated. By default nprofs_per_call is 1, but it can be increased to improve performance especially if RTTOV has been compiled with OpenMP and the nthreads wrapper option is increased in order to make use of multiple threads.

6.12. Accessing RTTOV outputs

Once RTTOV has been called the output data can be accessed by calling various methods. Note that this data remains available until RTTOV is called again for the same instrument (using the **runDirect** or **runK** methods for example) at which point it is replaced with the new output.

The simulated radiances can be obtained by calling the **myRttov.getRads** method. Simulated brightness temperatures (for channels with wavelengths above 3µm) and reflectances (for other channels) can be obtained by calling the **myRttov.getBtRefl** method.

It is also possible to access the full contents of the RTTOV transmission, radiance and radiance2 structures (so long as those member arrays were output by the simulations). You must set the relevant option flag (store_trans, store_rad, store_rad2) before calling RTTOV otherwise calls to these methods (C++) or accesses to the members (Python) will throw an exception. In C++ each method returns a vector of values for a given profile index or for given profile and channel indices while in Python you can access the full output array for all channels/profiles. The relevant methods and members are listed in Appendix C.

For RTTOV-SCATT only BT outputs are available for standard (passive) simulations: in this case you can access the cloudy BTs via the **myRttov**.**getBt** method. For **RttovScatt**/**RttovScattSafe** objects the store_trans and store_rad2 options have no effect: these outputs are not produced by RTTOV-SCATT. If store_rad is set then you can access the clear-sky BTs. If store_emis_terms is set then you can also access the emissivity retrieval outputs from the RTTOV-SCATT direct model.

For RTTOV-SCATT radar simulations, the reflectivity and attenuated reflectivity are available via the **myRttov.getZef** and **myRttov.getAZef** methods.

After calling the RTTOV K model the Jacobians can be obtained through the various methods /members listed in Appendix C. For example the temperature Jacobians are obtained using the **myRttov**.**getTK** method (C++) which returns the Jacobian for a given channel and profile or simply by **myRttov**.**TK** (Python) which returns the array of Jacobians for all channels and profiles (dimensions [nprofiles][nchannels][nlevels]).

In C++, to return the Jacobians for gas profiles and (if computed) for clouds and aerosols, the **myRttov**.**getItemK** method is used. The first argument is of type **rttov**::**itemIdType**: this enumeration is defined in wrapper/rttov_common.h and a complete list of the associated constants is given in Appendix K. For example, to obtain the water vapour Jacobian for the first channel and the first profile simulated use:

```
myRttov.getItemK(rttov::Q,0,0)
```

In Python there is also a **getItemK** method, but it is easier to reference each Jacobian directly as **myRttov**.**CH4K** (CH4 Jacobian), **myRttov**.**CfracK** (cloud fraction Jacobian), **myRttov**.**CirrK** (ice cloud Jacobian), and so on.

If you run aerosol simulations using a custom *scaercoef* aerosol optical property file you can access Jacobians using the **getItemK** method as usual in C++. The Python **Rttov** class has a **getUserAerNK** method which can be used to return the Jacobians for the specified aerosol type or the Jacobians can be accessed directly via the **AerNK** members (where *N*=1,2,..., 30). As indicated above, the method/members used to access aerosol Jacobians must correspond to the way the aerosols were specified in the input profile data. For OPAC or CAMS aerosols you must use the named Jacobian members (**InsoK** etc) if the named profile members (**Inso**) were used to specify the profile data.

Note that, similar to the input profiles, the cloud and aerosol profile Jacobians will be *nlevels* in size with a zero in the final element (the first *nlayers* elements contain the Jacobian).

For RTTOV-SCATT hydrometeor and cloud fractions, you should access the Jacobians in a consistent way to that in which they were specified: for example, if you specify the hydrometeor concentrations via the named methods (e.g. **setRain**) you should use **getRainK** (C++) or **RainK** (Python). Otherwise if you used **setHydroN** or **setHydroFracN**, then you should use **getItemK** in C++ or Python, or Python also provides **getHydroNK** and **getHydroFracNK** methods, and **HydroNK** and **HydroFracNK** members (where *N*=1,2,…, 30 in the latter members).

In C++ many of the methods which return RTTOV outputs take profile and channel indexes as arguments: these are zero-counted values into the list of profiles and channels simulated. For example, to return information for the first profile the profile index should be zero, and if you simulated channels 1, 3 and 5 of an instrument, the indices for these channels in the output are 0, 1 and 2 respectively.

In contrast **pyrttov** provides access to the whole array of each output for all channels and profiles.

The additional profile variables which are active in the Jacobian model can be accessed via the **getS2mK**, **getSkinK**, **getSimpleCloudK** methods (C++) or the **S2mK**, **SkinK** and **SimpleCloudK** members (Python). The order of the variables is the same as for the corresponding input arrays.

6.13. Deallocating memory

The deallocation of memory associated with an instrument represented by an **RttovSafe** or **Rttov** object is taken care of automatically when an object is destroyed.

7. Notes on thread-safety and technical implementation

RTTOV itself (the Fortran code) is fully thread-safe.

However, currently the ***only*** supported method of running multi-threaded simulations in the RTTOV wrapper is by compiling RTTOV with OpenMP support and setting the number of threads in the wrapper **nthreads** option.

The calls to `rttov_load_inst, rttov_drop_inst, rttov_load_atlas` and `rttov_drop_atlas` are not thread-safe. This means the **loadInst** and **load*Atlas** methods of **Rttov** and **Atlas** objects are not thread-safe either, nor is destruction of **Rttov** and **Atlas** objects.

Internally, the wrapper manages each loaded instrument (and, separately, each loaded atlas) via a linked list. When a destructor is called, the object is removed from the linked list. The loading of instruments and destruction of objects is therefore not thread-safe because it can result in race conditions when updating the linked list.

Each loaded instrument in the linked list stores simulation results in a data structure. You cannot make multiple simultaneous calls to run RTTOV simulations on a single loaded instance because there will be race conditions on this data structure.

In general, code which seeks to instantiate multiple **Rttov** objects and run simulations on them simultaneously is not supported and will not run correctly.

Furthermore, you must be very careful (especially in C++ code) to avoid inadvertently calling destructors of **Rttov** objects because this causes the instrument to be unloaded and memory to be deallocated. One example of this can be if a number of **Rttov** objects are assigned to a vector: if the vector is resized internally, the **Rttov** object destructors will be called and this will render the objects unusable for further simulations without reloading the instruments. One mitigation for this particular example is to ensure the vector has enough space for all **Rttov** objects that will be stored in it before adding any objects to it.

You must also not make copies of **Rttov** or **Atlas** objects, but passing references and pointers to them are OK.

It is intended to address the issues related to object copying and destruction in a future RTTOV release. We will also seek to improve the thread-safety of the wrapper.

8. Limitations of the wrapper

The wrapper currently has the following limitations:

- Not all emissivity/BRDF atlas options and outputs are available (for example standard deviation/covariance data and quality flags cannot currently be accessed).
- Jacobians of explicit optical properties for visible/IR scattering simulations are not available via the wrapper of the RTTOV K model.
- Aerosol simulations with user-defined *scaercoef* aerosol optical property files are supported up to a maximum of 30 aerosol species.
- RTTOV-SCATT simulations for custom hydrotable files are supported up to a maximum of 30 hydrometeor types.
- PC-RTTOV unavailable.
- HTFRTC unavailable.
- TL/AD models unavailable.

Appendix A: Gas IDs

Gas ID list: these are defined in src/wrapper/rttov_wrapper_handle.F90. See user guide Annex O for more information about the profile variables and sections 8.5, 8.6 and 8.7 for information about the cloud and aerosol types.

ID	Variable	nlevels or nlayers*
1	Water vapour (q)	nlevels
2	Ozone (O3)	nlevels
3	CO2	nlevels
4	N2O	nlevels
5	CO	nlevels
6	CH4	nlevels
7	SO2	nlevels
15	Cloud liquid water (clw) – “clear-sky” MW only (*not RTTOV-SCATT)*	nlevels
20	Cloud fraction (cfrac)	nlayers
21-25	Cloud liquid water types 1-5 (STCO, STMA, CUCC, CUCP, CUMA)	nlayers
30	Ice cloud (CIRR)	nlayers
31	Ice cloud effective diameter (icede)	nlayers
32	Cloud liquid water effective diameter (clwde)	nlayers
41-53	OPAC aerosol particle types 1-13	nlayers
81-89	CAMS aerosol particle types 1-9	nlayers
101-130	User-defined aerosol particle types 1-30	nlayers
60	RTTOV-SCATT hydro_frac (cloud fraction)	nlevels
61	RTTOV-SCATT cloud liquid water (CLW)	nlevels
62	RTTOV-SCATT cloud ice water (CIW)	nlevels
63	RTTOV-SCATT rain	nlevels
64	RTTOV-SCATT snow	nlevels
65	RTTOV-SCATT graupel	nlevels
201-230	Arbitrary hydrometeor types (e.g. for custom hydrotables)	nlevels
301-330	Multiple hydrometeor cloud fractions	nlevels

*As noted above cloud and aerosol profiles are specified on layers so only the first nlayers values are used, the final element of the array (nlevels) is ignored.

Appendix B: RTTOV wrapper subroutines

The following table lists the main subroutines in the RTTOV wrapper:

Subroutine	**Description**
rttov_load_inst	Specify initial RTTOV and wrapper options and load an instrument
rttov_set_options	Modify one or more RTTOV and wrapper options
rttov_print_options	Print the current RTTOV and wrapper options
rttov_call_direct	Call the RTTOV direct model
rttov_call_k	Call the RTTOV K model
rttov_visir_scatt_call_direct	Call the RTTOV direct model for visible/IR scattering with explicit optical properties
rttov_visir_scatt_call_k	Call the RTTOV K model for visible/IR scattering with explicit optical properties
rttov_scatt_call_direct	Call the RTTOV-SCATT direct model
rttov_scatt_call_k	Call the RTTOV-SCATT K model
rttov_drop_inst	Deallocate the data for a specified instrument
rttov_drop_all	Deallocate all instrument and atlas data
rttov_load_brdf_atlas rttov_load_ir_emis_atlas rttov_load_mw_emis_atlas	Initialise the BRDF and emissivity atlases
rttov_get_emisbrdf	Return emissivity/BRDF values from a given atlas
rttov_drop_atlas	Deallocate a BRDF or emissivity atlas
rttov_bpr	Calculate bpr scattering parameter from given phase function
rttov_legcoef	Calculate Legendre coefficients from given phase function

The main subroutine calls to the direct and K models return the simulated radiances and brightness temperatures (or reflectances) as described above. RTTOV provides a number of other radiance and transmittance outputs in the transmission, radiance and secondary radiance structures. Each member of these structures can be made available (provided it was calculated by the simulation) by setting the store_trans, store_rad, store_rad2 and/or store_emis_terms wrapper options. They can be accessed via one of the subroutine calls listed below. Note that these outputs are stored independently for each instrument, but for any given instrument they are overwritten by any subsequent direct or K model calls for that instrument.

Each subroutine interface is very similar: they all return the usual error status and take the instrument ID and an array argument of the size given below. For C/C++ calls the array dimensions must also be passed, but these are implicit for Python calls as described above.

Array sizes of *nchanprof* refer to *nchannels* * *nprofiles* (i.e. the total number of channels being simulated). From C and C++ you can pass an array of shape (nprofiles, nchannels) instead of one of shape (nchanprof) if this is more convenient. From Python you can pass an array of shape

(nchannels, nprofiles). See the example code. An example call from Python is:

```
> rad_clear = numpy.empty((nchannels,nprofiles), order='F', dtype=numpy.float64)
> err = rttov_get_rad_clear(inst_id, rad_clear)
```

The following tables list the members of the RTTOV radiance, radiance2, transmission and emissivity retrieval structures returned: see Annex O in the user guide for more information about these outputs.

Radiance structure members:

Subroutine	**Array argument and dimensions in C index order**
rttov_get_rad_clear	radiance%clear(nchanprof)
rttov_get_rad_total	radiance%total(nchanprof) – this is returned in the rads argument to the rttov_call_* subroutines
rttov_get_rad_cloudy	radiance%cloudy(nchanprof)
rttov_get_bt_clear	radiance%bt_clear(nchanprof)
rttov_get_bt	radiance%bt(nchanprof) – this is returned for IR/MW channels in the btrefl argument to the rttov_call_* subroutines
rttov_get_refl_clear	radiance%refl_clear(nchanprof)
rttov_get_refl	radiance%refl(nchanprof) – this is returned for VIS/NIR channels in the btrefl argument to the rttov_call_* subroutines
rttov_get_overcast	radiance%overcast(nchanprof, nlayers)
rttov_get_plane_parallel	radiance%plane_parallel – this is a scalar 0/1 (false/true)
rttov_get_rad_quality	radiance%quality(nchanprof) – integer array
rttov_get_geometric_height	radiance%geometric_height(nchanprof, nlevels)

Radiance2 structure members:

Subroutine	**Array argument and dimensions in C index order**
rttov_get_rad2_up	radiance2%up(nchanprof, nlayers)
rttov_get_rad2_down	radiance2%down(nchanprof, nlayers)
rttov_get_rad2_surf	radiance2%surf(nchanprof, nlayers)
rttov_get_rad2_upclear	radiance2%upclear(nchanprof)
rttov_get_rad2_dnclear	radiance2%dnclear(nchanprof)
rttov_get_rad2_refldnclear	radiance2%refldnclear(nchanprof)

Transmission structure members:

Subroutine	**Array argument and dimensions in C index order**
rttov_get_tau_total	transmission%tau_total(nchanprof)
rttov_get_tau_levels	transmission%tau_levels(nchanprof, nlevels)
rttov_get_tausun_total_path2	transmission%tausun_total_path2(nchanprof)
rttov_get_tausun_levels_path2	transmission%tausun_levels_path2(nchanprof, nlevels)
rttov_get_tausun_total_path1	transmission%tausun_total_path1(nchanprof)
rttov_get_tausun_levels_path1	transmission%tausun_levels_path1(nchanprof, nlevels)
rttov_get_tau_total_cld	transmission%tau_total_cld(nchanprof)
rttov_get_tau_levels_cld	transmission%tau_levels_cld(nchanprof, nlevels)

RTTOV-SCATT emissivity retrieval structure members:

Subroutine	**Array argument and dimensions in C index order**
rttov_get_emis_terms_cfrac	emis_terms%cfrac(nchanprof)
rttov_get_emis_terms_bsfc	emis_terms%bsfc(nchanprof, nlevels)
rttov_get_emis_terms_tau_cld	emis_terms%tau_cld(nchanprof)
rttov_get_emis_terms_up_cld	emis_terms%up_cld(nchanprof)
rttov_get_emis_terms_down_cld	emis_terms%down_cld(nchanprof)
rttov_get_emis_terms_tau_clr	emis_terms%tau_clr(nchanprof)
rttov_get_emis_terms_up_clr	emis_terms%up_clr(nchanprof)
rttov_get_emis_terms_down_clr	emis_terms%down_clr(nchanprof)

RTTOV-SCATT reflectivity structure members:

Subroutine	**Array argument and dimensions in C index order**
rttov_get_zef	reflectivity%zef(nchanprof, nlevels)
rttov_get_azef	reflectivity%azef(nchanprof, nlevels)

Appendix C: *RttovSafe* and *Rttov* classes (C++ and Python)

C++ RttovSafe and Rttov classes

The majority of the methods used for calling RTTOV are the same for both the **RttovSafe** and **Rttov** classes. The only one which differs is the method for associating profile data with the **RttovSafe** or **Rttov** instance.

Constructors:

RttovSafe ()
__RttovSafe__ class constructor method.

Rttov ()
__Rttov__ class constructor method.

Associating profile data with an *RttovSafe* object:

void **setTheProfiles** (std::vector< **rttov::Profile** > &theProfiles)
Associate a vector of __Profile__ objects with this __RttovSafe__ object; carries out checks on profiles before calling RTTOV to help prevent errors: all profiles must be have the same number of levels with the same content (gases, clouds, aerosols) and have the same gas_units.

Associating profile data with an *Rttov* object:

void **setProfiles** (**rttov::Profiles *profiles**)
Associate a __Profiles__ object with this __Rttov__ object; this is fast, but does not carry out any checks on profiles before calling RTTOV.

Methods common to *RttovSafe* and *Rttov* classes:

const string & **getFileCoef** () const
Return the coefficient filename.

const string & **getFileSccld** () const
Return the cloud coefficient filename.

const string & **getFileScaer** () const
Return the aerosol coefficient filename.

const string & **getFileMfasisCld** () const
Return the MFASIS cloud LUT filename.

const string & **getFileMfasisNN**() const
Return the MFASIS-NN coefficient filename.

void **setFileCoef** (const string &fileCoef)
Set the coefficient filename.

void **setFileSccld** (const string &fileSccld)
Set the cloud coefficient filename.

void **setFileScaer** (const string &fileScaer)

Set the aerosol coefficient filename.

void **setFileMfasisCld** (const string &fileMfasisCld)

Set the MFASIS cloud LUT filename.

void **setFileMfasisNN** (const string &fileMfasisNN)

Set the MFASIS-NN coefficient filename.

void **loadInst** ()

Load instrument with all channels.

void **loadInst** (const vector< int > &channels)

*Load instrument for a list of channels; the method **setFileCoef()** must have been called previously.*

int **getInstId** () const

Return the inst_id.

bool **isCoeffsLoaded** () const

Return true if instrument is loaded.

int **getNchannels** () const

Return the number of loaded channels.

double * **getRefPressures** ()

Return the pressure levels of the coefficient file.

int **getCoeffsNlevels** ()

Return the number of levels of the coefficient file.

double * **getWaveNumbers** ()

Return the channel central wavenumbers of the coefficient file.

bool **isProfileSet** () const

Return true if profiles have been associated.

int **getNprofiles** () const

Return the number of associated profiles.

void **updateOptions** ()

Update RTTOV options for the currently loaded instrument.

void **printOptions** ()

Print RTTOV options for the currently loaded instrument.

void **setSurfEmisRefl** (double ***surfemisrefl**)

Set pointer to array containing input/output surface emissivity, reflectance and specularity values; this must be previously allocated a double array of dimensions [5][nprofiles][nchannels]; this is used to pass emissivity/reflectance/specularity/effective Tskin values into RTTOV; if this is not called the Rttov object will allocate an array containing the values used by RTTOV which can be accessed by getSurfEmisRefl.

void **setAerAsb** (double *asb)

Set the aerosol absorption coefs, scattering coefs and bpr parameters.

void **setAerPha** (int nphangle, double *phangle, double *pha)

Set the aerosol phase functions.

void **setAerLegcoef** (int nmom, double *legcoef)

Set the aerosol phase function Legendre coefficients.

void **setCldAsb** (double *asb)

Set the cloud absorption coefs, scattering coefs and bpr parameters.

void **setCldPha** (int nphangle, double *phangle, double *pha)

Set the cloud phase functions.

void **setCldLegcoef** (int nmom, double *legcoef)

Set the cloud phase function Legendre coefficients.

void **printGases** ()

Print gases array contents on standard output.

void **runDirect** ()

Run the RTTOV direct model for all channels.

void **runDirect** (const vector< int > &channels)

Run the RTTOV direct model for a list of channels.

void **runK** ()

Run the RTTOV K model for all channels.

void **runK** (const vector< int > &channels)

Run the RTTOV K model for a list of channels.

const double * **getBtRefl** () const

*Return a pointer to an array of dimensions [nprofiles][nchannels] filled with computed brightness temperatures and reflectances by the previous run; this array is allocated by the **Rttov** object and is destroyed when a new run is performed or if the instance is destroyed.*

const double * **getRads** () const

*Return a pointer to an array of dimensions [nprofiles][nchannels] filled with computed radiances by the previous run; this array is allocated by the **Rttov** object and is destroyed when a new run is performed or if the instance is destroyed.*

std::vector< double > **getBtRefl** (const int profile)

Return vector of brightness temperatures/reflectances computed by the previous run for the given profile number.

std::vector< double > **getRads** (const int profile)

Return a vector of radiances computed by the previous run for the given profile number.

const double * **getSurfEmisRefl** () const

*Return a pointer to an array of dimensions [5][nprofiles][nchannels] containing output values of surface emissivity, reflectance, specularity, and effective Tskin; this array can be initialised by the user and set by calling the setSurfEmisRefl method; alternatively if the emissivity/reflectance array is allocated by the **Rttov** object it is deleted at the next run or when the **Rttov** instance is destroyed.*

int **getAerNphangle** () const

Return the number of aerosol phase function angles.

int **getAerNmom** () const

Return the number of aerosol phase function Legendre coefficients.

const double * **getAerAsb** () const

Return the aerosol absorption coefs, scattering coefs and bpr parameters.

const double * **getAerPhangle** () const

Return the aerosol phase function angles.

const double * **getAerLegcoef** () const

Return the aerosol phase function Legendre coefficients.

const double * **getAerPha** () const
Return the aerosol phase functions.

int **getCldNphangle** () const
Return the number of cloud phase function angles.

int **getCldNmom** () const
Return the number of cloud phase function Legendre coefficients.

const double * **getCldAsb** () const
Return the cloud absorption coefs, scattering coefs and bpr parameters.

const double * **getCldPhangle** () const
Return the cloud phase function angles.

const double * **getCldLegcoef** () const
Return the cloudphase function Legendre coefficients.

const double * **getCldPha** () const
Return the cloud phase functions.

double **calcBpr** (int nphangle, double *phangle, double *pha)
Calculate bpr parameter for given phase function.

void **calcLegcoef** (int nphangle, double *phangle, double *pha, int nmom, double *legcoef, int ngauss)
Calculate Legendre coefficients for given phase function.

std::vector< double > **getPK** (int profile, int channel)
Return the computed pressure Jacobians for a given profile and channel.

std::vector< double > **getTK** (int profile, int channel)
Return computed temperature Jacobians for a given profile and channel.

std::vector< double > **getSkinK** (int profile, int channel)
Return computed skin variable Jacobians for a given profile and channel.

std::vector< double > **getS2mK** (int profile, int channel)
Return computed 2m variable Jacobian for a given profile and channel.

std::vector< double > **getSimpleCloudK** (int profile, int channel)
Return computed simple cloud variable Jacobians for a given profile and channel.

std::vector< double > **getItemK** (rttov::itemIdType, int profile, int channel)
Return computed gas, cloud and aerosol Jacobian values for a given profile and channel.

std::vector< double > **getSurfEmisK** (int profile)
Return computed surface emissivity Jacobians for a given profile.

std::vector< double > **getSurfReflK** (int profile)
Return computed surface BRDF Jacobians for a given profile.

std::vector< double > **getSurfDiffuseReflK** (int profile)
Return computed surface diffuse reflectance Jacobians for a given profile.

std::vector< double > **getSpecularityK** (int profile)
Return computed surface specularity Jacobians for a given profile.

std::vector< double > **getTskinEffK** (int profile)
Return computed effective Tskin Jacobians for a given profile.

std::vector< double > **getTauTotal** (int profile)

Return RTTOV transmission tau_total output array of size [nchannels] for given profile, requires store_trans true.

std::vector< double > **getTauLevels** (int profile, int channel)

Return RTTOV transmission tau_levels output array of size [nlevels] for given profile and channel, requires store_trans true.

std::vector< double > **getTauSunTotalPath1** (int profile)

Return RTTOV transmission tausun_total_path1 output array of size [nchannels] for given profile, requires store_trans true.

std::vector< double > **getTauSunLevelsPath1** (int profile, int channel)

Return RTTOV transmission tausun_levels_path1 output array of size [nlevels] for given profile and channel, requires store_trans true.

std::vector< double > **getTauSunTotalPath2** (int profile)

Return RTTOV transmission tausun_total_path2 output array of size [nchannels] for given profile, requires store_trans true.

std::vector< double > **getTauSunLevelsPath2** (int profile, int channel)

Return RTTOV transmission tausun_levels_path2 output array of size [nlevels] for given profile and channel, requires store_trans true.

std::vector< double > **getTauTotalCld** (int profile)

Return RTTOV transmission tau_total_cld output array of size [nchannels] for given profile, requires store_trans true.

std::vector< double > **getTauLevelsCld** (int profile, int channel)

Return RTTOV transmission tau_levels_cld output array of size [nlevels] for given profile and channel, requires store_trans true.

std::vector< double > **getRadClear** (int profile)

Return RTTOV radiance clear output array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getRadTotal** (int profile)

Return RTTOV radiance total output array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getBtClear** (int profile)

Return RTTOV radiance bt_clear output array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getBt** (int profile)

Return RTTOV radiance bt output array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getReflClear** (int profile)

Return RTTOV radiance refl_clear output array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getRefl** (int profile)

Return RTTOV radiance refl output array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getRadCloudy** (int profile)

Return RTTOV radiance cloudy output array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getOvercast** (int profile, int channel)

Return RTTOV radiance overcast output array of size [nlayers] for given profile and channel, requires store_rad true.

std::vector< int > **getRadQuality** (int profile)

Return RTTOV radiance quality flag array of size [nchannels] for given profile, requires store_rad true.

bool **getPlaneParallel** ()

Return RTTOV radiance plane_parallel flag, requires store_rad true.

std::vector< double > **getGeometricHeight** (int profile, int channel)

Return RTTOV radiance geometric_height output array of size [nlevels] for given profile and channel, requires store_rad true.

std::vector< double > **getRad2UpClear** (int profile)

Return RTTOV radiance2 upclear output array of size [nchannels] for given profile, requires store_rad2 true.

std::vector< double > **getRad2DnClear** (int profile)

Return RTTOV radiance2 dnclear output array of size [nchannels] for given profile, requires store_rad2 true.

std::vector< double > **getRad2ReflDnClear** (int profile)

Return RTTOV radiance2 refldnclear output array of size [nchannels] for given profile, requires store_rad2 true.

std::vector< double > **getRad2Up** (int profile, int channel)

Return RTTOV radiance2 up output array of size [nlayers] for given profile and channel, requires store_rad2 true.

std::vector< double > **getRad2Down** (int profile, int channel)

Return RTTOV radiance2 down output array of size [nlayers] for given profile and channel, requires store_rad2 true.

std::vector< double > **getRad2Surf** (int profile, int channel)

Return RTTOV radiance2 surf output array of size [nlayers] for given profile and channel, requires store_rad2 true.

Python Rttov class

Methods:

Rttov ()

Rttov *class constructor method.*

loadInst (channels=None)

Load instrument for a list of channels if array of channel numbers is supplied or for all channels if channels argument is omitted; the FileCoef member must have been set previously. Throws an exception if an error is encountered.

updateOptions ()

Update RTTOV options for the currently loaded instrument. Throws an exception if an error is encountered.

printOptions ()

Print RTTOV options for the currently loaded instrument. Throws an exception if an error is encountered.

runDirect (channels=None)
Run the RTTOV direct model for the supplied list of channels or for all loaded channels if the channels argument is omitted. Throws an exception if an error is encountered.

runK (channels=None)
Run the RTTOV K model for the supplied list of channels or for all loaded channels if the channels argument is omitted. Throws an exception if an error is encountered.

float array **getItemK** (gas_id)
Return computed gas, cloud and aerosol Jacobian values. See Appendix A for the gas IDs. If the requested Jacobian was not calculated this returns None, otherwise the result will be an array with dimensions [nprofiles][nchannels][nlevels]. It is also possible to access each gas, cloud or aerosol variable's Jacobians directly (see members below).

float array **getUserAerNK** (n)
Return computed Jacobian for user-defined aerosol species n (1<=n<=30). If the requested Jacobian was not calculated this returns None, otherwise the result will be an array with dimensions [nprofiles][nchannels][nlevels].

float **calcBpr** (phangle, pha)
Calculate bpr parameter for given phase function pha defined on angles phangle.

float array **calcLegcoef** (phangle, pha, nmom, ngauss=0)
Calculate Legendre coefficients for given phase function pha defined on angles phangle. Returns an array of size (nmom+1). If ngauss >= nmom, then ngauss will determine the size of the Gaussian quadrature used in the calculation.

Members:

Options **Options**
*The **Options** instance associated with this **Rttov** object. You should set the options associated with this instrument by assigning to the members of this **Options** instance.*

Profiles **Profiles**
*The **Profiles** instance associated with this **Rttov** object; you should declare an instance of **Profiles**, populate it with profile data and assign it to this member.*

string **FileCoef**
Set the coefficient filename.

string **FileSccld**
Set the cloud coefficient filename.

string **FileScaer**
Set the aerosol coefficient filename.

string **FileMfasisCld**
Set the MFASIS cloud LUT filename.

string **FileMfasisNN**
Set the MFASIS-NN coefficient filename.

bool **CoeffsLoaded**
True if instrument is loaded (read-only).

int **Nchannels**

The number of loaded channels (read-only).

int **CoeffsNlevels**
The number of levels of the coefficient file (read-only).

float array **SurfEmisRefl**
Array containing input/output surface emissivity and reflectance values of dimensions [5][nprofiles][nchannels]; this is used to pass emissivity/reflectance/specularity/effective Tskin values into RTTOV; if this is not specified before calling RTTOV the ***Rttov*** *object will create one with all elements set negative (i.e. with calcemis and calcrefl set to true) which will contain the emissivity/reflectance values used by RTTOV after it has been called.*

float array **AerAsb**
The aerosol absorption coefs, scattering coefs and bpr parameters. Dimensions are [3][nprofiles][nchannels][nlayers].

float array **AerPhangle**
The aerosol phase function angles. Dimensions are [aer_nphangle].

float array **AerPha**
The aerosol phase functions. Dimensions are [nprofiles][nchannels][nlayers][aer_nphangle].

float array **AerLegcoef**
The aerosol phase function Legendre coefficients. Dimensions are [nprofiles][nchannels][nlayers][aer_nmom+1].

float array **CldAsb**
The cloud absorption coefs, scattering coefs and bpr parameters. Dimensions are [3][nprofiles][nchannels][nlayers].

float array **CldPhangle**
The cloud phase function angles. Dimensions are [cld_nphangle].

float array **CldPha**
The cloud phase functions. Dimensions are [nprofiles][nchannels][nlayers][cld_nphangle].

float array **CldLegcoef**
The cloud phase function Legendre coefficients. Dimensions are [nprofiles][nchannels][nlayers][cld_nmom+1].

float array **BtRefl**
Brightness temperatures/reflectances computed by the previous run, dimensions [nprofiles][nchannels].

float array **Rads**
Radiances computed by the previous run, dimensions [nprofiles][nchannels].

float array **PK**
Computed pressure Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **TK**
Computed temperature Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **QK**
Computed q Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **O3K**
Computed o3 Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CO2K**

Computed co2 Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **COK**
Computed co Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **N2OK**
Computed n2o Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CH4K**
Computed ch4 Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **SO2K**
Computed so2 Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CLWK**
Computed clw Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CfracK**
Computed cfrac Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **StcoK**
Computed stco (cloud type 1) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **StmaK**
Computed stma (cloud type 2) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CuccK**
Computed cucc (cloud type 3) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CucpK**
Computed cucp (cloud type 4) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CumaK**
Computed cuma (cloud type 5) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CirrK**
Computed cirr (cloud type 6) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **IcedeK**
Computed icede Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **ClwdeK**
Computed clwde Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **InsoK**
Computed inso (aerosol type 1) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **WasoK**
Computed waso (aerosol type 2) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **SootK**
Computed soot (aerosol type 3) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **SsamK**
Computed ssam (aerosol type 4) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **SscmK**
Computed sscm (aerosol type 5) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **MinmK**
Computed minm (aerosol type 6) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **MiamK**
Computed miam (aerosol type 7) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **MicmK**
Computed micm (aerosol type 8) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **MitrK**
Computed mitr (aerosol type 9) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **SusoK**
Computed suso (aerosol type 10) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **VolaK**
Computed vola (aerosol type 11) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **VapoK**
Computed vapo (aerosol type 12) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **AsduK**
Computed asdu (aerosol type 13) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **BcarK**
Computed bcar (aerosol type 1) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **Dus1K**
Computed dus1 (aerosol type 2) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **Dus2K**
Computed dus2 (aerosol type 3) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **Dus3K**
Computed dus3 (aerosol type 4) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **SulpK**
Computed sulp (aerosol type 5) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **Ssa1K**
Computed ssa1 (aerosol type 6) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **Ssa2K**
Computed ssa2 (aerosol type 7) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **Ssa3K**
Computed ssa3 (aerosol type 8) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **OmatK**
Computed omat (aerosol type 9) Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **AerNK** *where N=1, 2, …, 30*
Computed Jacobians for user-defined aerosol species N, dimensions [nprofiles][nchannels][nlevels].

float array **SkinK**
Computed skin variable Jacobians, dimensions [nprofiles][nchannels][9].

float array **S2mK**
Computed 2m variable Jacobian, dimensions [nprofiles][nchannels][6].

float array **SimpleCloudK**
Computed simple cloud variable Jacobians, dimensions [nprofiles][nchannels][2].

float array **SurfEmisK**
Computed surface emissivity Jacobians, dimensions [nprofiles][nchannels].

float array **SurfReflK**
Computed surface BRDF Jacobians, dimensions [nprofiles][nchannels].

float array **SurfDiffuseReflK**
Computed surface diffuse reflectance Jacobians, dimensions [nprofiles][nchannels].

float array **SpecularityK**
Computed surface specularity Jacobians, dimensions [nprofiles][nchannels].

float array **TskinEffK**
Computed effective Tskin Jacobians, dimensions [nprofiles][nchannels].

float array **TauTotal**
RTTOV transmission tau_total output array, dimensions [nprofiles][nchannels], requires store_trans true.

float array **TauLevels**
RTTOV transmission tau_levels output array, dimensions [nprofiles][nchannels][nlevels], requires store_trans true.

float array **TauSunTotalPath1**
RTTOV transmission tausun_total_path1 output array, dimensions [nprofiles][nchannels], requires store_trans true.

float array **TauSunLevelsPath1**
RTTOV transmission tausun_levels_path1 output array dimensions [nprofiles][nchannels][nlevels], requires store_trans true.

float array **TauSunTotalPath2**
RTTOV transmission tausun_total_path2 output array, dimensions [nprofiles][nchannels], requires store_trans true.

float array **TauSunLevelsPath2**
RTTOV transmission tausun_levels_path2 output array dimensions [nprofiles][nchannels][nlevels], requires store_trans true.

float array **TauTotalCld**
RTTOV transmission tau_total_cld output array, dimensions [nprofiles][nchannels], requires store_trans true.

float array **TauLevelsCld**
RTTOV transmission tau_levels_cld output array, dimensions [nprofiles][nchannels][nlevels], requires store_trans true.

float array **RadClear**
RTTOV radiance clear output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **RadTotal**
RTTOV radiance total output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **BtClear**
RTTOV radiance bt_clear output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **Bt**
RTTOV radiance bt output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **ReflClear**
RTTOV radiance refl_clear output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **Refl**
RTTOV radiance refl output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **RadCloudy**
RTTOV radiance cloudy output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **Overcast**
RTTOV radiance overcast output array, dimensions [nprofiles][nchannels][nlayers], requires store_rad true.

int array **RadQuality**
RTTOV radiance quality flag array of size [nprofiles][nchannels], requires store_rad true.

bool **PlaneParallel** ()
RTTOV radiance plane_parallel flag, requires store_rad true.

float array **GeometricHeight**
RTTOV radiance geometric_height output array, dimensions [nprofiles][nchannels][nlevels], requires store_rad true.

float array **Rad2UpClear**
RTTOV radiance2 upclear output array, dimensions [nprofiles][nchannels], requires store_rad2 true.

float array **Rad2DnClear**
RTTOV radiance2 dnclear output array, dimensions [nprofiles][nchannels], requires store_rad2 true.

float array **Rad2ReflDnClear**
RTTOV radiance2 refldnclear output array, dimensions [nprofiles][nchannels], requires store_rad2 true.

float array **Rad2Up**
RTTOV radiance2 up output array, dimensions [nprofiles][nchannels][nlayers], requires store_rad2 true.

float array **Rad2Down**
RTTOV radiance2 down output array, dimensions [nprofiles][nchannels][nlayers], requires store_rad2 true.

float array **Rad2Surf**
RTTOV radiance2 surf output array, dimensions [nprofiles][nchannels][nlayers], requires store_rad2 true.

Appendix D: *RttovScattSafe* and *RttovScatt* classes (C++ and Python)

C++ RttovScattSafe and RttovScatt classes

The majority of the methods used for calling RTTOV are the same for both the **RttovScattSafe** and **RttovScatt** classes. The only one which differs is the method for associating profile data with the **RttovScattSafe** or **RttovScatt** instance.

Constructors:

RttovScattSafe ()
__RttovScattSafe__ class constructor method.

RttovScatt ()
__RttovScatt__ class constructor method.

Associating profile data with an *RttovScattSafe* object:

void **setTheProfiles** (std::vector< **rttov::ProfileScatt** > &theProfiles)
Associate a vector of __ProfileScatt__ objects with this __RttovScattSafe__ object; carries out checks on profiles before calling RTTOV to help prevent errors: all profiles must be have the same number of levels with the same content (gases, hydrometeors) and have the same gas_units.

Associating profile data with an *RttovScatt* object:

void **setProfiles** (**rttov::ProfilesScatt *profiles**)
Associate a __ProfilesScatt__ object with this __RttovScatt__ object; this is fast, but does not carry out any checks on profiles before calling RTTOV.

Methods common to *RttovScattSafe* and *RttovScatt* classes:

const string & **getFileCoef** () const
Return the coefficient filename.

const string & **getFileHydrotable** () const
Return the hydrotable filename.

const string & **getFilePol** () const
Return the ARO-scaling polarisation look-up table filename.

void **setFileCoef** (const string &fileCoef)
Set the coefficient filename.

void **setFileHydrotable** (const string &fileHydrotable)
Set the hydrotable filename.

void **setFilePol** (const string &filePol)
Set the ARO-scaling polarisation look-up table filename.

bool **isCalcZef** () const

Return true if radar simulations are enabled.

void **setCalcZef** (bool calcZef)

Enable/disable radar simulations.

bool **isMultiHydroFrac** () const

Return true if separate hydrometeor cloud fractions are enabled.

void **setMultiHydroFrac** (bool multiHydroFrac)

Enable multiple individual (true) or single (false) hydrometeor cloud fractions.

void **loadInst** ()

*Load instrument with all channels the methods **setFileCoef()** and **setFileHydrotable()** must have been called previously.*

int **getInstId** () const

Return the inst_id.

bool **isCoeffsLoaded** () const

Return true if instrument is loaded.

int **getNchannels** () const

Return the number of loaded channels.

int **getCoeffsNlevels** ()

Return the number of levels of the coefficient file.

double * **getWaveNumbers** ()

Return the channel central wavenumbers of the coefficient file.

bool **isProfileSet** () const

Return true if profiles have been associated.

int **getNprofiles** () const

Return the number of associated profiles.

void **updateOptions** ()

Update RTTOV options for the currently loaded instrument.

void **printOptions** ()

Print RTTOV options for the currently loaded instrument.

void **setSurfEmis** (double ***surfemis**)

Set pointer to array containing input/output surface emissivity/effective Tskin values; this must be previously allocated a double array of dimensions [2][nprofiles][nchannels]; this is used to pass emissivity and effective Tskin values into RTTOV-SCATT; if this is not called the RttovScatt object will allocate an array containing the values used by RTTOV-SCATT which can be accessed by getSurfEmis.

void **printGases** ()

Print gases array contents on standard output.

void **runDirect** ()

Run the RTTOV-SCATT direct model for all channels.

void **runDirect** (const vector< int > &channels)

Run the RTTOV-SCATT direct model for a list of channels.

void **runK** ()

Run the RTTOV-SCATT K model for all channels.

void **runK** (const vector< int > &channels)

Run the RTTOV-SCATT K model for a list of channels.

const double * **getBt** () const

*Return a pointer to an array of dimensions [nprofiles][nchannels] filled with computed brightness temperatures by the previous run; this array is allocated by the **RttovScatt** object and is destroyed when a new run is performed or if the instance is destroyed.*

std::vector< double > **getBt** (const int profile)

Return vector of brightness temperatures computed by the previous run for the given profile number.

const double * **getSurfEmis** () const

*Return a pointer to an array of dimensions [2][nprofiles][nchannels] containing output values of surface emissivity and effective Tskin; this array can be initialised by the user and set by calling the setSurfEmis method; alternatively if the emissivity array is allocated by the **RttovScatt** object it is deleted at the next run or when the **RttovScatt** instance is destroyed.*

std::vector< double > **getPK** (int profile, int channel)

Return the computed pressure Jacobians for a given profile and channel.

std::vector< double > **getPhK** (int profile, int channel)

Return the computed pressure half-level Jacobians for a given profile and channel.

std::vector< double > **getTK** (int profile, int channel)

Return computed temperature Jacobians for a given profile and channel.

std::vector< double > **getUserCfracK** (int profile)

Return vector of user cloud fraction Jacobians for a given profile.

std::vector< double > **getSkinK** (int profile, int channel)

Return computed skin variable Jacobians for a given profile and channel.

std::vector< double > **getS2mK** (int profile, int channel)

Return computed 2m variable Jacobian for a given profile and channel.

std::vector< double > **getItemK** (rttov::itemIdType, int profile, int channel)

Return computed gas and hydrometeor Jacobian values for a given profile and channel.

std::vector< double > **getSurfEmisK** (int profile)

Return computed surface emissivity Jacobians for a given profile.

std::vector< double > **getTskinEffK** (int profile)

Return computed effective Tskin Jacobians for a given profile.

std::vector< double > **getBtClear** (int profile)

Return RTTOV radiance bt_clear output array of size [nchannels] for given profile, requires store_rad true.

std::vector< int > **getRadQuality** (int profile)

Return RTTOV radiance quality flag array of size [nchannels] for given profile, requires store_rad true.

std::vector< double > **getGeometricHeight** (int profile, int channel)

Return RTTOV radiance geometric_height output array of size [nlevels] for given profile and channel, requires store_rad true.

std::vector< double > **getZef** (int profile, int channel)

Return RTTOV reflectivity zef output array of size [nlevels] for given profile and channel for radar simulations.

std::vector< double > **getAZef** (int profile, int channel)

Return RTTOV reflectivity azef output array of size [nlevels] for given profile and channel for radar simulations.

std::vector< double > **getEmisTermsCfrac** (int profile)
Return RTTOV-SCATT emis retrieval cfrac output array of size [nchannels] for given profile, requires store_emis_terms true.

std::vector< double > **getEmisTermsBsfc** (int profile)
Return RTTOV-SCATT emis retrieval bsfc output array of size [nchannels] for given profile, requires store_emis_terms true.

std::vector< double > **getEmisTermsTauCld** (int profile)
Return RTTOV-SCATT emis retrieval tau_cld output array of size [nchannels] for given profile, requires store_emis_terms true.

std::vector< double > **getEmisTermsUpCld** (int profile)
Return RTTOV-SCATT emis retrieval up_cld output array of size [nchannels] for given profile, requires store_emis_terms true.

std::vector< double > **getEmisTermsDownCld** (int profile)
Return RTTOV-SCATT emis retrieval down_cld output array of size [nchannels] for given profile, requires store_emis_terms true.

std::vector< double > **getEmisTermsTauClr** (int profile)
Return RTTOV-SCATT emis retrieval tau_clr output array of size [nchannels] for given profile, requires store_emis_terms true.

std::vector< double > **getEmisTermsUpClr** (int profile)
Return RTTOV-SCATT emis retrieval up_clr output array of size [nchannels] for given profile, requires store_emis_terms true.

std::vector< double > **getEmisTermsDownClr** (int profile)
Return RTTOV-SCATT emis retrieval down_clr output array of size [nchannels] for given profile, requires store_emis_terms true.

Python RttovScatt class

Methods:

RttovScatt ()
RttovScatt *class constructor method.*

loadInst ()
Load instrument: all channels must be loaded for RTTOV-SCATT; the FileCoef and FileHydrotable members must have been set previously. Throws an exception if an error is encountered.

updateOptions ()
Update RTTOV options for the currently loaded instrument. Throws an exception if an error is encountered.

runDirect (channels=None)
Run the RTTOV-SCATT direct model for the supplied list of channels or for all loaded channels if the channels argument is omitted. Throws an exception if an error is encountered.

runK (channels=None)
Run the RTTOV-SCATT K model for the supplied list of channels or for all loaded channels if the

channels argument is omitted. Throws an exception if an error is encountered.

getItemK (gas_id)
Return computed gas, cloud and aerosol Jacobian values. See Appendix A for the gas IDs. If the requested Jacobian was not calculated this returns None, otherwise the result will be an array with dimensions [nprofiles][nchannels][nlevels]. It is also possible to access each gas, cloud or aerosol variable's Jacobians directly (see members below).

float array **getHydroNK** (n)
Return computed Jacobian for hydrometeor type n (1<=n<=30). If the requested Jacobian was not calculated this returns None, otherwise the result will be an array with dimensions [nprofiles][nchannels][nlevels].

float array **getHydroFracNK** (n)
Return computed Jacobian for cloud fraction for hydrometeor type n (1<=n<=30). If the requested Jacobian was not calculated this returns None, otherwise the result will be an array with dimensions [nprofiles][nchannels][nlevels].

Members:

Options **Options**
*The **Options** instance associated with this **RttovScatt** object. You should set the RTTOV-SCATT options associated with this instrument by assigning to the members of this **Options** instance.*

ProfilesScatt **Profiles**
*The **ProfilesScatt** instance associated with this **RttovScatt** object; you should declare an instance of **ProfilesScatt**, populate it with profile data and assign it to this member.*

string **FileCoef**
The coefficient filename.

string **FileHydrotable**
The hydrotable filename.

string **FilePol**
The ARO-scaling polarisation look-up table filename.

bool **CoeffsLoaded**
True if instrument is loaded (read-only).

int **Nchannels**
The number of loaded channels (read-only).

int **CoeffsNlevels**
The number of levels of the coefficient file (read-only).

float array **SurfEmis**
*Array containing input/output surface emissivity/effective Tskin values of dimensions [2][nprofiles][nchannels]; this is used to pass emissivity and effective Tskin values into RTTOV-SCATT; if this is not specified before calling RTTOV-SCATT the **RttovScatt** object will create one with all elements set negative (i.e. with calcemis set to true) which will contain the values used by RTTOV-SCATT after it has been called.*

float array **Bt**
Brightness temperatures computed by the previous run, dimensions [nprofiles][nchannels].

float array **PK**

Computed pressure Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **PhK**
Computed pressure helf-level Jacobians, dimensions [nprofiles][nchannels][nlevels+1].

float array **TK**
Computed temperature Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **QK**
Computed q Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **O3K**
Computed o3 Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **UserCfracK**
Computed user cfrac Jacobians, dimensions [nprofiles][nchannels].

float array **CcK**
Computed cloud cover Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **ClwK**
Computed cloud liquid water Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **CiwK**
Computed cloud ice water Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **RainK**
Computed rain Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **SnowK**
Computed snow Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **GraupelK**
Computed graupel Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **HydroNK** *where N=1, 2, ..., 30*
Computed Jacobians for hydrometeor type N, dimensions [nprofiles][nchannels][nlevels].

float array **HydroFracNK** *where N=1, 2, ..., 30*
Computed Jacobians for cloud fraction for hydrometeor type N, dimensions [nprofiles][nchannels][nlevels].

float array **SkinK**
Computed skin variable Jacobians, dimensions [nprofiles][nchannels][8].

float array **S2mK**
Computed 2m variable Jacobian, dimensions [nprofiles][nchannels][5].

float array **SurfEmisK**
Computed surface emissivity Jacobians, dimensions [nprofiles][nchannels].

float array **TskinEffK**
Computed effective Tskin Jacobians, dimensions [nprofiles][nchannels].

float array **BtClear**
RTTOV radiance bt_clear output array, dimensions [nprofiles][nchannels], requires store_rad true.

int array **RadQuality**
RTTOV radiance quality output array, dimensions [nprofiles][nchannels], requires store_rad true.

float array **GeometricHeight**
RTTOV radiance geometric_height output array, dimensions [nprofiles][nchannels][nlevels], requires store_rad true.

float array **EmisTermsCfrac**
RTTOV-SCATT emis retrieval cfrac output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsBsfc**
RTTOV-SCATT emis retrieval bsfc output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsTauCld**
RTTOV-SCATT emis retrieval tau_cld output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsUpCld**
RTTOV-SCATT emis retrieval up_cld output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsDownCld**
RTTOV-SCATT emis retrieval down_cld output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsTauClr**
RTTOV-SCATT emis retrieval tau_clr output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsUpClr**
RTTOV-SCATT emis retrieval up_clr output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsDownClr**
RTTOV-SCATT emis retrieval down_clr output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

Appendix E: *Profile* class (used with *RttovSafe* objects; C++ only)

Typically a vector of instances of this class is created, the profile data are assigned to each instance and then the vector is associated with one or more **RttovSafe** instances.

Profile (int nlevels)
Constructor method.

void **setGasUnits** (**rttov::gasUnitType** gasUnits)
Set the gas_units.

void **setMmrCldAer** (const bool mmrCldAer)
Set the mmr_cldaer flag.

void **setP** (const std::vector< double > &p)
Set the p (pressure) vector.

void **setT** (const std::vector< double > &t)
Set the temperatures vector.

void **setQ** (const std::vector< double > &q)
Set item q for the profile (vector size must equal nlevels)

void **setO3** (const std::vector< double > &o3)
Set item o3 for the profile (vector size must equal nlevels)

void **setCO2** (const std::vector< double > &co2)
Set item co2 for the profile (vector size must equal nlevels)

void **setN2O** (const std::vector< double > &n2o)
Set item n2o for the profile (vector size must equal nlevels)

void **setCO** (const std::vector< double > &co)
Set item co for the profile (vector size must equal nlevels)

void **setCH4** (const std::vector< double > &ch4)
Set item ch4 for the profile (vector size must equal nlevels)

void **setSO2** (const std::vector< double > &so2)
Set item so2 for the profile (vector size must equal nlevels)

void **setCLW** (const std::vector< double > &clw)
Set item clw for the profile (vector size must equal nlevels)

void **setCfrac** (const std::vector< double > &cfrac)
Set item cfrac for the profile (vector size must equal nlevels)

void **setStco** (const std::vector< double > &stco)
Set item stco for the profile (vector size must equal nlevels)

void **setStma** (const std::vector< double > &stma)
Set item stma for the profile (vector size must equal nlevels)

void **setCucc** (const std::vector< double > &cucc)
Set item cucc for the profile (vector size must equal nlevels)

void **setCucp** (const std::vector< double > &cucp)

Set item cucp for the profile (vector size must equal nlevels)

void **setCuma** (const std::vector< double > &cuma)

Set item cuma for the profile (vector size must equal nlevels)

void **setCirr** (const std::vector< double > &cirr)

Set item cirr for the profile (vector size must equal nlevels)

void **setClwde** (const std::vector< double > &clwde)

Set item clwde for the profile (vector size must equal nlevels)

void **setIcede** (const std::vector< double > &icede)

Set item icede for the profile (vector size must equal nlevels)

void **setInso** (const std::vector< double > &inso)

Set item inso for the profile (vector size must equal nlevels)

void **setWaso** (const std::vector< double > &waso)

Set item waso for the profile (vector size must equal nlevels)

void **setSoot** (const std::vector< double > &soot)

Set item soot for the profile (vector size must equal nlevels)

void **setSsam** (const std::vector< double > &ssam)

Set item ssam for the profile (vector size must equal nlevels)

void **setSscm** (const std::vector< double > &sscm)

Set item sscm for the profile (vector size must equal nlevels)

void **setMinm** (const std::vector< double > &minm)

Set item minm for the profile (vector size must equal nlevels)

void **setMiam** (const std::vector< double > &miam)

Set item miam for the profile (vector size must equal nlevels)

void **setMicm** (const std::vector< double > &micm)

Set item micm for the profile (vector size must equal nlevels)

void **setMitr** (const std::vector< double > &mitr)

Set item mitr for the profile (vector size must equal nlevels)

void **setSuso** (const std::vector< double > &suso)

Set item suso for the profile (vector size must equal nlevels)

void **setVola** (const std::vector< double > &vola)

Set item vola for the profile (vector size must equal nlevels)

void **setVapo** (const std::vector< double > &vapo)

Set item vapo for the profile (vector size must equal nlevels)

void **setAsdu** (const std::vector< double > &asdu)

Set item asdu for the profile (vector size must equal nlevels)

void **setBcar** (const std::vector< double > &bcar)

Set item bcar for the profile (vector size must equal nlevels)

void **setDus1** (const std::vector< double > &dus1)

Set item dus1 for the profile (vector size must equal nlevels)

void **setDus2** (const std::vector< double > &dus2)

Set item dus2 for the profile (vector size must equal nlevels)

void **setDus3** (const std::vector< double > &dus3)

Set item dus3 for the profile (vector size must equal nlevels)

void **setSulp** (const std::vector< double > &sulp)

Set item sulp for the profile (vector size must equal nlevels)

void **setSsa1** (const std::vector< double > &ssa1)

Set item ssa1 for the profile (vector size must equal nlevels)

void **setSsa2** (const std::vector< double > &ssa2)

Set item ssa2 for the profile (vector size must equal nlevels)

void **setSsa3** (const std::vector< double > &ssa3)

Set item ssa3 for the profile (vector size must equal nlevels)

void **setOmat** (const std::vector< double > &omat)

Set item omat for the profile (vector size must equal nlevels)

void **setUserAerN** (const std::vector< double > &aer, const int n)

Set profile aer of user-defined aerosol species n (1<=n<=30) for the profile (vector size must equal nlevels)

void **setAngles** (const double satzen, const double satazi, const double sunzen, const double sunazi)

Set satellite an solar angles.

void **setS2m** (const double p_2m, const double t_2m, const double q_2m, const double u_10m, const double v_10m, const double wind_fetch)

Set surface 2m and 10m parameters.

void **setSkin** (const double t, const double salinity, const double snow_fraction, const double foam_fraction, const double fastem_coef_1, const double fastem_coef_2, const double fastem_coef_3, const double fastem_coef_4, const double fastem_coef_5)

Set skin parameters.

void **setSurfType** (const int surftype, const int watertype)

Set surface type parameters.

void **setSurfGeom** (const double lat, const double lon, const double elevation)

Set surface geometry parameters.

void **setDateTimes** (const int yy, const int mm, const int dd, const int hh, const int mn, const int ss)

Set date and time.

void **setSimpleCloud** (const double ctp, const double cfraction)

Set simple cloud parameters.

void **setClwScheme** (const int clw_scheme, const int clwde_param)

Set clwscheme parameter.

void **setIceCloud** (const int ice_scheme, const int icede_param)

Set ice cloud parameters.

void **setZeeman** (const double Be, const double cosbk)

Set zeeman parameters.

Appendix F: *Profiles* class (used with *Rttov* objects; C++ and Python)

C++ Profiles class

Typically an instance of this class is created, the profile data are assigned to it and then it is associated with one or more **Rttov** instances.

Profiles (int nbprofiles, const int nblevels)
Constructor method for individual gas specification.

void **setGasUnits** (int gasUnits)
Set the gas_units.

void **setMmrCldAer** (bool mmrcldaer)
Set the mmr_cldaer flag.

void **setP** (double *p)
Set the pointer to the p array of size [nprofiles][nlevels].

void **setT** (double *t)
Set the pointer to the t array of size [nprofiles][nlevels].

void **setQ** (double *q)
Set the pointer to the q array of size [nprofiles][nlevels].

void **setO3** (double *o3)
Set the pointer to the o3 array of size [nprofiles][nlevels].

void **setCO2** (double *co2)
Set the pointer to the co2 array of size [nprofiles][nlevels].

void **setCO** (double *co)
Set the pointer to the co array of size [nprofiles][nlevels].

void **setN2O** (double *n2o)
Set the pointer to the n2o array of size [nprofiles][nlevels].

void **setCH4** (double *ch4)
Set the pointer to the ch4 array of size [nprofiles][nlevels].

void **setSO2** (double *so2)
Set the pointer to the so2 array of size [nprofiles][nlevels].

void **setCLW** (double *clw)
Set the pointer to the clw array of size [nprofiles][nlevels].

void **setAngles** (double *angles)
Set the pointer to the angles array of size [nprofiles][4] containing satzen, satazi, sunzen, sunazi for each profile.

void **setS2m** (double *s2m)
Set the pointer to the s2m array of size [nprofiles][6] containing 2m p, 2m t, 2m q, 10m wind u, v, wind fetch for each profile.

void **setSkin** (double *skin)

Set the pointer to the skin array of size [nprofiles][9] containing skin T, salinity, snow_fraction, foam_fraction, fastem_coefs(1:5) for each profile.

void **setSurfType** (int *surftype)

Set the pointer to the surftype array of size [nprofiles][2] containing surftype, watertype for each profile.

void **setSurfGeom** (double *surfgeom)

Set the pointer to the surfgeom array of size [nprofiles][3] containing latitude, longitude, elevation for each profile.

void **setDateTimes** (int *datetimes)

Set the pointer to the datetimes array of size [nprofiles][6] containing yy, mm, dd, hh, mm, ss for each profile.

void **setSimpleCloud** (double *simplecloud)

Set the pointer to the simplecloud array of size [nprofiles][2] containing ctp, cfraction for each profile.

void **setClwScheme** (int *clwscheme)

Set the pointer to the clwscheme array of size [nprofiles][2] containing clw_scheme, clwde_param for each profile.

void **setIceCloud** (int *icecloud)

Set the pointer to the icecloud array of size [nprofiles][2] containing ice_scheme, icede_param for each profile.

void **setZeeman** (double *zeeman)

Set the pointer to the zeeman array of size [nprofiles][2] containing be, cosbk for each profile.

void **setGasItem** (double *gasItem, **rttov::itemIdType** item_id)

Set a gas, cloud or aerosol profile variable; item likes clouds, cfrac or aerosols must have the same dimensions as temperature or water vapour [nprofiles][nlevels].

Python Profiles class

Typically an instance of this class is created, the profile data are assigned to it and then it is associated with one or more **Rttov** instances.

Methods:

Profiles (nprofiles, nlevels)

Constructor method.

setUserAerN (aer, n)

Set profile aer of size [nprofiles][nlevels] of user-defined aerosol species n (1<=n<=30). You can also access these individually via the AerN (N=1,2,...,30) members described below.

delUserAerN (n)

Delete profile data for user-defined aerosol species n (1<=n<=30).

Members:

int **GasUnits**

The gas_units.

int **MmrCldAer**

The mmr_cldaer flag.

float array **P**
The p array of size [nprofiles][nlevels].

float array **T**
The t array of size [nprofiles][nlevels].

float array **Q**
The q array of size [nprofiles][nlevels].

float array **O3**
The o3 array of size [nprofiles][nlevels].

float array **CO2**
The co2 array of size [nprofiles][nlevels].

float array **CO**
The co array of size [nprofiles][nlevels].

float array **N2O**
The n2o array of size [nprofiles][nlevels].

float array **CH4**
The ch4 array of size [nprofiles][nlevels].

float array **SO2**
The so2 array of size [nprofiles][nlevels].

float array **CLW**
The clw array of size [nprofiles][nlevels].

float array **Angles**
The angles array of size [nprofiles][4] containing satzen, satazi, sunzen, sunazi for each profile.

float array **S2m**
The s2m array of size [nprofiles][6] containing 2m p, 2m t, 2m q, 10m wind u, v, wind fetch for each profile.

float array **Skin**
The skin array of size [nprofiles][9] containing skin T, salinity, snow_fraction, foam_fraction, fastem_coefs(1:5) for each profile.

int array **SurfType**
The surftype array of size [nprofiles][2] containing surftype, watertype for each profile.

float array **SurfGeom**
The surfgeom array of size [nprofiles][3] containing latitude, longitude, elevation for each profile.

int array **DateTimes**
The datetimes array of size [nprofiles][6] containing yy, mm, dd, hh, mm, ss for each profile.

float array **SimpleCloud**
The simplecloud array of size [nprofiles][2] containing ctp, cfraction for each profile.

int array **IceCloud**
The icecloud array of size [nprofiles][2] containing ice scheme, icede_param for each profile.

int array **ClwScheme**
The clwscheme array of size [nprofiles][2] containing clw scheme, clwde_param for each profile.

float array **Zeeman**
The zeeman array of size [nprofiles][2] containing be, cosbk for each profile.

float array **Cfrac**
The cfrac array of size [nprofiles][nlevels].

float array **Stco**
The stco (cloud type 1) array of size [nprofiles][nlevels].

float array **Stma**
The stma (cloud type 2) array of size [nprofiles][nlevels].

float array **Cucc**
The cucc (cloud type 3) array of size [nprofiles][nlevels].

float array **Cucp**
The cucp (cloud type 4) array of size [nprofiles][nlevels].

float array **Cuma**
The cuma (cloud type 5) array of size [nprofiles][nlevels].

float array **Cirr**
The cirr (cloud type 6) array of size [nprofiles][nlevels].

float array **Icede**
The icede array of size [nprofiles][nlevels].

float array **Clwde**
The clwde array of size [nprofiles][nlevels].

float array **Inso**
The inso (aerosol type 1) array of size [nprofiles][nlevels].

float array **Waso**
The waso (aerosol type 2) array of size [nprofiles][nlevels].

float array **Soot**
The soot (aerosol type 3) array of size [nprofiles][nlevels].

float array **Ssam**
The ssam (aerosol type 4) array of size [nprofiles][nlevels].

float array **Sscm**
The sscm (aerosol type 5) array of size [nprofiles][nlevels].

float array **Minm**
The minm (aerosol type 6) array of size [nprofiles][nlevels].

float array **Miam**
The miam (aerosol type 7) array of size [nprofiles][nlevels].

float array **Micm**
The micm (aerosol type 8) array of size [nprofiles][nlevels].

float array **Mitr**
The mitr (aerosol type 9) array of size [nprofiles][nlevels].

float array **Suso**
The suso (aerosol type 10) array of size [nprofiles][nlevels].

float array **Vola**
The vola (aerosol type 11) array of size [nprofiles][nlevels].

float array **Vapo**
The vapo (aerosol type 12) array of size [nprofiles][nlevels].

float array **Asdu**
The asdu (aerosol type 13) array of size [nprofiles][nlevels].

float array **Bcar**
The bcar (aerosol type 1) array of size [nprofiles][nlevels].

float array **Dus1**
The dus1 (aerosol type 2) array of size [nprofiles][nlevels].

float array **Dus2**
The dus2 (aerosol type 3) array of size [nprofiles][nlevels].

float array **Dus3**
The dus3 (aerosol type 4) array of size [nprofiles][nlevels].

float array **Sulp**
The sulp (aerosol type 5) array of size [nprofiles][nlevels].

float array **Ssa1**
The ssa1 (aerosol type 6) array of size [nprofiles][nlevels].

float array **Ssa2**
The ssa2 (aerosol type 7) array of size [nprofiles][nlevels].

float array **Ssa3**
The ssa3 (aerosol type 8) array of size [nprofiles][nlevels].

float array **Omat**
The omat (aerosol type 9) array of size [nprofiles][nlevels].

float array **AerN** *where N=1, 2, …, 30*
The user-defined aerosol species N array of size [nprofiles][nlevels].

Appendix G: *ProfileScatt* class (used with *RttovScattSafe* objects; C++ only)

Typically a vector of instances of this class is created, the profile data are assigned to each instance and then the vector is associated with one or more **RttovScattSafe** instances.

ProfileScatt (int nlevels)
Constructor method.

void **setGasUnits** (**rttov::gasUnitType** gasUnits)
Set the gas_units.

void **setP** (const std::vector< double > &p)
Set the p (pressure) vector.

void **setPh** (const std::vector< double > &ph)
Set the ph (pressure half-levels) vector.

void **setT** (const std::vector< double > &t)
Set the temperatures vector.

void **setQ** (const std::vector< double > &q)
Set item q for the profile (vector size must equal nlevels)

void **setO3** (const std::vector< double > &o3)
Set item o3 for the profile (vector size must equal nlevels)

void **setHydroFrac** (const std::vector< double > &hydro_frac)
Set item hydro_frac for the profile (vector size must equal nlevels)

void **setClw** (const std::vector< double > &clw)
Set item clw for the profile (vector size must equal nlevels)

void **setCiw** (const std::vector< double > &ciw)
Set item ciw for the profile (vector size must equal nlevels)

void **setSnow** (const std::vector< double > &snow)
Set item snow for the profile (vector size must equal nlevels)

void **setRain** (const std::vector< double > &rain)
Set item rain for the profile (vector size must equal nlevels)

void **setGraupel** (const std::vector< double > &graupel)
Set item graupel for the profile (vector size must equal nlevels)

void **setUserCfrac** (const double usercfrac_in)
Set user cfrac for the profile.

void **setHydroN** (const std::vector< double > &hydro, const int n)
Set profile hydro for hydrometeor type n (1<=n<=30) for the profile (vector size must equal nlevels)

void **setHydroFracN** (const std::vector< double > &hydro_frac, const int n)
Set profile hydro_frac for hydrometeor type n (1<=n<=30) for the profile (vector size must equal nlevels)

void **setAngles** (const double satzen, const double satazi)
Set satellite angles.

void **setS2m** (const double p_2m, const double t_2m, const double q_2m, const double u_10m, const double v_10m)
Set surface 2m and 10m parameters.

void **setSkin** (const double t, const double salinity, const double foam_fraction, const double fastem_coef_1, const double fastem_coef_2, const double fastem_coef_3, const double fastem_coef_4, const double fastem_coef_5)
Set skin parameters.

void **setSurfType** (const int surftype_in)
Set surface type.

void **setSurfGeom** (const double lat, const double lon, const double elevation)
Set surface geometry parameters.

void **setDateTimes** (const int yy, const int mm, const int dd, const int hh, const int mn, const int ss)
Set date and time.

void **setZeeman** (const double Be, const double cosbk)
Set zeeman parameters.

Appendix H: *ProfilesScatt* class (used with *RttovScatt* objects; C++ and Python)

C++ ProfilesScatt class

Typically an instance of this class is created, the profile data are assigned to it and then it is associated with one or more **RttovScatt** instances.

ProfilesScatt (int nbprofiles, const int nblevels)
Constructor method for individual gas specification.

void **setGasUnits** (int gasUnits)
Set the gas_units.

void **setP** (double *p)
Set the pointer to the p array of size [nprofiles][nlevels].

void **setPh** (double *ph)
Set the pointer to the ph array of size [nprofiles][nlevels+1].

void **setT** (double *t)
Set the pointer to the t array of size [nprofiles][nlevels].

void **setQ** (double *q)
Set the pointer to the q array of size [nprofiles][nlevels].

void **setO3** (double *o3)
Set the pointer to the o3 array of size [nprofiles][nlevels].

void **setHydroFrac** (double *hydro_frac)
Set the pointer to the hydro_frac array of size [nprofiles][nlevels].

void **setClw** (double *clw)
Set the pointer to the clw array of size [nprofiles][nlevels].

void **setCiw** (double *ciw)
Set the pointer to the ciw array of size [nprofiles][nlevels].

void **setSnow** (double *snow)
Set the pointer to the snow array of size [nprofiles][nlevels].

void **setRain** (double *rain)
Set the pointer to the rain array of size [nprofiles][nlevels].

void **setGraupel** (double *graupel)
Set the pointer to the graupel array of size [nprofiles][nlevels].

void **setUserCfrac** (double *usercfrac)
Set the pointer to the user cfrac array of size [nprofiles].

void **setAngles** (double *angles)
Set the pointer to the angles array of size [nprofiles][2] containing satzen, satazi for each profile.

void **setS2m** (double *s2m)
Set the pointer to the s2m array of size [nprofiles][5] containing 2m p, 2m t, 2m q, 10m wind u, v for each profile.

void **setSkin** (double *skin)

Set the pointer to the skin array of size [nprofiles][8] containing skin T, salinity, foam_fraction, fastem_coefs(1:5) for each profile.

void **setSurfType** (int *surftype)

Set the pointer to the surftype array of size [nprofiles] containing surftype for each profile.

void **setSurfGeom** (double *surfgeom)

Set the pointer to the surfgeom array of size [nprofiles][3] containing latitude, longitude, elevation for each profile.

void **setDateTimes** (int *datetimes)

Set the pointer to the datetimes array of size [nprofiles][6] containing yy, mm, dd, hh, mm, ss for each profile.

void **setZeeman** (double *zeeman)

Set the pointer to the zeeman array of size [nprofiles][2] containing be, cosbk for each profile.

void **setGasItem** (double *gasItem, **rttov::itemIdType** item_id)

Set a gas or hydrometeor profile variable must have the same dimensions as temperature or water vapour [nprofiles][nlevels].

Python ProfilesScatt class

Typically an instance of this class is created, the profile data are assigned to it and then it is associated with one or more **RttovScatt** instances.

Methods:

ProfilesScatt (nprofiles, nlevels)

Constructor method.

setHydroN (hydro, n)

Set profile hydro of size [nprofiles][nlevels] of hydrometeor type n (1<=n<=30). You can also access these individually via the HydroN (N=1,2,...,30) members described below.

delHydroN (n)

Delete profile data for hydrometeor type n (1<=n<=30).

setHydroFracN (hydro_frac, n)

Set profile hydro_frac of size [nprofiles][nlevels] of cloud fraction for hydrometeor type n (1<=n<=30). You can also access these individually via the HydroFracN (N=1,2,...,30) members described below.

delHydroFracN (n)

Delete profile data for cloud fraction for hydrometeor type n (1<=n<=30).

Members:

int **GasUnits**

The gas_units.

float array **P**

The p array of size [nprofiles][nlevels].

float array **Ph**

The ph array of size [nprofiles][nlevels+1].

float array **T**

The t array of size [nprofiles][nlevels].

float array **Q** (double *q)

The q array of size [nprofiles][nlevels].

float array **O3** (double *o3)

The o3 array of size [nprofiles][nlevels].

float array **HydroFrac** (double *hydro_frac)

The hydro_frac array of size [nprofiles][nlevels].

float array **Clw** (double *clw)

The clw array of size [nprofiles][nlevels].

float array **Ciw** (double *ciw)

The ciw array of size [nprofiles][nlevels].

float array **Snow** (double *snow)

The snow array of size [nprofiles][nlevels].

float array **Rain** (double *rain)

The rain array of size [nprofiles][nlevels].

float array **Graupel** (double *graupel)

The graupel array of size [nprofiles][nlevels].

float array **UserCfrac** (double *usercfrac)

The user cfrac array of size [nprofiles].

float array **Angles** (double *angles)

The angles array of size [nprofiles][2] containing satzen, satazi for each profile.

float array **S2m** (double *s2m)

The s2m array of size [nprofiles][5] containing 2m p, 2m t, 2m q, 10m wind u, v for each profile.

float array **Skin** (double *skin)

The skin array of size [nprofiles][8] containing skin T, salinity, foam_fraction, fastem_coefs(1:5) for each profile.

float array **SurfType** (int *surftype)

The surftype array of size [nprofiles] containing surftype for each profile.

float array **SurfGeom** (double *surfgeom)

The surfgeom array of size [nprofiles][3] containing latitude, longitude, elevation for each profile.

float array **DateTimes** (int *datetimes)

The datetimes array of size [nprofiles][6] containing yy, mm, dd, hh, mm, ss for each profile.

float array **Zeeman** (double *zeeman)

The zeeman array of size [nprofiles][2] containing be, cosbk for each profile.

float array **HydroN** *where N=1, 2, ..., 30*

The hydrometeor type N array of size [nprofiles][nlevels].

float array **HydroFracN** *where N=1, 2, ..., 30*

The hydromteor type N cloud fraction array of size [nprofiles][nlevels].

Appendix I: *Options* class (C++ and Python)

C++ Options class

The methods listed below are used to set the RTTOV and wrapper options. Methods also exist to query the options: see wrapper/Options.h. The **Rttov/RttovSafe/RttovScatt/RttovScattSafe** objects have options members so there is usually no need to create instances of this class manually. Note that some members access both the standard RTTOV and the RTTOV-SCATT-equivalent options at the same time.

Options ()

Constructor method.

void **setApplyRegLimits** (bool applyRegLimts)

Set the opts%config%apply_reg_limits and opts_scatt%config%apply_reg_limits options.

void **setDoCheckinput** (bool doCheckinput)

Set the opts%config%do_checkinput and opts_scatt%config%do_checkinput options.

void **setVerbose** (bool verbose)

Set the opts%config%verbose and opts_scatt%config%verbose options.

void **setOpdep13GasClip** (bool opdep13GasClip)

Set the opts%config%opdep13_gas_clip and opts_scatt%config% opdep13_gas_clip options.

void **setFixHgpl** (bool fixHgpl)

Set the opts%config%verbose and opts_scatt%config%verbose options.

void **setAddInterp** (bool addinterp)

Set the opts%interpolation%addinterp option.

void **setInterpMode** (int interpMode)

Set the opts%interpolation%interp_mode and opts_scatt%interp_mode options.

void **setRegLimitExtrap** (bool regLimitExtrap)

Set the opts%interpolation%reg_limit_extrap and opts_scatt%reg_limit_extrap options.

void **setSpacetop** (bool spacetop)

Set the opts%interpolation%spacetop option.

void **setLgradp** (bool lgradp)

Set the opts%interpolation%lgradp and opts_scatt%lgradp options.

void **setOzoneData** (bool ozoneData)

Set the opts%rt_all%ozone_data and opts_scatt%ozone_data options.

void **setCO2Data** (bool co2Data)

Set the opts%rt_all%co2_data option.

void **setCH4Data** (bool ch4Data)

Set the opts%rt_all%ch4_data option.

void **setCOData** (bool coData)

Set the opts%rt_all%co_data option.

void **setN2OData** (bool n2oData)

Set the opts%rt_all%n2o_data option.

void **setSO2Data** (bool so2Data)
Set the opts%rt_all%so2_data option.

void **setDoLambertian** (bool doLambertian)
Set the opts%rt_all%do_lambertian option.

void **setLambertianFixedAngle** (bool lambertianFixedAngle)
Set the opts%rt_all%lambertian_fixed_angle option.

void **setUseT2mOpdep** (bool useT2mOpdep)
Set the opts%rt_all%use_t2m_opdep and opts_scatt%use_t2m_opdep options.

void **setUseQ2m** (bool useQ2m)
Set the opts%rt_all%use_q2m and opts_scatt%use_q2m options.

void **setUseTskinEff** (bool useTskinEff)
Set the opts%rt_all%use_tskin_eff and opts_scatt% use_tskin_eff options.

void **setSwitchrad** (bool switchrad)
Set the opts%rt_all%switchrad option.

void **setAddRefrac** (bool addRefrac)
Set the opts%rt_all%addrefrac option.

void **setPlaneParallel** (bool planeParallel)
Set the opts%rt_all%plane_parallel option.

void **setRadDownLinTau** (bool radDownLinTau)
Set the opts%rt_all%rad_down_lin_tau and opts_scatt%rad_down_lin_tau options.

void **setDtauTest** (bool dtauTest)
Set the opts%rt_all%dtau_test and opts_scatt%dtau_test options.

void **setTransmittancesOnly**(bool transmittancesOnly)
Set the opts%rt_all%transmittances_only option.

void **setCLWData** (bool clwData)
Set the opts%rt_mw%clw_data option.

void **setCLWScheme** (int clwScheme)
Set the opts%rt_mw%clw_scheme option.

void **setCLWCloudTop** (double clwCloudTop)
Set the opts%rt_mw%clw_cloud_top option.

void **setFastemVersion** (int fastemVersion)
Set the opts%rt_mw%fastem_version and opts_scatt%fastem_version options.

void **setFastem3RwdFix** (int fastem3RwdFix)
Set the opts%rt_mw%fastem3_rwd_fix and opts_scatt%fastem3_rwd_fix options.

void **setSupplyFoamFraction** (bool supplyFoamFraction)
Set the opts%rt_mw%supply_foam_fraction and opts_scatt%supply_foam_fraction options.

void **setSolarSeaBrdfModel** (int solarSeaBrdfModel)
Set the opts%rt_ir%solar_sea_brdf_model option.

void **setIrSeaEmisModel** (int irSeaEmisModel)
Set the opts%rt_ir%ir_sea_emis_model option.

void **setAddSolar** (bool addsolar)
Set the opts%rt_ir%addsolar option.

void **setRayleighMaxWavelength** (double rayleighMaxWavelength)
Set the opts%rt_ir%rayleigh_max_wavelength option.

void **setRayleighMinPressure** (double rayleighMinPressuer)
Set the opts%rt_ir%rayleigh_min_pressure option.

void **setRayleighSingleScatt** (bool rayleighSingleScatt)
Set the opts%rt_ir%rayleigh_single_scatt option.

void **setRayleighDepol** (bool rayleighDepol)
Set the opts%rt_ir%rayleigh_depol option.

void **setDoNlteCorrection** (bool doNlteCorrection)
Set the opts%rt_ir%do_nlte_correction option.

void **setAddAerosl** (bool addaerosl)
Set the opts%rt_ir%addaerosl option.

void **setAddClouds** (bool addclouds)
Set the opts%rt_ir%addclouds option.

void **setUserAerOptParam** (bool userAerOptParam)
Set the opts%rt_ir%user_aer_opt_param option.

void **setUserCldOptParam** (bool userCldOptParam)
Set the opts%rt_ir%user_cld_opt_param option.

void **setGridBoxAvgCloud** (bool gridBoxAvgCloud)
Set the opts%rt_ir%grid_box_avg_cloud option.

void **setCloudOverlap** (int cloudOverlap)
Set the opts%rt_ir%cloud_overlap option.

void **setCCLowCloudTop** (double ccLowCloudTop)
Set the opts%rt_ir%cc_low_cloud_top option.

void **setCldcolThreshold** (double cldcolThreshold)
Set the opts%rt_ir%cldcol_threshold option.

void **setIrScattModel** (int irScattModel)
Set the opts%rt_ir%ir_scatt_model option.

void **setVisScattModel** (int visScattModel)
Set the opts%rt_ir%vis_scatt_model option.

void **setDomNstreams** (int domNstreams)
Set the opts%rt_ir%dom_nstreams option.

void **setDomAccuracy** (double domAccuracy)
Set the opts%rt_ir%dom_accuracy option.

void **setDomOpdepThreshold** (double domOpdepThreshold)
Set the opts%rt_ir%dom_opdep_threshold option.

void **setDomRayleigh** (bool domRayleigh)
Set the opts%rt_ir%dom_rayleigh option.

void **setLuserCfrac** (bool lusercfrac)
Set the opts_scatt%lusercfrac option.

void **setCCThreshold** (double ccThreshold)
Set the opts_scatt%cc_threshold option.

void **setPolMode** (bool polMode)
Set the opts_scatt%pol_mode option.

void **setIcePolarisation** (double icePolarisation)
Set the opts_scatt%ice_polarisation option.

void **setHydroCfracTLAD** (bool hydroCfracTLAD)
Set the opts_scatt%hydro_cfrac_tlad option.

void **setZeroHydroTLAD** (bool zeroHydroTLAD)
Set the opts_scatt%zero_hydro_tlad option.

void **setDoOpdepCalc** (bool doOpdepCalc)
Set the opts%dev%do_opdep_calc option.

void **setNthreads** (int nthreads)
Set the number of threads RTTOV will use (compile RTTOV with OpenMP to make use of this)

void **setNprofsPerCall** (int nprofsPerCall)
Set the number of profiles passed into rttov_direct or rttov_k per call.

void **setVerboseWrapper** (bool verboseWrapper)
Set the verbose_wrapper option.

void **setCheckOpts** (bool checkOpts)
Set the check_opts option.

void **setStoreRad** (bool storeRad)
Set the store_rad wrapper option.

void **setStoreRad2** (bool storeRad2)
Set the store_rad2 wrapper option.

void **setStoreTrans** (bool storeTrans)
Set the store_trans wrapper option.

void **setStoreEmisTerms** (bool storeEmisTerms)
Set the store_emis_terms wrapper option.

bool **isApplyRegLimits** ()
Return the opts%config%apply_reg_limits and opts_scatt%config%apply_reg_limits options.

bool **isDoCheckinput** ()
Return the opts%config%do_checkinput and opts_scatt%config%do_checkinput options.

bool **isVerbose** ()
Return the opts%config%verbose and opts_scatt%config%verbose options.

bool **isOpdep13GasClip** ()
Return the opts%config%opdep13_gas_clip and opts_scatt%config% opdep13_gas_clip options.

bool **isFixHgpl** ()
Return the opts%config%fix_hgpl and opts_scatt%config%fix_hgpl options.

bool **isAddInterp** ()
Return the opts%interpolation%addinterp option.

int **getInterpMode** () const
Return the opts%interpolation%interp_mode and opts_scatt%interp_mode options.

bool **isRegLimitExtrap** ()
Return the opts%interpolation%reg_limit_extrap and opts_scatt%reg_limit_extrap options.

bool **isSpacetop** ()
Return the opts%interpolation%spacetop option.

bool **isLgradp** ()
Return the opts%interpolation%lgradp and opts_scatt%lgradp options.

bool **isOzoneData** ()
Return the opts%rt_all%ozone_data and opts_scatt%ozone_data options.

bool **isCO2Data** ()
Return the opts%rt_all%co2_data option.

bool **isCH4Data** ()
Return the opts%rt_all%ch4_data option.

bool **isCOData** ()
Return the opts%rt_all%co_data option.

bool **isN2OData** ()
Return the opts%rt_all%n2o_data option.

bool **isSO2Data** ()
Return the opts%rt_all%so2_data option.

bool **isDoLambertian** ()
Return the opts%rt_all%do_lambertian option.

bool is**LambertianFixedAngle** ()
Return the opts%rt_all%lambertian_fixed_angle option.

bool **isUseT2mOpdep** ()
Return the opts%rt_all%use_t2m_opdep and opts_scatt%use_t2m_opdep options.

bool **isUseQ2m** ()
Return the opts%rt_all%use_q2m and opts_scatt%use_q2m options.

bool **isUseTskinEff** ()
Return the opts%rt_all%use_tskin_eff and opts_scatt% use_tskin_eff options.

bool **isSwitchrad** ()
Return the opts%rt_all%switchrad option.

bool **isAddRefrac** ()
Return the opts%rt_all%addrefrac option.

bool **isPlaneParallel** ()
Return the opts%rt_all%plane_parallel option.

bool **isRadDownLinTau** ()
Return the opts%rt_all%rad_down_lin_tau and opts_scatt%rad_down_lin_tau options.

bool **isDtauTest** ()
Return the opts%rt_all%dtau_test and opts_scatt%dtau_test options.

bool **isTransmittancesOnly** ()
Return the opts%rt_all%transmittances_only option.

bool **isCLWData** ()
Return the opts%rt_mw%clw_data option.

int **getCLWScheme** () const
Return the opts%rt_mw%clw_scheme option.

double **getCLWCloudTop** () const
Return the opts%rt_mw%clw_cloud_top option.

int **getFastemVersion** () const
Return the opts%rt_mw%fastem_version and opts_scatt%fastem_version options.

bool is**Fastem3RwdFix** ()
Return the opts%rt_mw%fastem3_rwd_fix and opts_scatt%fastem3_rwd_fix options.

bool **isSupplyFoamFraction** ()
Return the opts%rt_mw%supply_foam_fraction and opts_scatt%supply_foam_fraction options.

int **getSolarSeaBrdfModel** () const
Return the opts%rt_ir%solar_sea_brdf_model option.

int **getIrSeaEmisModel** () const
Return the opts%rt_ir%ir_sea_emis_model option.

bool **isAddSolar** ()
Return the opts%rt_ir%addsolar option.

double **getRayleighMaxWavelength** () const
Return the opts%rt_ir%rayleigh_max_wavelength option.

double **getRayleighMinPressure** () const
Return the opts%rt_ir%rayleigh_min_pressure option.

bool **isRayleighSingleScatt** ()
Return the opts%rt_ir%rayleigh_single_scatt option.

bool **isRayleighDepol ()**
Return the opts%rt_ir%rayleigh_depol option.

bool **isDoNlteCorrection** ()
Return the opts%rt_ir%do_nlte_correction option.

bool **isAddAerosl** ()
Return the opts%rt_ir%addaerosl option.

bool **isAddClouds** ()
Return the opts%rt_ir%addclouds option.

bool **isUserAerOptParam** ()
Return the opts%rt_ir%user_aer_opt_param option.

bool **isUserCldOptParam** ()
Return the opts%rt_ir%user_cld_opt_param option.

bool **isGridBoxAvgCloud** ()
Return the opts%rt_ir%grid_box_avg_cloud option.

bool **getCloudOverlap** ()
Return the opts%rt_ir%cloud_overlap option.

double **getCCLowCloudTop** () const
Return the opts%rt_ir%cc_low_cloud_top option.

double **getCldcolThreshold** () const
Return the opts%rt_ir%cldcol_threshold option.

int **getIrScattModel** () const
Return the opts%rt_ir%ir_scatt_model option.

int **getVisScattModel** () const
Return the opts%rt_ir%vis_scatt_model option.

int **getDomNstreams** () const
Return the opts%rt_ir%dom_nstreams option.

double **getDomAccuracy** () const
Return the opts%rt_ir%dom_accuracy option.

double **getDomOpdepThreshold** () const
Return the opts%rt_ir%dom_opdep_threshold option.

bool **isAddPC** ()
Return the opts%rt_ir%pc%addpc option.

bool **isAddRadrec** ()
Return the opts%rt_ir%pc%addradrec option.

int **getIpcreg** () const
Return the opts%rt_ir%pc%ipcreg option.

int **getIpcbnd** () const
Return the opts%rt_ir%pc%ipcbnd option.

bool **isLuserCfrac** ()
Return the opts_scatt%lusercfrac option.

double **getCCThreshold** () const
Return the opts_scatt%cc_threshold option.

int **isPolMode** () const
Return the opts_scatt%pol_mode option.

double **getIcePolarisation** () const
Return the opts_scatt%ice_polarisation option.

bool **isHydroCfracTLAD** ()
Return the opts_scatt%hydro_cfrac_tlad option.

bool **isZeroHydroTLAD** ()
Return the opts_scatt%zero_hydro_tlad option.

bool **isDoOpdepCalc** ()
Return the opts%dev%do_opdep_calc option.

int **getNthreads** () const
Return the number of threads RTTOV will use (compile RTTOV with OpenMP to make use of this)

int **getNprofsPerCall** () const
Return the number of profiles passed into rttov_direct or rttov_k per call.

bool **isVerboseWrapper** () const
Return set the verbose_wrapper option.

bool **isCheckOpts** () const
Return set the check_opts option.

bool **isStoreRad** () const
Return the store_rad wrapper option.

bool **isStoreRad2** () const
Return the store_rad2 wrapper option.

bool **isStoreTrans** () const
Return the store_trans wrapper option.

bool **isStoreEmisTerms** () const
Return the store_emis_terms wrapper option.

Python Options class

The members below correspond directly to the RTTOV and wrapper options and are referenced directly. The **Rttov/RttovScatt** classes have an Options member so there is usually no need to create instances of this class manually. Note that some members access both the standard RTTOV and the RTTOV-SCATT-equivalent options at the same time.

Methods:

Options ()
Constructor method.

Members:

bool **ApplyRegLimits**
The opts%config%apply_reg_limits and opts_scatt%config%apply_reg_limits options.

bool **DoCheckinput**
The opts%config%do_checkinput and opts_scatt%config%do_checkinput options.

bool **Verbose**
The opts%config%verbose and opts_scatt%config%verbose options.

bool **Opdep13GasClip**
The opts%config%opdep13_gas_clip and opts_scatt%config%opdep13_gas_clip options.

bool **FixHgpl**
The opts%config%fix_hgpl and opts_scatt%config%fix_hgpl options.

bool **AddInterp**
The opts%interpolation%addinterp option.

int **InterpMode**
The opts%interpolation%interp_mode and opts_scatt%interp_mode options.

bool **RegLimitExtrap**
The opts%interpolation%reg_limit_extrap and opts_scatt%reg_limit_extrap options.

bool **Spacetop**
The opts%interpolation%spacetop option.

bool **Lgradp**
The opts%interpolation%lgradp and opts_scatt%lgradp options.

bool **OzoneData**
The opts%rt_all%ozone_data and opts_scatt%ozone_data options.

bool **CO2Data**
The opts%rt_all%co2_data option.

bool **CH4Data**
The opts%rt_all%ch4_data option.

bool **COData**
The opts%rt_all%co_data option.

bool **N2OData**
The opts%rt_all%n2o_data option.

bool **SO2Data**
The opts%rt_all%so2_data option.

bool **DoLambertian**
The opts%rt_all%do_lambertian option.

bool **LambertianFixedAngle**
The opts%rt_all%lambertian_fixed_angle option.

bool **UseT2mOpdep**
The opts%rt_all%use_t2m_opdep and opts_scatt%use_t2m_opdep options.

bool **UseQ2m**
The opts%rt_all%use_q2m and opts_scatt%use_q2m options.

bool **UseTskinEff**
The opts%rt_all%use_tskin_eff and opts_scatt% use_tskin_eff options.

bool **Switchrad**
The opts%rt_all%switchrad option.

bool **AddRefrac**
The opts%rt_all%addrefrac option.

bool **PlaneParallel**
The opts%rt_all%plane_parallel option.

bool **RadDownLinTau**
The opts%rt_all%rad_down_lin_tau and opts_scatt%rad_down_lin_tau options.

bool **DtauTest**
The opts%rt_all%dtau_test and opts_scatt%dtau_test options.

bool **TransmittancesOnly**
The opts%rt_all%transmittances_only option.

bool **CLWData**
The opts%rt_mw%clw_data option.

int **CLWScheme**
The opts%rt_mw%clw_scheme option.

float **CLWCloudTop**
The opts%rt_mw%clw_cloud_top option.

int **FastemVersion**
The opts%rt_mw%fastem_version and opts_scatt%fastem_version options.

bool **Fastem3RwdFix** ()
The opts%rt_mw%fastem3_rwd_fix and opts_scatt%fastem3_rwd_fix options.

bool **SupplyFoamFraction**
The opts%rt_mw%supply_foam_fraction and opts_scatt%supply_foam_fraction options.

int **SolarSeaBrdfModel**
The opts%rt_ir%solar_sea_brdf_model option.

int **IrSeaEmisModel**
The opts%rt_ir%ir_sea_emis_model option.

bool **AddSolar**
The opts%rt_ir%addsolar option.

float **RayleighMaxWavelength**
The opts%rt_ir%rayleigh_max_wavelength option.

float **RayleighMinPressure**
The opts%rt_ir%rayleigh_min_pressure option.

bool **RayleighSingleScatt**
The opts%rt_ir%rayleigh_single_scatt option.

bool **RayleighDepol**
The opts%rt_ir%rayleigh_depol option.

bool **DoNlteCorrection**
The opts%rt_ir%do_nlte_correction option.

bool **AddAerosl**
The opts%rt_ir%addaerosl option.

bool **AddClouds**
The opts%rt_ir%addclouds option.

bool **UserAerOptParam**
The opts%rt_ir%user_aer_opt_param option.

bool **UserCldOptParam**
The opts%rt_ir%user_cld_opt_param option.

bool **GridBoxAvgCloud**
The opts%rt_ir%grid_box_avg_cloud option.

bool **CloudOverlap**
The opts%rt_ir%cloud_overlap option.

float **CCLowCloudTop**
The opts%rt_ir%cc_low_cloud_top option.

float **CldcolThreshold**
The opts%rt_ir%cldcol_threshold option.

int **IrScattModel**
The opts%rt_ir%ir_scatt_model option.

int **VisScattModel**
The opts%rt_ir%vis_scatt_model option.

int **DomNstreams**
The opts%rt_ir%dom_nstreams option.

float **DomAccuracy**
The opts%rt_ir%dom_accuracy option.

float **DomOpdepThreshold**
The opts%rt_ir%dom_opdep_threshold option.

bool **DomRayleigh**
The opts%rt_ir%dom_rayleigh option.

bool **AddPC**
The opts%rt_ir%pc%addpc option.

bool **AddRadrec**
The opts%rt_ir%pc%addradrec option.

int **Ipcreg**
The opts%rt_ir%pc%ipcreg option.

int **Ipcbnd**
The opts%rt_ir%pc%ipcbnd option.

bool **LuserCfrac**
The opts_scatt%lusercfrac option.

float **CCThreshold**
The opts_scatt%cc_threshold option.

int **PolMode** ()
The opts_scatt%pol_mode option.

float **IcePolarisation**
The opts_scatt%ice_polarisation option.

bool **HydroCfracTLAD**
The opts_scatt%hydro_cfrac_tlad option.

bool **ZeroHydroTLAD**
The opts_scatt%zero_hydro_tlad option.

bool **DoOpdepCalc** ()
The opts%dev%do_opdep_calc option.

int **Nthreads**
The number of threads RTTOV will use (compile RTTOV with OpenMP to make use of this)

int **NprofsPerCall**
The number of profiles passed into rttov_direct or rttov_k per call.

bool **VerboseWrapper**
Return set the verbose_wrapper option.

bool **CheckOpts**
Return set the check_opts option.

bool **StoreRad**
The store_rad wrapper option.

bool **StoreRad2**
The store_rad2 wrapper option.

bool **StoreTrans**
The store_trans wrapper option.

bool **StoreEmisTerms**
The store_emis_terms wrapper option.

Appendix J: *Atlas* class (C++ and Python)

C++ Atlas class

Atlas ()
Atlas class constructor method.

Atlas (bool **verbose**)
Atlas class constructor method.

const string & getAtlasPath () const
Return the path for the atlas files.

void **setAtlasPath** (const string &atlasPath)
Set the path for the atlas files.

bool **isAtlasLoaded** () const
Return true if atlas has been loaded.

void **setVerbose** (bool **verbose**)
Set the verbose boolean.

void **setIncLand** (bool incLand)
Set the inc_land boolean.

void **setIncSeaIce** (bool incSeaIce)
Set the inc_seaice boolean.

void **setIncSea** (bool incSea)
Set the inc_sea boolean.

bool **getIncLand** () const
Return the inc_land boolean.

bool **getIncSeaIce** () const
Return the inc_seaice boolean.

bool **getIncSea** () const
Return the inc_sea boolean.

bool **loadBrdfAtlas** (int month, int atlas_id=-1)
Initialise the BRDF atlas for use with any instrument.

bool **loadBrdfAtlas** (int month, **rttov::Rttov** *rttov, int atlas_id=-1)
Initialise the BRDF atlas for a specific instrument.

bool **loadBrdfAtlas** (int month, **rttov::RttovSafe** *rttov, int atlas_id=-1)
Initialise the BRDF atlas for a specific instrument.

bool **loadIrEmisAtlas** (int month, bool ang_corr=false, int atlas_id=-1)
Initialise the IR emissivity atlas for use with any instrument.

bool **loadIrEmisAtlas** (int month, **rttov::Rttov** *rttov, bool ang_corr=false, int atlas_id=-1)
Initialise the IR emissivity atlas for a specific instrument.

bool **loadIrEmisAtlas** (int month, **rttov::RttovSafe** *rttov, bool ang_corr=false, int atlas_id=-1)
Initialise the IR emissivity atlas for a specific instrument.

bool **loadMwEmisAtlas** (int month, int atlas_id=-1)
Initialise the MW emissivity atlas for use with any instrument (TELSEM2)

bool **loadMwEmisAtlas** (int month, **rttov::Rttov** *rttov, int year=0, int atlas_id=-1)
Initialise the MW emissivity atlas for a specific instrument (CNRM MW atlas)

bool **loadMwEmisAtlas** (int month, **rttov::RttovSafe** *rttov, int year=0, int atlas_id=-1)
Initialise the MW emissivity atlas for a specific instrument (CNRM MW atlas)

bool **loadMwEmisAtlas** (int month, **rttov::RttovScatt** *rttov, int year=0, int atlas_id=-1)
Initialise the MW emissivity atlas for a specific instrument (CNRM MW atlas)

bool **loadMwEmisAtlas** (int month, **rttov::RttovScattSafe** *rttov, int year=0, int atlas_id=-1)
Initialise the MW emissivity atlas for a specific instrument (CNRM MW atlas)

void **fillEmisBrdf** (double *emisBrdf, **rttov::Rttov** *rttov, const vector< int > &channels=vector< int >{})
Return emissivities/BRDFs.

void **fillEmisBrdf** (double *emisBrdf, **rttov::RttovSafe** *rttov, const vector< int > &channels=vector< int >{})
Return emissivities/BRDFs.

void **fillEmisBrdf** (double *emisBrdf, **rttov::RttovScatt** *rttov, const vector< int > &channels=vector< int >{})
Return emissivities.

void **fillEmisBrdf** (double *emisBrdf, **rttov::RttovScattSafe** *rttov, const vector< int > &channels=vector< int >{})
Return emissivities.

void **dropAtlas** ()
Deallocate memory for the atlas.

Python Atlas class

Methods:

Atlas (verbose=True)
Constructor method.

bool **loadBrdfAtlas**(month, inst=None, atlas_id=-1)
Load BRDF atlas data for specified month. Returns True if successful, False otherwise. The inst argument can be a loaded Rttov instance to initialise the BRDF atlas for a specific instrument (for faster calls).

bool **loadIrEmisAtlas**(month, inst=None, ang_corr=False, atlas_id=-1)
Load IR emissivity atlas data for specified month. Returns True if successful, False otherwise. The inst argument can be a loaded Rttov instance to initialise the BRDF atlas for a specific instrument (for faster calls).

bool **loadMwEmisAtlas**(month, inst=None, atlas_id=-1)
Load MW emissivity atlas data for specified month. Returns True if successful, False otherwise. The inst argument can be a loaded Rttov or RttovScatt instance: this is required for the CNRM atlas, but is ignored by TELSEM2.

float array **getEmisBrdf**(inst, channels=None)
Return array of emissivity/BRDF values of dimensions [nprofiles][nchannels]. The inst argument is a loaded Rttov or RttovScatt instance which has profile data associated with it. Values are returned for the supplied channel list or otherwise for all loaded channels for the instrument. Throws an execption if an error is encountered.

dropAtlas ()
Deallocate atlas data.

Members:

string **AtlasPath**
Path to the atlas data to be loaded: must be set before calling one of the "load" methods.

bool **IncLand**
If True emissivity/BRDF values are returned for profiles with land surface type; otherwise negative values are returned for such profiles. Default: True.

bool **IncSea**
If True emissivity/BRDF values are returned for profiles with sea surface type; otherwise negative values are returned for such profiles. Default: True.

bool **IncSeaIce**
If True emissivity/BRDF values are returned for profiles with sea-ice surface type; otherwise negative values are returned for such profiles. Default: True.

bool **Verbose**
Verbosity flag.

Appendix K: Enumeration types (C++)

The enumerations are defined in wrapper/rttov_common.h.

The following table lists the constants of the enumeration **rttov::gasUnitType** used to specify the profile gas_units variable in the **setGasUnits** method of the **Profile** and **ProfileScatt** classes.

Enumeration constants	**Description**
unknown	Default initialisation, ppmv over moist air will be used
ppmv_dry	Gas units of ppmv over dry air
kg_per_kg	Gas units of kg/kg over moist air
ppmv_wet	Gas units of ppmv over moist air

The following table lists the constants of the enumeration **rttov::itemIdType** used for setting gas, cloud and aerosol profiles in the **setGasItem** method of the **Profiles** and **ProfilesScatt** classes and to obtain the Jacobians for gases, aerosol and cloud profiles using the **getItemK** method of the **Rttov**, **RttovSafe**, **RttovScatt** and **RttovScattSafe** classes after running the RTTOV K model.

Enumeration constants	Description
Q, O3, CO2, N2O, CO, CH4, SO2	RTTOV variable gases
CLW	Cloud liquid water (for non-scattering MW simulations)
CFRAC	Cloud fraction for visible/IR cloud scattering simulations
STCO, STMA, CUCC, CUCP, CUMA	The 5 cloud liquid water particle types for visible/IR cloud scattering simulations.
CIRR	The ice cloud particle type for visible/IR cloud scattering simulations.
ICEDE	The ice cloud particle effective diameter input for visible/IR cloud scattering simulations.
CLWDE	The cloud liquid water particle effective diameter input for visible/IR cloud scattering simulations.
INSO, WASO, SOOT, SSAM, SSCM, MINM, MIAM, MICM, MITR, SUSO, VOLA, VAPO, ASDU	The 13 OPAC aerosol particle types for visible/IR aerosol scattering simulations.
BCAR, DUS1, DUS2, DUS3, SULP, SSA1, SSA2, SSA3, OMAT	The 9 CAMS aerosol particle types for visible/IR aerosol scattering simulations.
AER1, AER2, ..., AER30	Aerosol particle types 1-30. These are intended for use with user-generated *scaercoef* aerosol optical property files.
SCATT_HYDRO_FRAC	RTTOV-SCATT hydrometeor cloud fraction (for single cloud fraction)
SCATT_CLW, SCATT_CIW, SCATT_RAIN, SCATT_SNOW, SCATT_GRAUPEL	RTTOV-SCATT default hydrometeor types
HYDRO1, HYDRO2, ..., HYDRO30	RTTOV-SCATT arbitrary hydrometeor types 1-30 (for use with custom hydrotable files)
HYDRO_FRAC1, HYDRO_FRAC2, ..., HYDRO_FRAC30	RTTOV-SCATT cloud fractions for hydrometeor types 1-30 (for use when supplying individual cloud fractions per hydrometeor, or when using custom hydrotable files)

--END--