Python/C/C++ wrapper

for RTTOV v14

James Hocking

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

Change record			
Version	Date	Author / changed by	Remarks
0.1	01/03/2024	J Hocking	First draft for v14 beta.
1.0	04/07/2024	J Hocking	Updates following code developments
1.0.1	18/10/2024	J Hocking	Updates after Met Office review
1.0.2	06/12/2024	J Hocking	Updates after DRR

Table of contents

1. Introduction....3
2. Compilation and example code....4
3. RTTOV classes....7
   3.1. Introduction....7
   3.2. General method for calling RTTOV....9
   3.3. Setting RTTOV options....9
   3.4. Loading an instrument....10
   3.5. Specifying surface emissivities and reflectances....11
   3.6. Using the emissivity and BRDF atlases....12
   3.7. Profile data for an RttovSafe object (C++ only)....13
   3.8. Profile data for an Rttov object (C++ and Python)....15
   3.9. Specifying explicit hydrometeor/aerosol optical properties for scattering simulations....18
   3.10. Calling RTTOV....20
   3.11. Accessing RTTOV outputs....21
   3.12. Deallocating memory....22
4. Notes on thread-safety and technical implementation....23
5. Limitations of the wrapper....24
6. Description of underlying wrapper interface....25
   6.1. Loading an instrument....25
   6.2. Changing RTTOV options....29
   6.3. Using the emissivity and/or BRDF atlases....29
   6.4. Calling the RTTOV direct model....32
   6.5. Calling the RTTOV K model....36
   6.6. Calling the RTTOV direct model with explicit optical properties....38
   6.7. Calling the RTTOV K model with explicit optical properties....40
   6.8. Deallocating memory....41
   6.9. Additional wrapper routines....42
   6.10. Specific information for Python....43
   6.11. Specific information for C/C++....43
Appendix A: C++ *RttovSafe* and *Rttov* classes....44
Appendix B: Python *Rttov* class....51
Appendix C: C++ *Profile* class (used with *RttovSafe* objects)....60
Appendix D: C++ *Profiles* class (used with *Rttov* objects)....64
Appendix E: Python *Profiles* class....66
Appendix F: C++ *Options* class....71
Appendix G: Python *Options* class....79
Appendix H: C++ *Atlas* class....83
Appendix I: Python *Atlas* class....85
Appendix J: Enumeration types (C++) and constants (Python)....87
Appendix K: Gas IDs....90
Appendix L: RTTOV wrapper subroutines....91

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 that 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. Section 3 describes the **recommended** way to use the wrapper from C++ or Python: this is via the object-oriented interfaces implemented in C++ and Python. Section 4 gives some important technical information about the wrapper implementation related to thread-safety and other issues. Section 5 outlines the current limitations of the wrapper.

If you want to call the wrapper from C or create an interface in a different language, then you may need to use the underlying wrapper interface described in section 6. Section 6 is **not relevant** if you are using the C++/Python interface described in section 3.

Finally, the appendices provide additional information about the object-oriented classes and the underlying interface.

Currently the wrapper supports calls to the RTTOV direct and K models for clear-sky and scattering calculations optionally including use of the surface emissivity and BRDF atlases.

RTTOV v14.0 represents a significant update to RTTOV, and so various aspects of the wrapper interface have changed since v13 as a result. The most significant changes are:

- pressure half-levels are *nlevels* in size, and all other vertical profiles are *nlayers=nlevels-1* in size, and are provided on the pressure full-levels which interleave the half-levels.
- all classes and interfaces related to RTTOV-SCATT have been removed as scattering simulations for microwave sensors are done through the standard RTTOV classes.
- the heterogeneous surface capability means that an additional *nsurfaces* dimension has been added to the emissivity/reflectance input/output array, and to the profile surface type, near-surface, and skin arrays.
- surface emissivities and reflectances computed/used by RTTOV are now available through separate accessor methods/functions after setting a new “StoreEmisRefl” option instead of overwriting the values in the input emissivity/reflectance array.
- all option and input/output variable names conform to those in RTTOV v14: many were renamed/updated since v13.

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 netCDF library (see the user guide).

The easiest way to compile RTTOV is to edit the file build/Makefile.local to point to your netCDF 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 5.3 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.

For the object-oriented interface you need to include the relevant class definitions. The example code in the top-level wrapper/ directory demonstrates this. To call the underlying interface subroutines directly (not recommended in C++/Python) you need to include the src/wrapper/rttov_c_interface.h header file in your code and compile against the RTTOV libraries.

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.

It is recommended to use the pyrttov package which provides an object-oriented interface to RTTOV. Alternatively, to call the interface subroutines directly you can import them from rttov_wrapper_f2py, 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.

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

pyrttov_example.py	Use of pyrttov Python package for multiple instruments and use of the emissivity/BRDF atlases
pyrttov_visir_scatt_optp_example.py	Use of pyrttov for visible/IR scattering simulations where optical properties are input
pyrttov_mw_scatt_example.py	Use of pyrttov for MW scattering simulations
pyrttov_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_visir_scatt_optp_example.cpp	Use of C++ Rttov class for visible/IR scattering simulations where optical properties are input
Rttov_mw_scatt_example.cpp	Use of C++ Rttov class for MW scattering simulations
Rttov_radar_example.cpp	Use of C++ Rttov class for MW radar simulations
RttovSafe_example.cpp	Use of C++ RttovSafe class for multiple instruments including use of the emis/BRDF atlases
RttovSafe_visir_scatt_optp_example.cpp	Use of C++ RttovSafe class for visible/IR scattering simulations where optical properties are input
RttovSafe_mw_scatt_example.cpp	Use of C++ RttovSafe class for MW scattering simulations
RttovSafe_radar_example.cpp	Use of C++ RttovSafe class for MW radar simulations

Examples of calling RTTOV via underlying wrapper interface directly:

interface_example_ir_scatt_c.c	Calling IR scattering simulation in C
interface_example_ir_scatt_cpp.cpp	Calling IR scattering simulation in C++
interface_example_mw_scatt_cpp.cpp	Calling MW scattering simulation in C++
interface_example_ir_scatt.py	Calling IR scattering simulation in Python
interface_example_mw_scatt.py	Calling MW scattering simulation in Python
Makefile	Makefile to compile all the above 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
```

C++ object-oriented interface

The following files define the classes used by the C++ object-oriented interface to RTTOV (see section 3); 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.
RttovProfile.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.
RttovOptions.h/.cpp	Class representing RTTOV and wrapper options.
RttovAtlas.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/.

Python pyrttov package

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 3 provides details on using the pyrttov package.

3. RTTOV classes

3.1. Introduction

C++ object-oriented interface

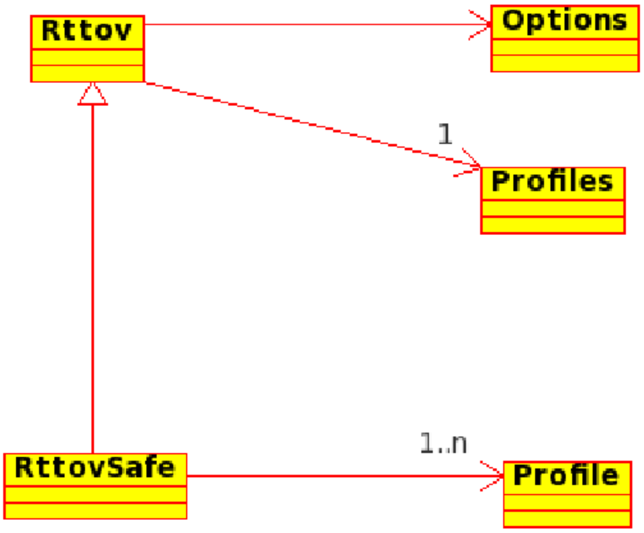
A number of C++ classes have been created in order to provide an object-oriented interface to RTTOV: **Rttov**, **RttovSafe**, **Options**, **Profiles, Profile,** and **Atlas**.

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 is associated with a **Profiles** instance using the **Rttov.setProfiles** method 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. This is a more user-friendly, but (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 contains data for 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 vector of **Profile** objects is made with the **RttovSafe.setTheProfiles** method. This method takes as argument a vector of instances of the **Profile** object. The other methods of the **RttovSafe** class are inherited from the **Rttov** class.

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 land surface emissivity and BRDF values which can be passed to an **Rttov** or **RttovSafe** object.

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 reflectance. 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 **Options, Profiles, Rttov,** 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 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, 1)
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 Python code calling the underlying interface described in section 6). Therefore the array index 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").

3.2. 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 A: 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. Appendix B gives the methods and members of the Python **Rttov** class.

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 in the following sub-sections.

3.3. 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.

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
```

The methods (C++) and members (Python) of the **Options** class are listed in Appendices F and G. Annex J of the RTTOV user guide describes the RTTOV options. The wrapper-specific options are listed in the table below.

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 you can pass any number of profiles. The wrapper will then break these down into chunks and the underlying `rttov_direct/rttov_k` subroutines are called for **NprofsPerCall** at a time until all profiles have been simulated. You may obtain improved performance (especially with multi-threaded execution) by increasing **NprofsPerCall** 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.

By default, the calls to RTTOV return the total TOA radiances and the equivalent brightness temperatures or reflectances (depending on channel wavelength). If you require access to additional RTTOV outputs you should set the **StoreEmisRefl**, **StoreTrans**, **StoreRad**, **StoreRad2**, **StoreDiagOutput**, and/or **StoreEmisTerms** options. You can then access the corresponding additional outputs via the relevant methods (C++) or members (Python) after calling RTTOV.

Wrapper Option	**Value**	**Description**
VerboseWrapper	True/False	Set to true for more verbose output from the wrapper (default false, all output suppressed except fatal error messages).
Nthreads	Integer	If <=1 RTTOV is called via the standard interface (i.e., rttov_direct/rttov_k), if >1 RTTOV is called via the parallel interface (i.e., rttov_parallel_direct/ rttov_parallel_k) using the specified number of threads (default 1).
NprofsPerCall	Integer > 0	Sets the number of profiles passed to each call to rttov_direct or rttov_k *within* the wrapper (default 1).
CheckOpts	True/False	If set to true the Fortran `rttov_user_check_options` subroutine (see user guide Annex I) is called to help ensure consistency between the selected options and the loaded coefficient file (default false).
StoreEmisRefl	True/False	Set to true to enable access to surface emissivities, BRDFs, and diffuse reflectances used in the RTTOV simulations (default false).
StoreTrans	True/False	Set to true to enable access to transmittance outputs from RTTOV calls (default false).
StoreRad	True/False	Set to true to enable access to radiance outputs from RTTOV calls (default false).
StoreRad2	True/False	Set to true to enable access to secondary radiance outputs from RTTOV calls (default false). If this is set to true then StoreRad is automatically set to true as well.
StoreDiagOutput	True/False	Set to true to enable access to diagnostic outputs from RTTOV calls (default false).
StoreEmisTerms	True/False	Set to true to enable access to the emissivity retrieval outputs from RTTOV direct model calls (default false).
RadarKAzef	True/False	Radar K input perturbations are in zef_k if false or azef_k if true (default false).

3.4. 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 hydrotable and/or aertable optical property file names should also be specified using the **setFileHydrotable** and **setFileAertable** methods respectively. For MFASIS-NN simulations the MFASIS-NN coefficient file should be specified using **setFileMfasisNN**. If using the ARO-scaling polarisation mode with MW hydrometeor scattering simulations, the **MwPolMode** must be set to ARO-scaling and the location of the look-up table can be specified using **setFileMwPol** 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.

After an instrument has been loaded the options can be changed. If you call the **myRttov.updateOptions** method and the wrapper **CheckOpts** 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 Fortran subroutine). Note that changing the coefficient filename(s) after loading the instrument will have no effect.

3.5. Specifying surface emissivities and reflectances

You should refer to user guide section 8.3 to understand how RTTOV handles surface emissivity and reflectance.

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][nsurfaces][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. Data are specified for every surface for each profile.

Where emissivity/BRDF/diffuse reflectance values in this input array are greater than or equal to zero the corresponding elements of the RTTOV calc_emis/calc_brdf/calc_diffuse_refl arrays will be set to false respectively, and the input values of the surface parameters will be used for the simulations. Where the emissivity/BRDF/diffuse reflectance values in **SurfEmisRefl** are less than zero the corresponding elements of the RTTOV calc_emis/calc_brdf/calc_diffuse_refl 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 land surface emissivity and BRDF: this is described in the next section.

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).

In RTTOV v13 and earlier, the **SurfEmisRefl** array was overwritten during the wrapper call with the emissivity/reflectance values used in the simulations. This is no longer the case: the **SurfEmisRefl** array remains unmodified after calling RTTOV. To access the emissivities/reflectances used in the simulations, set the **StoreEmisRefl** option to true (see section 3.3) and use the **getSurfEmis, getSurfBrdf, getDiffuseRefl** methods (C++) or **SurfEmis, SurfBrdf, SurfDiffuseRefl** members (Python) to access the values.

It is not mandatory to specify/set the **SurfEmisRefl** array. If you do not then it is equivalent to setting calc_emis, calc_brdf, and calc_diffuse_refl 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 **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
```

3.6. 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. Any combination of atlases and months can be used: each **Atlas** object is independent. The methods and members of the **Atlas** class are given in Appendices H and I. 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 **Atlas** 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 three-dimensional array of size *[nprofiles][nsurfaces][nchannels]*. 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 three-dimensional array of size *[nprofiles][nsurfaces][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 (see sections 3.7 and 3.8). 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 resulting arrays are populated for all surfaces associated with each profile.

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 Boolean 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/surfaces 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/surfaces 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/surfaces.

The **MaxDistance** member can be set to a positive value: in this case, if an IR emissivity or BRDF atlas doesn't have emissivity/BRDF data at the specified lat/lon location, the atlas will return a nearby valid value (if one exists) within the specified distance of the original location. Units are km.

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.

3.7. 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 C.

A **Profile** object is instantiated as follows, where *nlevels* is the number of pressure half-levels and *nsurfaces* is the number of surfaces:

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

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

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 J. If unspecified the default is kg/kg over moist air, but a warning is printed if you do not set this explicitly.

Pressure half-levels are mandatory (set using the **setPHalf** method), but, as in RTTOV itself, the pressure full-levels (**setP**) do not need to be specified.

If *nsurtfaces>1* then the **setSurfaceFraction** method must be called to specify the surface coverage fraction for surface indices 1,...,*nsurfaces-1*. This should not be called if *nsurfaces=1*.

The **setAngles**, **setSurfGeom** and **setDateTimes** methods must all be called for every **Profile** instance, and the **setNearSurface**, **setSkin, setSurfType** methods must be called for every surface for every **Profile**. 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 3.8 lists the variables that must be specified in each array (the order of the variables is important).

The **setSimpleCloud**, **setClwdeParam**, **setIcedeParam**, **setHydroFracEff** 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.

For aerosol simulations, you can specify individual aerosol concentration profiles by their index using the **setAerN** method. The wrapper is limited to 30 aerosol species. This method can be used with the supplied NWP SAF OPAC and CAMS aertable files, but methods are provided for each OPAC and CAMS species based on the aerosol name (e.g., **setInso**), and these may be used instead. There are currently no named methods for the ICON-ART aertable files, so the **setAerN** method must be used in that case. Any unspecified aerosol type will have concentrations set to zero. The aerosol concentration units are specified via the **setMmrAer** method which defaults to true (kg/kg).

The **setAerClimProf** method can be used to generate climatological profiles based on the OPAC components for a selection of different scenarios. This must be used with an OPAC aertable file, and the other profile variables (in particular gas units, aerosol units, pressure half-levels, temperature, water vapour, SurfGeom) must be specified before this method is called. The argument is an index 1-10 for the required climatological scenario. See the rttov_aer_clim_prof subroutine in Annex I of the RTTOV user guide for more information. The method adds the relevant aerosol profiles to the **Profile** object. It is important not to change the aerosol units (via **setMmrAer**) *after* this method has been called.

For hydrometeor simulations, you can specify individual hydrometeor concentration profiles by their index using the **setHydroN** method. The wrapper is limited to 30 hydrometeor species. This method can be used with the NWP SAF UV/VIS/IR and MW hydrotables, but methods are provided for each individual hydrometeor type in these hydrotable files. In most cases you must specify a single hydro (cloud) fraction profile: this can be done via the **setHydroFrac** method. If you are specifying per-hydrometeor fractions (if the **PerHydroFrac** option is set to true) then you can use the **setHydroFracN** method. Finally, for UV/VIS/IR species that have an explicit dependence on particle size, you can input the particle effective diameter profiles using the **setHydroDeffN** method. In addition, two methods (**setClwDeff** and **setIceBaumDeff**) are provided for the NWP SAF UV/VIS/IR hydrotable particle types with Deff-dependence. If the concentration, fraction, or Deff profiles are unspecified for a hydrometeor type, the corresponding profile values are set to zero. The hydrometeor concentration units are specified via the **setMmrHydro** method which defaults to true (kg/kg).

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. All profiles must have the same number of levels and surfaces (*nlevels* and *nsurfaces*), and the gas, aerosol, and hydrometeor units must be the same for all profiles. Once the collection of **Profile** instances is fully populated it is associated with the **RttovSafe** instance by calling the **myRttov.setTheProfiles** method. This method makes the following checks to help prevent errors**:**

- 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 and surfaces
- 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 hydrometeors, gas_units, mmr_hydro, mmr_aer)
- 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 optional fields have not been set initialise them with default values.

It is very important that ***all*** profile data are associated with the **Profile** objects ***before*** they are associated with the **RttovSafe** instance.

3.8. 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 containing data for all profiles to be simulated. 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 Appendices D and E.

A **Profiles** object is instantiated as follows, where *nprofiles* is the number of profiles, *nlevels* is the number of pressure half-levels, and *nsurfaces* is the number of surfaces.

In C++:

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

In Python:

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

In C++ the data for each profile variable is provided to the Profiles instance as a pointer to an array containing the data for all profiles 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.

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

In Python, the **gasUnitType** class can be used to specify the units rather than using integer literals (see Appendix J).

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.

Pressure half-levels are provided via an array of size *[nprofiles][nlevels]*. All other vertical profile variables (temperature, gas concentrations, aerosol and hydrometeor variables) are provided via arrays of size *[nprofiles][nlayers]*. As in RTTOV itself, specification of pressure full-levels is optional.

If *nsurtfaces>1* then the **setSurfaceFraction** method must be called in C++ to specify the surface coverage fraction for surface indices 1,...,*nsurfaces-1* for every profile (array dimensions *[nprofiles][nsurfaces-1]*). In Python, the array is assigned directly to the **SurfaceFraction** member. Surface fraction should not be specified if *nsurfaces=1*.

In C++ the **setAngles**, **setNearSurface**, **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 or three dimensional array (see Appendices D and E). The first dimension is *nprofiles*, and the final 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**, **setClwdeParam**, **setIcedeParam**, **setHydroFracEff** 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**, **NearSurface**, **SimpleCloud**, etc) are assigned for each **Profiles** instance rather than via a method call.

In C++ to supply the hydrometor 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 J. (You can also set the gas profiles using this method, but it is clearer to use the methods like **setQ** which are specific to each gas).

In Python there is no equivalent to **setGasItem**. For aerosol simulations, you can specify individual aerosol concentration profiles by their index using the **setAerN** method or by assigning directly to the **Aer*N*** members where *N=1,2,…,30.* The wrapper is limited to 30 aerosol species. Either approach can be used with the supplied NWP SAF OPAC and CAMS aertable files, but members are provided for each OPAC and CAMS species based on the aerosol name (e.g., **Inso**), and profiles may be assigned directly to these instead. There are currently no named members for the ICON-ART aertable files, so the **setAerN** method or **Aer*N*** members must be used in that case. Any unspecified aerosol type will have concentrations set to zero.

The aerosol concentration units are specified via the **setMmrAer** method (C++) or **MmrAer** member (Python) which default to true (kg/kg).

The **setAerClimProf** method (in both C++ and Python) can be used to generate climatological profiles based on the OPAC components for a selection of different scenarios. This must be used with an OPAC aertable file, and the other profile variables (in particular gas units, aerosol units, pressure half-levels, temperature, water vapour, SurfGeom) must be specified before this method is called. The argument is an array of integer indices 1-10 for the required climatological scenario for each profile. See the rttov_aer_clim_prof subroutine in Annex I of the RTTOV user guide for more information. The method adds the relevant aerosol profiles to all profiles in the **Profiles** object. It is important not to change the aerosol units (via **setMmrAer/MmrAer**) *after* this method has been called.

For hydrometeor simulations in Python, you can specify individual hydrometeor concentration profiles by their index using the **setHydroN** method or by assigning directly to the **Hydro*N*** members where *N=1,2,...,30.* . The wrapper is limited to 30 hydrometeor species. Either approach can be used with the NWP SAF UV/VIS/IR and MW hydrotables, but members are provided for each individual hydrometeor type in these hydrotable files and these may be used instead. In most cases you must specify a single hydro (cloud) fraction profile: this can be done via the **HydroFrac** member. If you are specifying per-hydrometeor fractions (if the **PerHydroFrac** option is set to true) then you can use the **setHydroFracN** method or assign to the **HydroFrac*N*** members where *N=1,2,...,30*. Finally, for UV/VIS/IR species that have an explicit dependence on particle size, you can input the particle effective diameter profiles using the **setHydroDeffN** method or assign to the **HydroDeff*N*** members where *N=1,2,...,30*. In addition, two members (**ClwDeff** and **IceBaumDeff**) are provided for the NWP SAF UV/VIS/IR hydrotable particle types with Deff-dependence. If the concentration, fraction, or Deff profiles are unspecified for a hydrometeor type, the corresponding profile values are set to zero.

The hydrometeor concentration units are specified via the **setMmrHydro** method (C++) or **MmrHydro** member (Python) which default to true (kg/kg).

Once all the necessary profile data have been specified in the **Profiles** instance it can be associated with the **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. It is very important that ***all*** profile data are associated with the **Profiles** object ***before*** it is associated with the **Rttov** instance. 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.

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, scattering, etc) Unused variables can be set to zero.

Array	**Type**	**Dimensions***	**Mandatory/ Optional**	**Variable list** (names refer to Fortran rttov_profile derived type members)
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
SurfaceFraction	Real	[nprofiles] [nsurfaces-1]	Mandatory if nsurfaces>1	Surface coverage fractions for surface indices 1, ..., nsurfaces-1. Do not specify if nsurfaces=1.
SurfGeom	Real	[nprofiles][3]	Mandatory	(latitude, longitude, elevation) per profile
SurfType	Integer	[nprofiles] [nsurfaces][2]	Mandatory	(skin%surftype, skin%watertype) per surface per profile
Skin	Real	[nprofiles] [nsurfaces][9]	Mandatory	(skin%t, skin%salinity, skin%snow_fraction, skin %foam_fraction, skin%fastem(1:5)) per surface per profile
NearSurface	Real	[nprofiles] [nsurfaces][5]	Mandatory	(t2m, q2m, wind_u10m, wind_v10m, wind_fetch) per surface per profile
SimpleCloud	Real	[nprofiles][2]	Optional	(ctp, cfraction) per profile
ClwdeParam	Integer	[nprofiles]	Optional	clwde_param per profile
IcedeParam	Integer	[nprofiles]	Optional	icede_param per profile
HydroFracEff	Real	[nprofiles]	Optional	hydro_frac_eff 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.*

3.9. Specifying explicit hydrometeor/aerosol optical properties for scattering simulations

This section applies to scattering simulations where explicit optical property profiles are provided rather than using pre-defined optical properties contained in aertable or hydrotable files. This is described in section 8.4.11 of the user guide: you should read that section in order to understand the RTTOV scattering options and inputs. It also describes the limitations associated with explicit optical property inputs.

These simulations are activated by setting the **Hydrometeors** or **Aerosols** (or both) options to true and the corresponding **UserHydroOptParam** or **UserAerOptParam** options to true. In this case the **FileHydrotable** or **FileAertable** members of **Rttov/RttovSafe** do not need to be specified.

Separate optical property inputs are available for hydrometeors and aerosols. The optical properties are provided in the same way for both. The only difference is that for hydrometeor simulations you must specify a profile of hydro fractions (**HydroFrac**) 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 hydrometeors. Also note that you can specify explicit optical properties for hydrometeors and use the pre-defined aerosol particle types from an aertable file (as described in sections 3.7 and 3.8) 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. See the RTTOV user guide for full information about these inputs, their units, and when each is required.

Argument	**Type**	**Description**
esba[4][nprofiles][nchannels][nlayers]	Real	Extinction coefficients (esba(1,:,:,:)), single-scattering albedos (esba(2,:,:,:)), bpr parameters (esba(3,:,:,:)), and asymmetry parameters (esba(4,:,:,:)). See below for how to calculate bpr and asymmetry parameters.
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 the **Solar** option 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 the **Solar** option is true (i.e., when solar radiation is included).
lcoef[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 required for 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 Appendices A and B. The only mandatory input is the *esba* array containing the extinction coefficients, single-scattering albedos, bpr parameters (Chou-scaling solver), and asymmetry parameter (delta-Eddington solver). This is assigned to the **Rttov**/**RttovSafe** object using the **setHydroEsba**/ **setAerEsba** methods (C++) or directly assigning to the **HydroEsba**/**AerEsba** members (Python). The extinction coefficients and single-scattering albedos 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. For any channels for which delta-Eddington is not being used the asymmetry parameter values may be zero. In the case where either Chou-scaling or delta-Eddington is being used and solar radiation is not included no other optical property inputs need to be specified.

If the DOM solar solver is used 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 **setHydroPha/setAerPha** method. In Python the phase angles and phase functions are assigned directly to the **HydroPhangle**/**AerPhangle** and **HydroPha**/**AerPha** members.

If the DOM thermal or solar solvers are being used you must specify the Legendre coeffiicients corresponding to the phase functions: this applies to all channels being simulated via DOM. In C++ the **setHydroLcoef/setAerLcoef** method is used and in Python the coefficients are assigned directly to the **HydroLcoef/AerLcoef** 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 hydrometeors/aerosol the phase function values and Legendre coefficients can be zero.

RTTOV provides subroutines to calculate bpr and asymmetry parameters, and Legendre coefficients from phase functions: this is achieved via the **calcBpr**, **calcAsym**, and **calcLcoef** methods (C++ and Python) whose interfaces are described in Appendices A and B. 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**.

3.10. 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 K perturbation is set to 1 for brightness temperatures, reflectances, and radiances in all channels. The perturbations used in each channel depend on the **ADKBT** and **ADKRefl** options. See the user guide for details about the K model.

For radar simulations, the brightness temperature and radiance perturbations are set to zero and, by default, the **ZefK** input perturbations are set to 1 for all levels for all channels. This is usually not desirable. As described in the RTTOV user guide, to reconstruct the full radar reflectivity Jacobian it is necessary to call the K model multiple times, providing input reflectivity perturbations in one layer at a time. You can do this by specifying the reflectivity perturbation manually via the **setZefK** method (C++) or the **ZefK** member (Python). The **RadarKAzef** wrapper option determines whether the input K perturbations are applied to attenuated reflectivities (true) or unattenuated reflectivities (false).

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

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 **NprofsPerCall** option. 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 **NprofsPerCall** 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.

3.11. Accessing RTTOV outputs

Once RTTOV has been called the output data can be accessed by calling various methods (C++) or accessing corresponding members (Python). 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. In Python these are accessed via the **Rads** and **BtRefl** members of the **Rttov** class. For radar simulations, the output reflectivities are available via **getZef** and **getAZef** methods (C++) or **Zef** and **AZef** members (Python).

It is also possible to access the surface emissivities/reflectances used in the simulations and the full contents of the RTTOV transmission, radiance, radiance2, diagnostic output, and emissivity retrieval structures (so long as those member arrays were output by the simulations). You must set the relevant wrapper option (**StoreEmisRefl**, **StoreTrans**, **StoreRad**, **StoreRad2**, **StoreDiagOutput**, **StoreEmisTerms**) 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 (or surface for emissivity/reflectance outputs) indices while in Python you can access the full output array for all channels/profiles/surfaces (as relevant). The relevant methods and members are listed in Appendices A and B.

After calling the RTTOV K model the Jacobians can be obtained through the various methods/ members listed in Appendices A and B. 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][nlayers]*).

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.

In C++, to return the Jacobians for gas profiles and (if computed) for hydrometeors 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 J. 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 and an **itemIdType** class that can be used to obtain the relevant IDs (Appendix J) but it is easier (and recommended) to reference each Jacobian directly as **myRttov**.**QK** (water vapour Jacobian), for example.

For Python aerosol simulations, the aerosol concentration Jacobians for each aerosol index can be accessed via the **getAerNK** method or via the **Aer*N*K** members where *N=1,2,…,30.* In addition, for OPAC and CAMS aertable files, there are named members for Jacobians of each aerosol species (e.g., **InsoK**).

For Python hydrometeor simulations, the hydrometeor concentration Jacobians for each hydrometeor index can be accessed via the **getHydroNK** method or via the **Hydro*N*K** members where *N=1,2,…,30.* In addition, for NWP SAF UV/VIS/IR and MW hydrotable files, there are named members for Jacobians of each hydrometeor type. For simulations with a single hydro fraction input profile (**HydroFrac**), the corresponding Jacobian can be accessed via **HydroFracK**. Per-hydrometeor hydro fraction Jacobians are available via the **HydroFrac*N*K** members where *N=1,2,…,30.* Similarly for hydrometeor types with explicit dependence on particle size, the Deff Jacobians are available via the **HydroDeff*N*K** members where *N=1,2,…,30.* In addition, two members (**ClwDeffK** and **IceBaumDeffK**) are provided for Jacobians of the NWP SAF UV/VIS/IR hydrotable particle types with Deff-dependence.

The additional profile variables which are active in the Jacobian model can be accessed via the **getNearSurfaceK**, **getSkinK**, **getSimpleCloudK**, etc methods (C++) or the **NearSurfaceK**, **SkinK**, **SimpleCloudK**, etc members (Python). The order of the variables in these output vectors or arrays is the same as for the corresponding input arrays. The **SurfaceFractionK** Jacobian should only be accessed if *nsurfaces>1.*

For simulations with explicit optical property inputs, Jacobians of the extinction coefficients, single-scattering albedos, bpr, and asymmetry parameters are available. In C++ these are accessed via the **getHydroExtK**, **getAerSsaK**, etc members for hydrometeors and aerosols. In Python these are accessed all together via the **AerEsbaK** and **HydroEsbaK** members which return arrays of size *[4][nprofiles][nchannels][nlayers]* corresponding to the input *esba* optical property arrays.

3.12. 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.

4. 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 (these subroutines are described in section 6). 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 the loaded instruments (and, separately, the loaded atlases) via a linked list. When a constructor (destructor) is called, the object is added to (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 unless you are very careful.

Furthermore, you must take care (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.

To guard against accidental calls to destructors, the C++ **Rttov** and **Atlas** copy and assignment constructors are private.

5. Limitations of the wrapper

The wrapper currently has the following limitations:

- For emissivity/BRDF atlases, additional optional outputs are not available (for example standard deviation/covariance data and quality flags cannot currently be accessed).
- Not all ancillary user-level Fortran subroutines available in the RTTOV package are available through the wrapper. These include the *rttov_get_sea_emis*, *rttov_get_sea_brdf*, *rttov_calc_geo_sat_angles*, *rttov_calc_solar_angles*, *rttov_scale_ref_gas_prof*, and *rttov_emissivity_retrieval* subroutines, and the routines in the *rttov_zutility_mod* module.
- Aerosol simulations with custom aertable files are supported up to a maximum of 30 aerosol species.
- Hydrometeor simulations with custom hydrotable files are supported up to a maximum of 30 hydrometeor types.
- PC-RTTOV is unavailable.
- TL/AD models are unavailable.

6. Description of underlying wrapper interface

The recommended way to call the interface from Python or C++ is via the classes which are described in section 3. The details of the underlying interface described in this section are purposefully hidden from the user so the classes are a more user-friendly way of calling RTTOV. If you wish to call RTTOV from C (or other languages) you most likely need to use the interface described in this section.

The following sub-sections describe the interface in general terms: for Python and C/C++, the interfaces are very similar. To understand the wrapper interface itself you should read this and then refer to the final sections 6.10 and 6.11 below which contain information specific to Python and C/C++ respectively. Appendix L 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 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.

You should refer to section 4 for information on thread-safety in the wrapper, and section 5 for limitations of the wrapper.

6.1. Loading an instrument

The `rttov_load_inst` subroutine is used to load an instrument. In this call you provide a string containing the coefficient and optical property filename(s) to load (the “rtcoef” file is mandatory, and other filenames are optional depending on the simulations being run), 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 7.5 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.

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*** within 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_o3co2.dat ' \
  'opts%rt_all%o3_data 1 ' \
  'opts%rt_all%solar 1 '   \
  'nthreads 4 '
```

NB The space separation between options is important and there must be no spaces within 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_aertable	Full path to aerosol optical property file	For aerosol simulations, path to aertable file.
file_hydrotable	Full path to hydrometeor optical property file	For hydrometeor simulations, path to hydrotable file.
file_mfasis_nn	Full path to MFASIS-NN file	For hydrometeor simulations using MFASIS-NN solar solver.
file_mw_pol	Full path to ARO-scaling polarisation look-up table	For MW hydrometeor 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 J) can be set in the options string. The key value is given as in the table in Annex J 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_all%solar	0 or 1	Turn solar radiation off/on.
opts%scatt%hydrometeors	Integer 1-5	Set interpolation mode.
opts%scatt%thermal_solver	Integer 1-3	Set thermal scattering solver.

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 (i.e., rttov_direct/rttov_k), if >1 RTTOV is called via the parallel interface (i.e., rttov_parallel_direct/ rttov_parallel_k) 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_check_options` subroutine (see user guide Annex I) is called to help ensure consistency between the selected options and the loaded coefficient file (default 0).

store_emis_refl	0 or 1	Set to 1 to enable access to surface emissivities, BRDFs, and diffuse reflectances used in the RTTOV simulations (default 0).
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 calls (default 0).
store_rad2	0 or 1	Set to 1 to enable access to secondary radiance outputs from RTTOV calls (default 0). If this is set to 1 then store_rad is automatically set to 1 as well.
store_diag_output	0 or 1	Set to 1 to enable access to diagnostic outputs from RTTOV calls (default 0).
store_emis_terms	0 or 1	Set to 1 to enable access to the emissivity retrieval outputs from RTTOV direct model calls (default 0).
radar_k_azef	0 or 1	Radar K input perturbations are in zef_k if 0 or azef_k if 1 (default 0).

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 outputs you should set the store_emis_refl, store_trans, store_rad, store_rad2, store_diag_output, and/or store_emis_terms options. You can then use the subroutines listed in Appendix L to access this data after calling RTTOV. See the user guide for more information on RTTOV outputs.

If you are performing hydrometeor or aerosol scattering simulations with optical properties from hydrotable or aertable files you must ensure the hydrometeors and/or aerosols RTTOV options are set to true and the paths to the required optical property file(s) are specified in the options string when loading the instrument. If you wish to carry out MFASIS-NN simulations you must set the path to the MFASIS-NN coefficient file in the options string in addition to the hydrotable file. When using the ARO-scaling polarisation option, the pol_mode must be set to this option and the full path to the polarisation look up table must be provided.

6.2. 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 property files if those 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 I):

```
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.

6.3. 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. Full 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 subroutines 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, camel_version, year)
rttov_load_mw_emis_atlas(atlas_wrap_id, path, month, atlas_id, inst_id, year)
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 (1-12)	in	Month 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).
camel_version	Integer (2 or 3)	in	IR atlas only: specifies version of CAMEL single-year or climatology atlas (v2 or v3), ignored for UWIRemis atlas.
year	Integer	in	Emissivity atlas only: specifies year for CAMEL v3 single-year atlas, or for CNRM atlas. See user guide. Ignored for other atlases.

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 single-year 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,    &
                   max_distance,             &
                   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.
max_distance	Real	in	Maximum distance (km) within which to search for a valid emissivity/BRDF if no value is found at given location (IR emissivity and BRDF atlases only, set to 0 for no search).
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 J and table 7.2 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 7.5).

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.

You can specify a positive value in the max_distance argument to enable a search for a nearby valid emissivity/BRDF if there is no value at the given location (IR emissivity and BRDF atlases only). See the user guide for more information on this feature.

6.4. 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,                    &
                  surface_fraction, surftype, skin, near_surface, &
                  simplecloud, clwde_param, icede_param,          &
                  hydro_frac_eff, zeeman,                         &
                  p_half, p, t,                                   &
                  gas_units, mmr_hydro, mmr_aer,                  &
                  gas_id, gases, surfemisrefl,                    &
                  btrefl, rads,                                   &
                  nchannels, ngases, nlevels, nprofiles, nsurfaces)
```

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.
surface_fraction(nprofiles, nsurfaces-1)	Real	in	Surface coverage fraction for surface indices 1, ..., nsurfaces-1 for each profile. If nsurfaces=1 this is of size 0.
surftype(nprofiles,nsurfaces,2)	Integer	in	(skin%surftype, skin%watertype) for each surface for each profile.
skin(nprofiles,nsurfaces,9)	Real	in	(skin%t, skin%salinity, skin%snow_fraction, skin %foam_fraction, skin%fastem(1:5)) for each surface for each profile.
near_surface(nprofiles,nsurfaces,5)	Real	in	(t2m, q2m,wind_u10m, wind_v10m, wind_fetch) for each surface for each profile.
simplecloud(nprofiles,2)	Real	in	(ctp, cfraction) for each profile.
clwde_param(nprofiles)	Integer	in	clwde_param for each profile.
icede_param(nprofiles)	Integer	in	icede_param for each profile.
hydro_frac_eff(nprofiles)	Real	in	hydro_frac_eff for each profile.
zeeman(nprofiles,2)	Real	in	(Be, cosbk) for each profile.
p_half(nprofiles,nlevels)	Real	in	Pressure half-levels for each profile.
p(nprofiles,nlayers)	Real	in	Pressure full-levels for each profile, may be zero in which case RTTOV calculates them internally.
t(nprofiles,nlayers)	Real	in	Temperature on pressure full-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_hydro	Integer	in	Set profile mmr_hydro flag for hydrometeor units: non-zero=>kg/kg; 0=>g/m^3
mmr_aer	Integer	in	Set profile mmr_aer flag for aerosol units: non-zero=>kg/kg; 0=>cm^-3
gas_id(ngases)	Integer	in	List of IDs for gases, aerosol and hydrometeor profiles present in the gases array, see below.
gases(ngases,nprofiles,nlayers)	Real	in	Gas, aerosol and hydrometeor profiles on full-levels for each profile: must contain at least water vapour profiles, see below.
surfemisrefl(5,nprofiles,nsurfaces, nchannels)	Real	in	Input surface emissivity, BRDF, diffuse reflectance, specularity, and effective Tskin values for each channel for each surface for each profile.
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).
nsurfaces	Integer	in	Number of surfaces associated with each profile (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 7.5).

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 2/3/4-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 J and table 7.2 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 section 8.3 to understand how RTTOV treats surface emissivity and reflectance.

The surfemisrefl(0,:,:), surfemisrefl(1,:,:), and surfemisrefl(2,:,:) arrays are used to control the input or calculation of surface emissivities, BRDFs, and diffuse reflectances respectively for all channels for each profile. If you provide non-negative (i.e. >=0) values for any channel then the corresponding elements of calc_emis, calc_brdf, and/or calc_diffuse_refl 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(0/1/2,:,:) is negative then calc_emis/calc_brdf/calc_diffuse_refl 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(3,:,:) array is used to specify the surface specularity which is only used with the RTTOV lambertian option. Finally, the surfemisrefl(4,:,:) array is used to specify the per-channel effective skin temperatures which are used only with the use_tskin_eff option.

Specifying gas, aerosol and hydrometeor profiles

RTTOV coefficient files support varying numbers of trace gases (see section 3 of the user guide). In addition, scattering simulations using pre-defined optical properties (stored in aertable and/or hydrotable files, see user guide section 8.4) require one or more profiles of hydrometeor and aerosol concentrations, one or more hydro (cloud) fraction profiles for hydrometeor simulations, and optionally hydro Deff profiles for UV/VIS/IR hydrometeor simulations. Any or all of these are supplied to the interface using the gases array.

The list of gas, aerosol and hydrometeor inputs you are passing into RTTOV are specified in the gas_id array. There is one element per input variable which should contain the corresponding ID for that variable (see appendix K 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.

As an example, suppose we wish to run an IR hydrometeor simulation with the STCO liquid water cloud and Baum ice cloud types. Water vapour is always mandatory and the hydrometeor simulations also require a hydro fraction profile. Then the gas_id and gases arrays could be specified as follows (pseudo-code):

```
# ngases = 4, for gas IDs see appendix K:
# 1=>q, 201=>hydro_frac, 301=>STCO (hydro type 1), 307=>ice cloud (hydro type 6)
gas_id[:] = [1, 201, 301, 307]

# water vapour
gases[0, 0:nprofiles, 0:nlayers] = q[0:nprofiles, 0:nlayers]

# hydro_frac
gases[1, 0:nprofiles, 0:nlayers] = hydro_frac[0:nprofiles, 0:nlayers]
```

```
# STCO liquid water cloud
gases[2, 0:nprofiles, 0:nlayers] = strat_cont[0:nprofiles, 0:nlayers]

# Baum ice cloud
gases[3, 0:nprofiles, 0:nlayers] = ice_cloud[0:nprofiles, 0:nlayers]
```

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 the emissivities/reflectances used in the simulations and to all of the RTTOV radiance, transmittance, and other outputs, assuming the relevant wrapper options were set (store_emis_refl, store_rad, store_rad2, store_trans, store_diag_output, store_emis_terms): see section 6.1 and appendix L.

6.5. 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,                         &
             surface_fraction, surface_fraction_k, surftype,      &
             skin, skin_k, near_surface, near_surface_k,          &
             simplecloud, simplecloud_k, clwde_param, icede_param, &
             hydro_frac_eff, hydro_frac_eff_k, zeeman,            &
             p_half, p_half_k, p, p_k, t, t_k,                    &
             gas_units, mmr_hydro, mmr_aer,                       &
             gas_id, gases, gases_k, surfemisrefl, surfemisrefl_k, &
             btrefl, rads, bt_k, rads_k, zef_k,                   &
             nchannels, ngases, nlevels, nprofiles, nsurfaces)
```

Argument	Type	Intent	Description
surface_fraction_k(nprofiles, nsurfaces,nchannels)	Real	inout	Calculated Jacobians for surface fractions. The values for the final surface index (nsurfaces) are always zero because the surface fraction for the final surface is not specified by the user.
skin_k(nprofiles,nsurfaces, 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.*
near_surface_k(nprofiles,nsurfaces, nchannels,5)	Real	inout	Calculated Jacobians for (t2m, q2m,wind_u10m, wind_v10m, wind_fetch).
simplecloud_k(nprofiles,nchannels,2)	Real	inout	Calculated Jacobians for (ctp, cfraction).

hydro_frac_eff_k(nprofiles, nchannels)	Real	inout	Calculated Jacobians for hydro_frac_eff.
p_half_k(nprofiles,nchannels,nlevels)	Real	inout	Calculated Jacobians for pressure half-levels.
p_k(nprofiles,nchannels,nlayers)	Real	inout	Calculated Jacobians for pressure full-levels.
t_k(nprofiles,nchannels,nlayers)	Real	inout	Calculated Jacobians for temperature.
gases_k(ngases,nprofiles,nchannels, nlayers)	Real	inout	Calculated Jacobians for gas, aerosols and hydrometeors, variable order matches the input gas_id and gases arrays, see above.
surfemisrefl_k(5,nprofiles,nsurfaces, nchannels)	Real	inout	Calculated Jacobians for surface emissivity, BRDF, diffuse reflectance, specularity, and effective Tskin.
btrefl_k(nprofiles,nchannels)	Real	in	Input BT (channels are wavelengths > 3µm) or reflectance (wavelengths < 3µm) perturbations.
rads_k(nprofiles,nchannels)	Real	in	Input radiance perturbations.
zef_k(nprofiles,nchannels,nlayers)	Real	in	Input radar reflectivity perturbations.

Notes:

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

For radar simulations, you would typically want to set btrefl_k and rads_k to zero, and specify the reflectivity perturbations in zef_k (potentially for one layer at a time: see the user guide). The radar_k_azef wrapper option determines whether the input perturbations are applied to attenuated (true) or unattenuated (false) reflectivities.

The user guide advises 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.

6.6. Calling the RTTOV direct model with explicit optical properties

Section 8.4.11 of the user guide describes scattering simulations with explicit optical properties. For scattering simulations using only pre-defined optical properties in aertable/hydrotable files see section 6.4 above. When calling this interface either opts%scatt%hydrometeors or opts%scatt %aerosols (or both) must be true and the corresponding opts%scatt%user_hydro_opt_param or opts %scatt%user_aer_opt_param (or both) must be true. You can use pre-defined optical properties from the relevant optical property file for hydrometeors or aerosols and supply explicit optical properties for the other via this interface: follow the procedure described in section 6.4 above for the pre-defined hydrometeor/aerosol optical properties. The interface is as follows:

```
rttov_call_direct_scatt_optp(                             &
    err, inst_id, channel_list,                           &
    datetimes, angles, surfgeom,                          &
    surface_fraction, surftype, skin, near_surface,       &
    clwde_param, icede_param, hydro_frac_eff, zeeman,     &
    p_half, p, t,                                         &
    gas_units, mmr_hydro, mmr_aer, gas_id, gases,         &
    aer_phangle, aer_esba, aer_lcoef, aer_pha,            &
    hydro_phangle, hydro_esba, hydro_lcoef, hydro_pha,    &
    surfemisrefl, btrefl, rads,                           &
    nchannels, ngases, nlevels, nprofiles, nsurfaces,     &
    aer_nphangle, aer_nmom, hydro_nphangle, hydro_nmom)
```

This subroutine call is similar to `rttov_call_direct` except for the additional optical property inputs. The simplecloud input is not present because it does not pertain to full scattering simulations. All other common inputs are identical.

There are additional optical parameter inputs: these are provided separately for aerosols and hydrometeors. 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. The inputs are described in the table below are for aerosols: the hydrometeor ones are identical. See the RTTOV user guide for full information about these inputs, their units, and when each is required.

Argument	**Type**	**Description**
aer_esba(4,nprofiles,nchannels, nlayers)	Real	Extinction coefficients (aer_esba(1,:,:,:)), singe-scattering albedos (aer_esba(2,:,:,:)), bpr parameters (aer_esba(3,:,:,:)), and asymmetry parameters (aer_esba(4,:,:,:)). See below for how to calculate bpr and asymmetry parameters.
aer_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).
aer_phangle(aer_nphangle)	Real	Angle grid on which phase functions are defined (degrees). First value must be 0° and final value must be 180°.
aer_pha(nprofiles,nchannels, nlayers,aer_nphangle)	Real	Azimuthally-averaged phase functions normalised such that the integral over all scattering angles is 4π. Phase functions are only used when solar radiation is active, and are only required for solar-affected channels.

aer_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).
aer_lcoef(nprofiles,nchannels, nlayers,aer_nmom+1)	Real	Legendre coefficients corresponding to each phase function. Note the final dimension is aer_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 hydrometeor simulations you must always supply a hydro fraction profile: this is done via the “gases” input array as described in section 6.4.

For layers containing no hydrometeor/aerosol (i.e., where the corresponding extinction is zero), all other parameters including the phase function values and Legendre coefficients can be zero.

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

Wrappers are provided for the RTTOV subroutines which calculate bpr values, asymmetry parameters, 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_asym(err, phangle, pha, asym, 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.
asym	Real	out	Calculated asymmetry parameter.
nphangle	Integer	in	Number of angles on which phase functions are defined (not required in Python).

```
rttov_lcoef(err, phangle, pha, lcoef, 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.
lcoef(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).

6.7. 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 6.5 above.

```
rttov_call_k_scatt_optp(                                                  &
    err, inst_id, channel_list,                                           &
    datetimes, angles, surfgeom,                                          &
    surface_fraction, surface_fraction_k, surftype,                       &
    skin, skin_k, near_surface, near_surface_k,                           &
    clwde_param, icede_param, hydro_frac_eff, hydro_frac_eff_k, zeeman,  &
    p_half, p_half_k, p, p_k, t, t_k,                                     &
    gas_units, mmr_hydro, mmr_aer, gas_id, gases, gases_k,               &
    aer_phangle, aer_esba, aer_esba_k, aer_lcoef, aer_pha,               &
    hydro_phangle, hydro_esba, hydro_esba_k, hydro_lcoef, hydro_pha,     &
    surfemisrefl, surfemisrefl_k,                                         &
    btrefl, rads, bt_k, rads_k, zef_k,                                    &
    nchannels, ngases, nlevels, nprofiles, nsurfaces,                     &
    aer_nphangle, aer_nmom, hydro_nphangle, hydro_nmom)
```

The K variables are exactly the same as those described in section 6.5 above. The additional output aer_esba_k and hydro_esba_k contain the Jacobians of the corresponding input optical properties. The Legendre coefficients and phase functions are not implemented as "active" variables in the RTTOV K model so Jacobians are not calculated for them.

6.8. 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.

6.9. Additional wrapper routines

There are some additional wrapper routines providing access to additional RTTOV functionality.

Two subroutines are available which can be used to determine the RTTOV major and minor version numbers:

```
rttov_get_major_version(major_version)
rttov_get_minor_version(minor_version)
```

In both cases the arguments are INTENT(OUT) integers.

The following routine wraps the rttov_aer_clim_prof Fortran subroutine (see user guide Annex I):

```
rttov_get_aer_clim_prof(p_half, t, q, gas_units, mmr_aer, latitude, elevation, &
                        index, aer_prof, nlevels, naer_opac)
```

This is used to obtain climatological aerosol profiles based on the OPAC components. These can then be used in aerosol simulations with the OPAC aertable files by assigning the output profiles to the corresponding aerosol indices in the gas_id and gases arrays.

The arguments are as follows:

Argument	**Type**	**Intent**	**Description**
p_half(nlevels)	Real	in	Pressure half-levels (hPa).
t(nlayers)	Real	in	Temperatures on pressure full-levels (K).
q(nlayers)	Real	in	Water vapour on pressure full-levels (units according to gas_units).
gas_units	Integer	in	Units for input q: 2=>ppmv over moist air, 1=>kg/kg over moist air, 0=>ppmv over dry air.
mmr_aer	Integer 0 or 1	in	Units for output aerosols: 0=>cm^-3, 1=>kg/kg.
latitude	Real	in	Latitude of profile (degrees).
elevation	Real	in	Surface elevation (km).
index	Integer	in	Index of climatological scenario (1-10, see user guide).
aer_prof(nlayers,naer_opac)	Real	inout	Output climatological aerosol profiles.
nlevels	Integer	in	Number of pressure half-levels, nlayers=nlevels-1 (not required in Python).
naer_opac	Integer	in	Number of aerosol species in OPAC aertable files, this must always be 13, but only aerosol indices 1-10 are populated (not required in Python).

6.10. Specific information for Python

As noted above, it is strongly recommended to use *pyrttov* (section 3) to run RTTOV from Python rather than using these underlying function calls. The documentation for Python here is included only for completeness.

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 to the interface routines 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 in this section (6). 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.

6.11. Specific information for C/C++

As noted above, if calling RTTOV from C++ it is strongly recommended to use the class-based interface described in section 3. These underlying function calls are documented here specifically for calling RTTOV from other languages such as 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.

Appendix A: 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, hydrometeors, 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.

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

Options **options**
The __Options__ instance associated with this __Rttov/RttovSafe__ object. You should set the options associated with this instrument using the relevant members of this __Options__ instance.

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

int **getMajorVersion**() const
Return the RTTOV major version number.
int **getMinorVersion**() const
Return the RTTOV minor version number.
void **updateOptions** ()
Update RTTOV options for the currently loaded instrument.

void **printOptions** ()
Print RTTOV options for the currently loaded instrument.

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

void **setFileAertable** (const string &fileAertable)

Set the aerosol optical property filename.

void **setFileHydrotable** (const string &fileHydrotable)

Set the hydrometeor optical property filename.

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

Set the MFASIS-NN coefficient filename.

void **setFileMwPol** (const string &fileMwPol)

Set the ARO-scaled polarisation LUT 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.*

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.

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

*Set pointer to array containing input/output surface emissivity, BRDF, diffuse reflectance, specularity, and effective Tskin values; this must be previously allocated a double array of dimensions [5][nprofiles][nsurfaces][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 negative values indicating that RTTOV should supply emissivtiy/reflectance values.*

void **setAerEsba** (double *esba)

Set the aerosol extinction coefs, single-scattering albedos, bpr parameters, and asymmetry parameters. Dimensions of esba are [4][nprofiles][nchannels][nlayers].

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

Set the aerosol phase functions. Dimensions of phangle [nphangle] and dimensions of pha are [nprofiles][nchannels][nlayers][nphangle].

void **setAerLcoef** (int nmom, double *lcoef)

Set the aerosol phase function Legendre coefficients. Dimensions of lcoef are [nprofiles][nchannels][nlayers][nmom+1].

void **setHydroEsba** (double *esba)

Set the hydrometeor extinction coefs, single-scattering albedos, bpr parameters, and asymmetry parameters. Dimensions of esba are [4][nprofiles][nchannels][nlayers].

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

Set the hydrometeor phase functions. Dimensions of phangle [nphangle] and dimensions of pha are [nprofiles][nchannels][nlayers][nphangle].

void **setHydroLcoef** (int nmom, double *lcoef)

Set the hydrometeor phase function Legendre coefficients. Dimensions of lcoef are [nprofiles][nchannels][nlayers][nmom+1].

double **calcBpr** (int nphangle, double *phangle, double *pha)

Calculate bpr parameter for given phase function pha defined on angles phangle. The angles are in degrees running from 0° to 180°, and the angle grid does not need to be evenly spaced.

double **calcAsym** (int nphangle, double *phangle, double *pha)

Calculate asymmetry parameter for given phase function pha defined on angles phangle. The angles are in degrees running from 0° to 180°, and the angle grid does not need to be evenly spaced.

void **calcLcoef** (int nphangle, double *phangle, double *pha, int nmom, double *lcoef, int ngauss)

Calculate Legendre coefficients for given phase function pha defined on angles phangle. The angles are in degrees running from 0° to 180°, and the angle grid does not need to be evenly spaced. Populates lcoef, an array of size (nmom+1). If ngauss >= nmom, then ngauss will determine the size of the Gaussian quadrature used in the calculation. Set to zero to use the default (1000).

void **setZefK** (double *zef_k)

Set input reflectivity perturbations for radar Jacobian simulations. Dimensions of zef_k are [nprofiles][nchannels][nlayers].

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 * **getSurfEmisRefl** () const

Return a pointer to an array of dimensions [5][nprofiles][nsurfaces][nchannels] containing input values of surface emissivity, BRDF, diffuse 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.*

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.

std::vector< double > **getSurfEmis** (int profile, int surface)

Return surface emissivities used in the simulations for a given profile and surface, requires store_emis_refl true.

std::vector< double > **getSurfBrdf** (int profile, int surface)

Return surface BRDFs used in the simulations for a given profile and surface, requires store_emis_refl true.

std::vector< double > **getSurfDiffuseRefl** (int profile, int surface)

Return surface diffuse reflectances used in the simulations for a given profile and surface, requires store_emis_refl true.

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< double > **getBtOvercast** (int profile, int channel)

Return RTTOV radiance bt_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.

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.

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

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

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

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

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

Return RTTOV diagnostic output geometric_height output array of size [nlayers] for given profile, requires store_diag_output true.

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

Return RTTOV diagnostic output geometric_height_half output array of size [nlevels] for given profile, requires store_diag_output true.

double **getDiagOutputHydroFracEff** (int profile)

Return RTTOV diagnostic output hydro_frac_eff output for given profile, requires store_diag_output true.

double **getEmisTermsBsfc** (int profile, int channel)

Return emissivity retrieval bsfc output for given profile and channel, requires store_emis_terms true.

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

*Return emissivity retrieval column_weight output array of size [ncolumns] for given profile and channel, requires store_emis_terms true. The value of ncolumns is 1 for non-hydrometeor scattering simulations, 2 for hydrometeor scattering with two-column cloud overlap, and 2*nlayers+1 for hydrometeor scattering with max/random overlap.*

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

Return emissivity retrieval tau_sfc output array of size [ncolumns] for given profile and channel, requires store_emis_terms true.

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

Return emissivity retrieval rad_up output array of size [ncolumns] for given profile and channel, requires store_emis_terms true.

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

Return emissivity retrieval rad_down output array of size [ncolumns] for given profile and channel, requires store_emis_terms true.

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

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

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

Return the computed pressure full-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 > **getItemK** (rttov::itemIdType, int profile, int channel)

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

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

Return computed surface fraction Jacobians for a given profile and channel.

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

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

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

Return computed near-surface variable Jacobian for a given profile, surface 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 > **getHydroFracEffK** (int profile)

Return computed hydro_frac_eff variable Jacobians for a given profile.

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

Return computed surface emissivity Jacobians for a given profile and surface.

std::vector< double > **getSurfBrdfK** (int profile, int surface)

Return computed surface BRDF Jacobians for a given profile and surface.

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

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

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

std::vector< double > **getAerExtK** (int profile, int channel)
Return computed aerosol extinction coeffiicent Jacobians for a given profile and channel.

std::vector< double > **getAerSsaK** (int profile, int channel)
Return computed aerosol single-sca ttering albedo Jacobians for a given profile and channel.

std::vector< double > **getAerBprK** (int profile, int channel)
Return computed aerosol bpr parameter Jacobians for a given profile and channel.

std::vector< double > **getAerAsymK** (int profile, int channel)
Return computed aerosol asymmetry parameter Jacobians for a given profile and channel.

std::vector< double > **getHydroExtK** (int profile, int channel)
Return computed hydrometeor extinction coeffiicent Jacobians for a given profile and channel.

std::vector< double > **getHydroSsaK** (int profile, int channel)
Return computed hydrometeor single-scattering albedo Jacobians for a given profile and channel.

std::vector< double > **getHydroBprK** (int profile, int channel)
Return computed hydrometeor bpr parameter Jacobians for a given profile and channel.

std::vector< double > **getHydroAsymK** (int profile, int channel)
Return computed hydrometeor asymmetry parameter Jacobians for a given profile and channel.

Appendix B: Python *Rttov* class

Methods:

Rttov ()

Rttov class constructor method.

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.

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.

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, hydrometeor and aerosol Jacobian values. See Appendix J for the itemIdType class that can be used to obtain the IDs). If the requested Jacobian was not calculated this returns None, otherwise the result will be an array with dimensions [nprofiles][nchannels][nlayers]. It is possible (and recommended) to access each gas, hydrometeor or aerosol variable's Jacobians directly (see members below).

float array **getAerNK** (n)

Return computed Jacobian for 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][nlayers].

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][nlayers].

float array **getHydroFracNK** (n)

Return computed Jacobian for hydro 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][nlayers].

float array **getHydroDeffNK** (n)

Return computed Jacobian for hydro Deff 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][nlayers].

float **calcBpr** (phangle, pha)

Calculate bpr parameter for given phase function pha defined on angles phangle. The angles are in degrees running from 0° to 180°, and the angle grid does not need to be evenly spaced.

float **calcAsym** (phangle, pha)

Calculate asymmetry parameter for given phase function pha defined on angles phangle. The angles are in degrees running from 0° to 180°, and the angle grid does not need to be evenly spaced.

float array **calcLcoef** (phangle, pha, nmom, ngauss=0)

Calculate Legendre coefficients for given phase function pha defined on angles phangle. The angles are in degrees running from 0° to 180°, and the angle grid does not need to be evenly spaced. Returns an array of size (nmom+1). If ngauss >= nmom, then ngauss will determine the size of the Gaussian quadrature used in the calculation. Set to zero to use the default (1000).

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.*

int **MajorVersion**

The RTTOV major version number (read-only).

int **MinorVersion**

The RTTOV minor version number (read-only).

string **FileCoef**

The gas optical depth coefficient filename.

string **FileAertable**

The aerosol optical property filename.

string **FileHydrotable**

The hydrometeor optical property filename.

string **FileMfasisNN**

The MFASIS-NN coefficient filename.

string **FileMwPol**

The ARO-scaled polarisation LUT 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 **WaveNumbers**

Return the channel central wavenumbers of the coefficient file.

float array **SurfEmisRefl**

Array containing input surface emissivity and reflectance values of dimensions [5][nprofiles][nsurfaces][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 calc_emis, calc_brdf, and calc_diffuse_refl set to true).*

float array **AerEsba**

The aerosol extinction coefs, single-scattering albedos, bpr parameters, and asymmetry parameters. Dimensions are [4][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 **AerLcoef**

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

float array **HydroEsba**

The hydrometeor aextinction coefs, single-scattering albedos, bpr parameters, and asymmetry parameters . Dimensions are [4][nprofiles][nchannels][nlayers].

float array **HydroPhangle**

The hydrometeor phase function angles. Dimensions are [cld_nphangle].

float array **HydroPha**

The hydrometeor phase functions. Dimensions are [nprofiles][nchannels][nlayers][cld_nphangle].

float array **HydroLcoef**

The hydrometeor phase function Legendre coefficients. Dimensions are [nprofiles][nchannels][nlayers][cld_nmom+1].

float array **ZefK**

Input reflectivity perturbations for radar Jacobian simulations, dimensions [nprofiles][nchannels][nlayers].

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 **SurfEmis**

Computed surface emissivities used in the simulations, dimensions [nprofiles][nsurfaces][nchannels], requires store_emis_refl true.

float array **SurfBrdf**

Computed surface BRDFs used in the simulations, dimensions [nprofiles][nsurfaces][nchannels], requires store_emis_refl true..

float array **SurfDiffuseRefl**

Computed surface diffuse reflectances used in the simulations, dimensions [nprofiles][nsurfaces][nchannels], requires store_emis_refl true..

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.

float array **BtOvercast**
RTTOV radiance bt_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.

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.

float array **Zef**
RTTOV radar reflectivity zef output array, dimensions [nprofiles][nchannels][nlayers].

float array **AZef**
RTTOV radar reflectivity azef output array, dimensions [nprofiles][nchannels][nlayers].

float array **DiagOutputGeometricHeight**
RTTOV diagnostic_output geometric_height output array, dimensions [nprofiles][nlayers], requires store_diag_output true.

float array **DiagOutputGeometricHeightHalf**
RTTOV diagnostic_output geometric_height_half output array, dimensions [nprofiles][nlevels], requires store_diag_output true.

float array **DiagOutputHydroFracEff**
RTTOV diagnostic_output hydro_frac_eff output array, dimensions [nprofiles], requires store_diag_output true.

float array **EmisTermsBsfc**
Emissivity retrieval bsfc output array, dimensions [nprofiles][nchannels], requires store_emis_terms true.

float array **EmisTermsColumnWeight**
*Emissivity retrieval column_weight output array, dimensions [nprofiles][nchannels][ncolumns], requires store_emis_terms true. The value of ncolumns is 1 for non-hydrometeor scattering simulations, 2 for hydrometeor scattering with two-column cloud overlap, and 2*nlayers+1 for hydrometeor scattering with max/random overlap.*

float array **EmisTermsTauSfc**
Emissivity retrieval tau_sfc output array, dimensions [nprofiles][nchannels][ncolumns], requires store_emis_terms true.

float array **EmisTermsRadUp**

Emissivity retrieval rad_up output array, dimensions [nprofiles][nchannels][ncolumns], requires store_emis_terms true.

float array **EmisTermsRadDown**
Emissivity retrieval rad_down output array, dimensions [nprofiles][nchannels][ncolumns], requires store_emis_terms true.

float array **PHalfK**
Computed pressure half-level Jacobians, dimensions [nprofiles][nchannels][nlevels].

float array **PK**
Computed pressure full-level Jacobians, dimensions [nprofiles][nchannels][nlayers].

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

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

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

float array **CO2K**
Computed co2 Jacobians, dimensions [nprofiles][nchannels][nlayers].

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

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

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

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

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

float array **SurfaceFractionK**
Computed surface fraction Jacobians, dimensions [nprofiles][nsurfaces-1][nchannels].

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

float array **NearSurfaceK**
Computed near-surface variable Jacobian, dimensions [nprofiles][nsurfaces][nchannels][5].

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

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

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

float array **SurfDiffuseReflK**

Computed surface diffuse reflectance Jacobians, dimensions [nprofiles][nsurfaces][nchannels].

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

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

float array **AerEsbaK**
Computed Jacobians of aerosol extinction coefs, single-scattering albedos, bpr parameters, and asymmetry parameters, dimensions [4][nprofiles][nchannels][nlayers].

float array **HydroEsbaK**
Computed Jacobians of hydrometeor extinction coefs, single-scattering albedos, bpr parameters, and asymmetry parameters, dimensions [4][nprofiles][nchannels][nlayers].

float array **HydroFracEffK**
Computed effective hydro fraction Jacobians, dimensions [nprofiles][nchannels].

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

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

float array **HydroDeffNK** *where N=1, 2, …, 30*
Computed Jacobians for hydro Deff for hydrometeor type N, dimensions [nprofiles][nchannels][nlayers].

float array **AerNK** *where N=1, 2, …, 30*
Computed Jacobians for aerosol type N, dimensions [nprofiles][nchannels][nlayers].

float array **HydroFracK**
Computed Jacobians for hydro fraction, dimensions [nprofiles][nchannels][nlayers].

float array **StcoK**
Computed OPAC stco (UV/VIS/IR hydrometeor type 1) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **StmaK**
Computed OPAC stma (UV/VIS/IR hydrometeor type 2) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **CuccK**
Computed OPAC cucc (UV/VIS/IR hydrometeor type 3) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **CucpK**
Computed OPAC cucp (UV/VIS/IR hydrometeor type 4) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **CumaK**
Computed OPAC cuma (UV/VIS/IR hydrometeor type 5) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **ClwdK**

Computed "cloud liquid Deff" (UV/VIS/IR hydrometeor type 6) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **BaumK**
Computed Baum cloud ice (UV/VIS/IR hydrometeor type 7) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **BaranK**
Computed Baran cloud ice (UV/VIS/IR hydrometeor type 8) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **ClwDeffK**
Computed Clwd type Deff Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **BaumIceDeffK**
Computed Baum type Deff Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **MWRainK**
Computed rain (MW hydrometeor type 1) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **MWSnowK**
Computed snow (MW hydrometeor type 2) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **MWGraupelK**
Computed graupel (MW hydrometeor type 3) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **MWClwK**
Computed cloud liquid water (MW hydrometeor type 4) Jacobians, dimensions [nprofiles][nchannels][nlayers].

float array **MWCiwK**
Computed cloud ice water (MW hydrometeor type 5) Jacobians, dimensions [nprofiles][nchannels][nlayers].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 (const int nlevels, const int nsurfaces)
Constructor method.

void **setGasUnits** (const **rttov::gasUnitType** gasUnits)
Set the gas_units. See Appendix J for the gasUnitType enumeration constants.

void **setMmrHydro** (const bool mmrHydro)
Set the mmr_hydro flag.

void **setMmrAer** (const bool mmrAer)
Set the mmr_aer flag.

void **setPHalf** (const std::vector< double > &pHalf)
Set the pressure half-levels vector (size nlevels).

void **setP** (const std::vector< double > &p)
Set the pressure full-levels vector (size nlayers).

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

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

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

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

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

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

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

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

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

void **setAngles** (const double satzen, const double satazi, const double sunzen, const double sunazi)
Set satellite an solar angles.

void **setSurfaceFraction** (const std::vector<double>& surface_fraction)
Set surface coverage fraction for each surface except the last one. Vector size is nsurfaces-1, do not specify if nsurfaces=1.

void **setNearSurface** (const isurf, const double t_2m, const double q_2m, const double u_10m, const double v_10m, const double wind_fetch)

Set near-surface parameters for surface index isurf (0<=isurf<=nsurfaces-1).

void **setSkin** (const isurf, 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 for surface index isurf (0<=isurf<=nsurfaces-1).

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

Set surface type parameters for surface index isurf (0<=isurf<=nsurfaces-1).

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

Set zeeman parameters.

void **setHydroFracEff** (const double hydro_frac_eff)

Set item hydro_frac_eff for the profile.

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 (size nlayers).

void **setHydroN** (const std::vector< double > &hydro, const int n)

Set profile hydro for hydrometeor type n (1<=n<=30) for the profile (size nlayers).

void **setHydroDeffN** (const std::vector< double > &hydro_deff, const int n)

Set profile hydro_deff for hydrometeor type n (1<=n<=30) for the profile (size nlayers).

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

Set profile aer of aerosol type n (1<=n<=30) for the profile (size nlayers).

void **setAerClimProf**(int index)

Generate climatological aerosol profiles based on OPAC types. The index argument is an integer (1-10) indicating which of the 10 climatological types to generate. See the rttov_aer_clim_prof subroutine in Annex I of the RTTOV user guide.

void **setHydroFrac** (const std::vector< double > &hydro_frac)

Set item hydro_frac for the profile (size nlayers).

void **setStco** (const std::vector< double > &stco)

Set item stco (UV/VIS/IR hydrometeor type 1) for the profile (size nlayers).

void **setStma** (const std::vector< double > &stma)

Set item stma (UV/VIS/IR hydrometeor type 2) for the profile (size nlayers).

void **setCucc** (const std::vector< double > &cucc)

Set item cucc (UV/VIS/IR hydrometeor type 3) for the profile (size nlayers).

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

Set item cucp (UV/VIS/IR hydrometeor type 4) for the profile (size nlayers).

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

Set item cuma (UV/VIS/IR hydrometeor type 5) for the profile (size nlayers).

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

Set item clwd (cloud liquid water "Deff" type, UV/VIS/IR hydrometeor type 6) for the profile (size nlayers).

void **setBaum** (const std::vector< double > &baum)

Set item Baum ice cloud (UV/VIS/IR hydrometeor type 7) for the profile (size nlayers).

void **setBaran** (const std::vector< double > &baran)

Set item Baran ice cloud (UV/VIS/IR hydrometeor type 8) for the profile (size nlayers).

void **setClwDeff** (const std::vector< double > &clwDeff)

Set item Deff for Clwd type (UV/VIS/IR hydrometeor type 6) for the profile (size nlayers).

void **setBaumIceDeff** (const std::vector< double > &baumIceDeff)

Set item Deff for Baum type (UV/VIS/IR hydrometeor type 7) for the profile (size nlayers).

void **setClwdeParam** (const int clwde_param)

Set item clwde_param for the profile.

void **setIcedeParam** (const int icede_param)

Set item icede_param for the profile.

void **setMwRain** (const std::vector< double > &rain)

Set item rain (MW hydrometeor type 1) for the profile (size nlayers).

void **setMwSnow** (const std::vector< double > &snow)

Set item snow (MW hydrometeor type 2) for the profile (size nlayers).

void **setMwGraupel** (const std::vector< double > &graupel)

Set item graupel (MW hydrometeor type 3) for the profile (size nlayers).

void **setMwClw** (const std::vector< double > &clw)

Set item cloud liquid water (MW hydrometeor type 4) for the profile (size nlayers).

void **setMwCiw** (const std::vector< double > &ciw)

Set item cloud ice water (MW hydrometeor type 5) for the profile (size nlayers).

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

Set item inso (OPAC aerosol type 1) for the profile (size nlayers).

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

Set item waso (OPAC aerosol type 2) for the profile (size nlayers).

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

Set item soot (OPAC aerosol type 3) for the profile (size nlayers).

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

Set item ssam (OPAC aerosol type 4) for the profile (size nlayers).

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

Set item sscm (OPAC aerosol type 5) for the profile (size nlayers).

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

Set item minm (OPAC aerosol type 6) for the profile (size nlayers).

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

Set item miam (OPAC aerosol type 7) for the profile (size nlayers).

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

Set item micm (OPAC aerosol type 8) for the profile (size nlayers).

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

Set item mitr (OPAC aerosol type 9) for the profile (size nlayers).

void **setSuso** (const std::vector< double > &suso)
Set item suso (OPAC aerosol type 10) for the profile (size nlayers).

void **setVola** (const std::vector< double > &vola)
Set item vola (OPAC aerosol type 11) for the profile (size nlayers).

void **setVapo** (const std::vector< double > &vapo)
Set item vapo (OPAC aerosol type 12) for the profile (size nlayers).

void **setAsdu** (const std::vector< double > &asdu)
Set item asdu (OPAC aerosol type 13) for the profile (size nlayers).

void **setBcar** (const std::vector< double > &bcar)
Set item bcar (CAMS aerosol type 1) for the profile (size nlayers).

void **setDus1** (const std::vector< double > &dus1)
Set item dus1 (CAMS aerosol type 2) for the profile (size nlayers).

void **setDus2** (const std::vector< double > &dus2)
Set item dus2 (CAMS aerosol type 3) for the profile (size nlayers).

void **setDus3** (const std::vector< double > &dus3)
Set item dus3 (CAMS aerosol type 4) for the profile (size nlayers).

void **setSulp** (const std::vector< double > &sulp)
Set item sulp (CAMS aerosol type 5) for the profile (size nlayers).

void **setSsa1** (const std::vector< double > &ssa1)
Set item ssa1 (CAMS aerosol type 6) for the profile (size nlayers).

void **setSsa2** (const std::vector< double > &ssa2)
Set item ssa2 (CAMS aerosol type 7) for the profile (size nlayers).

void **setSsa3** (const std::vector< double > &ssa3)
Set item ssa3 (CAMS aerosol type 8) for the profile (size nlayers).

void **setOmat** (const std::vector< double > &omat)
Set item omat (CAMS aerosol type 9) for the profile (size nlayers).

Appendix D: C++ *Profiles* class (used with *Rttov* objects)

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

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

void **setMmrHydro**(const bool mmrhydro)
Set the mmr_hydro flag.

void **setMmrAer** (const bool mmraer)
Set the mmr_aer flag.

void **setPHalf** (double *pHalf)
Set the pointer to the p half-levels array of size [nprofiles][nlevels].

void **setP** (double *p)
Set the pointer to the p full-levels array of size [nprofiles][nlayers].

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

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

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

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

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

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

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

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

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

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

void **setSurfaceFraction** (double *surfaceFraction)
Set the pointer to the surfaceFraction array of size [nprofiles][nsurfaces-1] containing surface coverage fraction for each surface except the last one for each profile. Do not specify if nsurfaces=1.

void **setNearSurface** (double *nearSurface)

Set the pointer to the nearSurface array of size [nprofiles][nsurfaces][5] containing 2m t, 2m q, 10m wind u, v, wind fetch for each surface for each profile.

void **setSkin** (double *skin)

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

void **setSurfType** (int *surftype)

Set the pointer to the surftype array of size [nprofiles][nsurfaces][2] containing surftype, watertype for each surface 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 year, month, day, hour, minute, second 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 **setZeeman** (double *zeeman)

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

void **setClwdeParam** (int *clwdeParam)

Set the pointer to the clwdeParam array of size [nprofiles] containing clwde_param for each profile.

void **setIcedeParm** (int *icedeParam)

Set the pointer to the icedeParam array of size [nprofiles] containing icede_param for each profile.

void **setHydroFracEff** (double *hydroFracEff)

Set the pointer to the hydroFracEff array of size [nprofiles] containing hydro_frac_eff for each profile.

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

Set a gas, hydrometeor, or aerosol profile variable, gasItem size [nprofiles][nlayers]. See Appendix J for the itemIdType enumeration constants.

setAerClimProf (int* indices)

Generate climatological aerosol profiles based on OPAC types. The indices argument is an array of size [nprofiles] containing integers (1-10) indicating which of the 10 climatological types to generate for each profile. See the rttov_aer_clim_prof subroutine in Annex I of the RTTOV user guide.

Appendix E: 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, nsurfaces)
Constructor method.

setAerClimProf (indices)
Generate climatological aerosol profiles based on OPAC types. The indices argument is an array of size [nprofiles] containing integers (1-10) indicating which of the 10 climatological types to generate for each profile. See the rttov_aer_clim_prof subroutine in Annex I of the RTTOV user guide.

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

delAerN (n)
Delete profile data for aerosol type n (1<=n<=30).

setHydroN (hydro, n)
Set profile hydro of size [nprofiles][nlayers] 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][nlayers] of hydro 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 hydro fraction for hydrometeor type n (1<=n<=30).

setHydroDeffN (hydro_deff, n)
Set profile hydro_deff of size [nprofiles][nlayers] of hydro Deff for hydrometeor type n (1<=n<=30). You can also access these individually via the HydroDeffN (N=1,2,...,30) members described below.

delHydroDeffN (n)
Delete profile data for hydro Deff for hydrometeor type n (1<=n<=30).

Members:

int **GasUnits**
The gas_units (see Appendix J for the gasUnitType class).

int **MmrHydro**
The mmr_hydro flag.

int **MmrAer**
The mmr_aer flag.

float array **PHalf**

The pressure half-levels array of size [nprofiles][nlevels].

float array **P**
: *The pressure full-levels array of size [nprofiles][nlayers].*

float array **T**
: *The temperature array of size [nprofiles][nlayers].*

float array **Q**
: *The water vapour array of size [nprofiles][nlayers].*

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

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

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

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

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

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

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

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

float array **SurfaceFraction**
: *The surface_fraction array of size [nprofiles][nsurfaces-1] containing the surface coverage fraction for each surface except the last one for each profile. Do not specify if nsurfaces=1.*

float array **NearSurface**
: *The near_surface array of size [nprofiles][nsurfaces][5] containing 2m t, 2m q, 10m wind u, v, wind fetch for each surface for each profile.*

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

int array **SurfType**
: *The surftype array of size [nprofiles][nsurfaces][2] containing surftype, watertype for each surface 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 year, month, day, hour, minute, second for each profile.*

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

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

float array **HydroFracEff**
The hydro_frac_eff for each profile, of size [nprofiles].

float array **HydroFracN** *where N=1, 2, …, 30*
The hydromteor type N hydro fraction array of size [nprofiles][nlayers].

float array **HydroN** *where N=1, 2, …, 30*
The hydrometeor type N array of size [nprofiles][nlayers].

float array **HydroDeffN** *where N=1, 2, …, 30*
The hydromteor type N effective diameter array of size [nprofiles][nlayers].

float array **AerN** *where N=1, 2, …, 30*
The aerosol type N array of size [nprofiles][nlayers].

float array **HydroFrac**
The hydro_frac array of size [nprofiles][nlayers].

float array **Stco**
The OPAC stco cloud liquid type (UV/VIS/IR hydrometeor type 1) array of size [nprofiles][nlayers].

float array **Stma**
The OPAC stma cloud liquid type (UV/VIS/IR hydrometeor type 2) array of size [nprofiles][nlayers].

float array **Cucc**
The OPAC cucc cloud liquid type (UV/VIS/IR hydrometeor type 3) array of size [nprofiles][nlayers].

float array **Cucp**
The OPAC cucp cloud liquid type (UV/VIS/IR hydrometeor type 4) array of size [nprofiles][nlayers].

float array **Cuma**
The OPAC cuma cloud liquid type (UV/VIS/IR hydrometeor type 5) array of size [nprofiles][nlayers].

float array **Clwd**
The "cloud liquid Deff" type (UV/VIS/IR hydrometeor type 6) array of size [nprofiles][nlayers].

float array **Baum**
The Baum cloud ice type (UV/VIS/IR hydrometeor type 7) array of size [nprofiles][nlayers].

float array **Baran**
The Baran cloud ice type (UV/VIS/IR hydrometeor type 8) array of size [nprofiles][nlayers].

float array **ClwDeff**
The Deff for Clwd type (Deff for UV/VIS/IR hydrometeor type 6) array of size [nprofiles][nlayers].

float array **BaumIceDeff**
The Deff for Baum type (Deff UV/VIS/IR hydrometeor type 7) array of size [nprofiles][nlayers].

int array **ClwdeParam**
The clwde_param for each profile, of size [nprofiles].

int array **IcedeParam**
The icede_param for each profile, of size [nprofiles]

float array **MWRain**
The rain (MW hydrometeor type 1) array of size [nprofiles][nlayers].

float array **MWSnow**
The snow (MW hydrometeor type 2) array of size [nprofiles][nlayers].

float array **MWGraupel**
The graupel (MW hydrometeor type 3) array of size [nprofiles][nlayers].

float array **MWClw**
The cloud liquid water (MW hydrometeor type 4) array of size [nprofiles][nlayers].

float array **MWCiw**
The cloud ice water (MW hydrometeor type 5) array of size [nprofiles][nlayers].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

float array **Sulp**
The sulp (CAMS aerosol type 5) array of size [nprofiles][nlayers].

float array **Ssa1**
The ssa1 (CAMS aerosol type 6) array of size [nprofiles][nlayers].

float array **Ssa2**
The ssa2 (CAMS aerosol type 7) array of size [nprofiles][nlayers].

float array **Ssa3**
The ssa3 (CAMS aerosol type 8) array of size [nprofiles][nlayers].

float array **Omat**
The omat (CAMS aerosol type 9) array of size [nprofiles][nlayers].

Appendix F: 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/RttovOptions.h. The **Rttov** and **RttovSafe** objects have options members so there is usually no need to create instances of this class manually.

Options ()

> *Constructor method.*

void **setNthreads** (int nthreads)

> *Set the number of threads RTTOV will use (compile RTTOV with OpenMP to make use of this)*

int **getNthreads** () const

> *Return 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 the RTTOV direct or K models per call.*

int **getNprofsPerCall** () const

> *Return the number of profiles passed into the RTTOV direct or K models per call.*

void **setVerboseWrapper** (bool verboseWrapper)

> *Set the verbose_wrapper option (verbose output from wrapper).*

bool **isVerboseWrapper** () const

> *Return set the verbose_wrapper option.*

void **setCheckOpts** (bool checkOpts)

> *Set the check_opts option (call* rttov_user_check_opts *after loading an instrument or when updating options).*

bool **isCheckOpts** () const

> *Return set the check_opts option.*

void **setStoreEmisRefl** (bool storeEmisRefl)

> *Set the store_emis_refl wrapper option (output of surface emissivity/reflectance data).*

bool **isStoreEmisRefl** () const

> *Return the store_emis_refl wrapper option.*

void **setStoreTrans** (bool storeTrans)

> *Set the store_trans wrapper option (output of transmittance data).*

bool **isStoreTrans** () const

> *Return the store_trans wrapper option.*

void **setStoreRad** (bool storeRad)

> *Set the store_rad wrapper option (output of primary radiance data).*

bool **isStoreRad** () const

> *Return the store_rad wrapper option.*

void **setStoreRad2** (bool storeRad2)

> *Set the store_rad2 wrapper option (output of secondary radiance data).*

bool **isStoreRad2** () const

> *Return the store_rad2 wrapper option.*

void **setStoreDiagOutput** (bool diagOutput)

Set the store_diag_output wrapper option (output of diagnostic output data).

bool **isStoreDiagOutput** () const

Return the store_diag_output wrapper option.

void **setStoreEmisTerms** (bool storeEmisTerms)

Set the store_emis_terms wrapper option (output of emissivity retrieval terms data).

bool **isStoreEmisTerms** () const

Return the store_emis_terms wrapper option.

void **setRadarKAzef** (bool radarKAzef)

Set the radar_k_azef wrapper option (radar K inputs in Azef or Zef).

bool **isRadarKAzef** () const

Return the radar_k_azef wrapper option.

void **setVerbose** (bool verbose)

Set the opts%config%verbose option.

bool **isVerbose** ()

Return the opts%config%verbose option.

void **setApplyRegLimits** (bool applyRegLimts)

Set the opts%config%apply_reg_limits option.

bool **isApplyRegLimits** ()

Return the opts%config%apply_reg_limits option.

void **setCheckProfiles** (bool checkProfiles)

Set the opts%config%check_profiles option.

bool **isCheckProfiles** ()

Return the opts%config%check_profiles option.

void **setTransmittancesOnly** (bool transmittancesOnly)

Set the opts%config%transmittances_only option.

bool **isTransmittancesOnly** ()

Return the opts%config%transmittances_only option.

void **setBtOvercastCalc** (bool btOvercastCalc)

Set the opts%config%bt_overcast_calc option.

bool **isBtOvercastCalc** ()

Return the opts%config%bt_overcast_calc option.

void **setGasOpdepCalc** (bool gasOpdepCalc)

Set the opts%config%gas_opdep_calc option.

bool **isGasOpdepCalc** ()

Return the opts%config%gas_opdep_calc option.

void **setADKBT** (bool adkBt)

Set the opts%config%adk_bt option.

bool **isADKBT** ()

Return the opts%config%adk_bt option.

void **setADKRefl** (bool adkRefl)
Set the opts%config%adk_refl option.

bool **isADKRefl** ()
Return the opts%config%adk_refl option.

void **setOpdep13GasClip** (bool opdep13GasClip)
Set the opts%config%opdep13_gas_clip option.

bool **isOpdep13GasClip** ()
Return the opts%config%opdep13_gas_clip option.

void **setEnableInterp** (bool enableInterp)
Set the opts%interpolation%enable_interp option.

bool **isEnableInterp** ()
Return the opts%interpolation%enable_interp option.

void **setInterpMode** (int interpMode)
Set the opts%interpolation%interp_mode option.

int **getInterpMode** () const
Return the opts%interpolation%interp_mode option.

void **setPressureGradients** (bool pressureGradients)
Set the opts%interpolation%pressure_gradients option.

bool **isPressureGradients** ()
Return the opts%interpolation%pressure_gradients option.

void **setO3Data** (bool o3Data)
Set the opts%rt_all%o3_data option.

bool **isO3Data** ()
Return the opts%rt_all%o3_data option.

void **setCO2Data** (bool co2Data)
Set the opts%rt_all%co2_data option.

bool **isCO2Data** ()
Return the opts%rt_all%co2_data option.

void **setN2OData** (bool n2oData)
Set the opts%rt_all%n2o_data option.

bool **isN2OData** ()
Return the opts%rt_all%n2o_data option.

void **setCOData** (bool coData)
Set the opts%rt_all%co_data option.

bool **isCOData** ()
Return the opts%rt_all%co_data option.

void **setCH4Data** (bool ch4Data)
Set the opts%rt_all%ch4_data option.

bool **isCH4Data** ()
Return the opts%rt_all%ch4_data option.

void **setSO2Data** (bool so2Data)

Set the opts%rt_all%so2_data option.

bool **isSO2Data** ()

Return the opts%rt_all%so2_data option.

void **setSolar** (bool solar)

Set the opts%rt_all%solar option.

bool **isSolar** ()

Return the opts%rt_all%solar option.

void **setRayleighMaxWavelength** (double rayleighMaxWavelength)

Set the opts%rt_all%rayleigh_max_wavelength option.

double **getRayleighMaxWavelength** () const

Return the opts%rt_all%rayleigh_max_wavelength option.

void **setRayleighMinPressure** (double rayleighMinPressuer)

Set the opts%rt_all%rayleigh_min_pressure option.

double **getRayleighMinPressure** () const

Return the opts%rt_all%rayleigh_min_pressure option.

void **setRayleighSingleScatt** (bool rayleighSingleScatt)

Set the opts%rt_all%rayleigh_single_scatt option.

bool **isRayleighSingleScatt** ()

Return the opts%rt_all%rayleigh_single_scatt option.

void **setNlteCorrection** (bool nlteCorrection)

Set the opts%rt_all%nlte_correction option.

bool **isNlteCorrection** ()

Return the opts%rt_all%nlte_correction option.

void **setRefraction** (bool refraction)

Set the opts%rt_all%refraction option.

bool **isRefraction** ()

Return the opts%rt_all%refraction option.

void **setPlaneParallel** (bool planeParallel)

Set the opts%rt_all%plane_parallel option.

bool **isPlaneParallel** ()

Return the opts%rt_all%plane_parallel option.

void **setRadDownLinTau** (bool radDownLinTau)

Set the opts%rt_all%rad_down_lin_tau option.

bool **isRadDownLinTau** ()

Return the opts%rt_all%rad_down_lin_tau option.

void **setUseT2m** (bool useT2m)

Set the opts%rt_all%use_t2m option.

bool **isUseT2m** ()

Return the opts%rt_all%use_t2m option.

void **setUseQ2m** (bool useQ2m)

Set the opts%rt_all%use_q2m option.

bool **isUseQ2m** ()
Return the opts%rt_all%use_q2m option.

void **setSolarSeaReflModel** (int solarSeaReflModel)
Set the opts%surface%solar_sea_refl_model option.

int **getSolarSeaReflModel** () const
Return the opts%surface%solar_sea_refl_model option.

void **setIrSeaEmisModel** (int irSeaEmisModel)
Set the opts%surface%ir_sea_emis_model option.

int **getIrSeaEmisModel** () const
Return the opts%surface%ir_sea_emis_model option.

void **setMwSeaEmisModel** (int mwSeaEmisModel)
Set the opts%surface%mw_sea_emis_model option.

int **getMwSeaEmisModel** () const
Return the opts%surface%mw_sea_emis_model option.

void **setUseFoamFraction** (bool useFoamFraction)
Set the opts%surface%use_foam_fraction option.

bool **isUseFoamFraction** ()
Return the opts%surface%use_foam_fraction option.

void **setLambertian** (bool lambertian)
Set the opts%surface%lambertian option.

bool **isLambertian** ()
Return the opts%surface%lambertian option.

void **setLambertianFixedAngle** (bool lambertianFixedAngle)
Set the opts%surface%lambertian_fixed_angle option.

bool is**LambertianFixedAngle** ()
Return the opts%surface%lambertian_fixed_angle option.

void **setUseTskinEff** (bool useTskinEff)
Set the opts%surface%use_tskin_eff option.

bool **isUseTskinEff** ()
Return the opts%surface%use_tskin_eff option.

void **setCLWData** (bool clwData)
Set the opts%clw_absorption%clw_data option.

bool **isCLWData** ()
Return the opts%clw_absorption%clw_data option.

void **setPermittivityParam** (int *permittivityParam*)
Set the opts%clw_absorption%permittivity_param option.

int **getPermittivityParam** () const
Return the opts%clw_absorption%permittivity_param option.

void **setCLWCloudTop** (double clwCloudTop)
Set the opts%clw_absorption%clw_cloud_top option.

double **getCLWCloudTop** () const

Return the opts%clw_absorption%clw_cloud_top option.

void **setHydrometeors** (bool hydrometeors)

Set the opts%scatt%hydrometeors option.

bool **isHydrometeors** ()

Return the opts%scatt%hydrometeors option.

void **setAerosols** (bool aerosols)

Set the opts%scatt%aerosols option.

bool **isAerosols** ()

Return the opts%scatt%aerosols option.

void **setThermalSolver** (int thermalSolver)

Set the opts%scatt%thermal_solver option.

int **getThermalSolver** () const

Return the opts%scatt% thermal_solver option.

void **setSolarSolver** (int solarSolver)

Set the opts%scatt%solar_solver option.

int **getSolarSolver** () const

Return the opts%scatt%solar_solver option.

void **setRadar** (bool radar)

Set the opts%scatt%radar option.

bool **isRadar** ()

Return the opts%scatt%radar option.

void **setUserHydroOptParam** (bool userHydroOptParam)

Set the opts%scatt%user_hydro_opt_param option.

bool **isUserHydroOptParam** ()

Return the opts%scatt%user_hydro_opt_param option.

void **setUserAerOptParam** (bool userAerOptParam)

Set the opts%scatt%user_aer_opt_param option.

bool **isUserAerOptParam** ()

Return the opts%scatt%user_aer_opt_param option.

void **setBaranIceVersion**(int baranIceVersion)

Set the opts%scatt%baran_ice_version option.

int **getBaranIceVersion** () const

Return the opts%scatt%baran_ice_version option.

void **setRayleighMultiScatt** (bool rayleighMultiScatt)

Set the opts%scatt%rayleigh_multi_scatt option.

bool **isRayleighMultiScatt** ()

Return the opts%scatt%rayleigh_multi_scatt option.

void **setDomNstreams** (int domNstreams)

Set the opts%scatt%dom_nstreams option.

int **getDomNstreams** () const

Return the opts%scatt%dom_nstreams option.

void **setDomAccuracy** (double domAccuracy)

Set the opts%scatt%dom_accuracy option.

double **getDomAccuracy** () const

Return the opts%scatt%dom_accuracy option.

void **setDomOpdepThreshold** (double domOpdepThreshold)

Set the opts%scatt%dom_opdep_threshold option.

double **getDomOpdepThreshold** () const

Return the opts%scatt%dom_opdep_threshold option.

void **setChouTangMod** (bool chouTangMod)

Set the opts%scatt%chou_tang_mod option.

bool **isChouTangMod** ()

Return the opts%scatt%chou_tang_mod option.

void **setChouTangFactor** (double chouTangFactor)

Set the opts%scatt%chou_tang_factor option.

double **getChouTangFactor** () const

Return the opts%scatt%chou_tang_factor option.

void **setMwPolMode** (bool mwPolMode)

Set the opts%scatt%Mw_pol_mode option.

int **isMwPolMode** () const

Return the opts%scatt%mw_pol_mode option.

void **setIcePolarisation** (double icePolarisation)

Set the opts%scatt%ice_polarisation option.

double **getIcePolarisation** () const

Return the opts%scatt%ice_polarisation option.

void **setZeroHydroTLAD** (bool zeroHydroTLAD)

Set the opts%scatt%zero_hydro_tlad option.

bool **isZeroHydroTLAD** ()

Return the opts%scatt%zero_hydro_tlad option.

void **setOverlapParam** (int overlapParam)

Set the opts%cloud_overlap%overlap_param option.

bool **getOverlapParam** ()

Return the opts%cloud_overlap%overlap_param option.

void **setPerHydroFrac** (bool perHydroFrac)

Set the opts%cloud_overlap%per_hydro_frac option.

bool **isPerHydroFrac** ()

Return the opts%cloud_overlap%per_hydro_frac option.

void **setColThreshold** (double colThreshold)

Set the opts%cloud_overlap%col_threshold option.

double **getColThreshold** () const

Return the opts%cloud_overlap%col_threshold option.

void **setTwoColMaxFracMaxP** (double twoColMaxFracMaxP)

Set the opts%cloud_overlap%two_col_max_frac_max_p option.

double **getTwoColMaxFracMaxP** () const

Return the opts%cloud_overlap%two_col_max_frac_max_p option.

void **setHydroFracTLAD** (bool hydroFracTLAD)

Set the opts%cloud_overlap%hydro_frac_tlad option.

bool **isHydroFracTLAD** ()

Return the opts%cloud_overlap%hydro_frac_tlad option.

Appendix G: Python *Options* class

The members below correspond directly to the RTTOV and wrapper options and are referenced directly. The **Rttov** class has an **Options** member so there is usually no need to create instances of this class manually.

Methods:

Options ()

Constructor method.

Members:

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 the RTTOV direct or K models per call.

bool **VerboseWrapper**

The verbose_wrapper option (verbose output from wrapper).

bool **CheckOpts**

The check_opts option (call rttov_user_check_opts *after loading an instrument or when updating options).*

bool **StoreEmisRefl**

The store_emis_refl wrapper option (output of surface emissivity/reflectance data).

bool **StoreTrans**

The store_trans wrapper option (output of transmittance data).

bool **StoreRad**

The store_rad wrapper option (output of primary radiance data)..

bool **StoreRad2**

The store_rad2 wrapper option (output of secondary radiance data).

bool **StoreDiagOutput**

The store_diag_output wrapper option (output of diagnostic output data).

bool **StoreEmisTerms**

The store_emis_terms wrapper option (output of emissivity retrieval terms data).

bool **RadarKAzef**

The radar_k_azef wrapper option (radar K inputs in Azef or Zef).

bool **Verbose**

The opts%config%verbose option.

bool **ApplyRegLimits**

The opts%config%apply_reg_limits option.

bool **CheckProfiles**

The opts%config%check_profiles option.

bool **TransmittancesOnly**

The opts%config%transmittances_only option.

bool **BtOvercastCalc** ()

The opts%config%bt_overcast_calc option.

bool **GasOpdepCalc** ()

The opts%config%gas_opdep_calc option.

bool **ADKBT**

The opts%config%adk_bt option.

bool **ADKRefl**

The opts%config%adk_refl option.

bool **Opdep13GasClip**

The opts%config%opdep13_gas_clip option.

bool **EnableInterp**

The opts%interpolation%enable_interp option.

int **InterpMode**

The opts%interpolation%interp_mode option.

bool **PressureGradients**

The opts%interpolation%pressure_gradients option.

bool **O3Data**

The opts%rt_all%o3_data option.

bool **CO2Data**

The opts%rt_all%co2_data option.

bool **N2OData**

The opts%rt_all%n2o_data option.

bool **COData**

The opts%rt_all%co_data option.

bool **CH4Data**

The opts%rt_all%ch4_data option.

bool **SO2Data**

The opts%rt_all%so2_data option.

bool **Solar**

The opts%rt_all%solar option.

float **RayleighMaxWavelength**

The opts%rt_all%rayleigh_max_wavelength option.

float **RayleighMinPressure**

The opts%rt_all%rayleigh_min_pressure option.

bool **RayleighSingleScatt**

The opts%rt_all%rayleigh_single_scatt option.

bool **NlteCorrection**

The opts%rt_all%nlte_correction option.

bool **Refraction**

The opts%rt_all%refraction option.

bool **PlaneParallel**

The opts%rt_all%plane_parallel option.

bool **RadDownLinTau**

The opts%rt_all%rad_down_lin_tau option.

bool **UseT2m**

The opts%rt_all%use_t2m option.

bool **UseQ2m**

The opts%rt_all%use_q2m option.

int **SolarSeaReflModel**

The opts%surface%solar_sea_refl_model option.

int **IrSeaEmisModel**

The opts%surface%ir_sea_emis_model option.

int **MwSeaEmisModel**

The opts%surface%mw_sea_emis_model option.

bool **UseFoamFraction**

The opts%surface%use_foam_fraction option.

bool **Lambertian**

The opts%surface%lambertian option.

bool **LambertianFixedAngle**

The opts%surface%lambertian_fixed_angle option.

bool **UseTskinEff**

The opts%surface%use_tskin_eff option.

bool **CLWData**

The opts%clw_absorption%clw_data option.

int **PermittivityParam**

The opts%clw_absorption%permittivity_param option.

float **CLWCloudTop**

The opts%clw_absorption%clw_cloud_top option.

bool **Hydrometeors**

The opts%scatt%hydrometeors option.

bool **Aerosols**

The opts%scatt%aerosols option.

int **ThermalSolver**

The opts%scatt%thermal_solver option.

int **SolarSolver**

The opts%scatt%solar_solver option.

bool **Radar**

The opts%scatt%radar option.

bool **UserHydroOptParam**

The opts%scatt%user_hydro_opt_param option.

bool **UserAerOptParam**

The opts%scatt%user_aer_opt_param option.

int **BaranIceVersion**

The opts%scatt%baran_ice_version option.

bool **RayleighMultiScatt**

The opts%scatt%rayleigh_multi_scatt option.

int **DomNstreams**

The opts%scatt%dom_nstreams option.

float **DomAccuracy**

The opts%scatt%dom_accuracy option.

float **DomOpdepThreshold**

The opts%scatt%dom_opdep_threshold option.

bool **ChouTangMod**

The opts%scatt%chou_tang_mod option.

float **ChouTangFactor**

The opts%scatt%chou_tang_factor option.

int **MwPolMode** ()

The opts%scatt%mw_pol_mode option.

float **IcePolarisation**

The opts%scatt%ice_polarisation option.

bool **ZeroHydroTLAD**

The opts%scatt%zero_hydro_tlad option.

bool **OverlapParam**

The opts%cloud_overlap%overlap_param option.

bool **PerHydroFrac**

The opts%cloud_overlap%per_hydro_frac option.

float **ColThreshold**

The opts%cloud_overlap%col_threshold option.

float **TwoColMaxFracMaxP**

The opts%cloud_overlap%two_col_max_frac_max_p option.

bool **HydroFracTLAD**

The opts%cloud_overlap%hydro_frac_tlad option.

Appendix H: C++ *Atlas* class

Atlas ()

Atlas class constructor method.

Atlas (const 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** (const bool **verbose**)

Set the verbose boolean.

void **setIncLand** (const bool incLand)

Set the inc_land boolean.

bool **getIncLand** () const

Return the inc_land boolean.

void **setIncSeaIce** (const bool incSeaIce)

Set the inc_seaice boolean.

bool **getIncSeaIce** () const

Return the inc_seaice boolean.

void **setIncSea** (const bool incSea)

Set the inc_sea boolean.

bool **getIncSea** () const

Return the inc_sea boolean.

void **setMaxDistance** (const double maxDistance)

Set the max_distance double.

double **getMaxDistance** () const

Return the max_distance double.

bool **loadBrdfAtlas** (const int month, const int atlas_id=-1)

Initialise the BRDF atlas for use with any instrument.

bool **loadBrdfAtlas** (const int month, **rttov::Rttov*** rttov, const int atlas_id=-1)

Initialise the BRDF atlas for a specific instrument (rttov may be an Rttov or RttovSafe object).

bool **loadIrEmisAtlas** (const int month, const int camel_version=3, const int year=2007, const bool ang_corr=false, const int atlas_id=-1)

Initialise the IR emissivity atlas for use with any instrument.

bool **loadIrEmisAtlas** (const int month, **rttov::Rttov*** rttov, const int camel_version=3, const int year=2007, const bool ang_corr=false, const int atlas_id=-1)

Initialise the IR emissivity atlas for a specific instrument (rttov may be an Rttov or RttovSafe object).

bool **loadMwEmisAtlas** (const int month, const int atlas_id=-1)

Initialise the MW emissivity atlas for use with any instrument (TELSEM2)

bool **loadMwEmisAtlas** (const int month, **rttov::Rttov*** rttov, int const year=0, const int atlas_id=-1)

Initialise the MW emissivity atlas for a specific instrument (CNRM MW atlas) (rttov may be an Rttov or RttovSafe object).

void **fillEmisBrdf** (double* emisBrdf, **rttov::Rttov*** rttov, const vector<int> &channels=vector<int>{})

Return emissivities/BRDFs (rttov may be an Rttov or RttovSafe object). emisBrdf is of dimensions [nprofiles][nsurfaces][nchannels].

void **dropAtlas** ()

Deallocate memory for the atlas.

Appendix I: Python *Atlas* class

Methods:

Atlas (verbose=True)
Constructor method.

bool **isAtlasLoaded**()
Returns True if the atlas is loaded.

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, camel_version=3, year=2007, 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, year=0, 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 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][nsurfaces][nchannels]. The inst argument is a loaded Rttov 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 exception 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.

double **MaxDistance**
If MaxDistance is positive and the IR emissivity atlas or BRDF atlas has no data at the given lat/lon location, then a nearby emissivity/BRDF value is returned if one is found within the specified distance

(in km). Default: 0 km (i.e., do not search for nearby values).

bool **Verbose**

Verbosity flag.

Appendix J: Enumeration types (C++) and constants (Python)

Gas units

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** class in C++. This is defined in **wrapper/Rttov_common.h**.

Enumeration constants	Description
unknown	Default initialisation, kg/kg 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

In Python the **gasUnitType** class can be imported from **pyrttov** (defined in **wrapper/pyrttov/rttype.py**) and can be used to specify the gas units. This is used in code as, for example,

```
from pyrttov.rttype import gasUnitType
profiles.GasUnits = gasUnitType('ppmv_wet')
```

where the string argument is one of 'unknown', 'ppmv_dry', 'kg_per_kg', or 'ppmv_wet' as in the table above for the C++ enumeration constants..

ItemIDs (for gas, hydrometeor, and aerosol profiles)

The following table lists the constants of the C++ enumeration **rttov::itemIdType** used for setting gas, hydrometeor, and aerosol profiles in the **setGasItem** method of the **Profiles** class and to obtain the Jacobians for gases, hydrometeor, and aerosol profiles using the **getItemK** method of the **Rttov** and **RttovSafe** classes after running the RTTOV K model. These constants are defined in **wrapper/Rttov_common.h**.

Enumeration constants	Description
Q, O3, CO2, N2O, CO, CH4, SO2	RTTOV variable gases
CLW	Cloud liquid water (for non-scattering MW simulations)
AER1, AER2, ..., AER30	Aerosol particle types 1-30.
INSO, WASO, SOOT, SSAM, SSCM, MINM, MIAM, MICM, MITR, SUSO, VOLA, VAPO, ASDU	The 13 OPAC aerosol particle types.
BCAR, DUS1, DUS2, DUS3, SULP, SSA1, SSA2, SSA3, OMAT	The 9 CAMS aerosol particle types.
HYDRO_FRAC	Hydro fraction for common case with single hydro fraction per layer
HYDRO_FRAC1, HYDRO_FRAC2, ..., HYDRO_FRAC30	Hydro fractions for hydrometeor types 1-30 (for per-hydrometeor fractions)
HYDRO1, HYDRO2, ..., HYDRO30	Hydrometeor types 1-30

STCO, STMA, CUCC, CUCP, CUMA, CLWD, BAUM, BARAN	The hydrometeor types in UV/VIS/IR hydrotable files
MW_RAIN, MW_SNOW, MW_GRAUPEL, MW_CLW, MW_CIW,	The hydrometeor types in MW hydrotable files
HYDRO_DEFF1, HYDRO_DEFF2, ..., HYDRO_DEFF30	Hydrometeor Deff for hydrometeor types 1-30
CLW_DEFF	Hydrometeor Deff for the CLWD hydrometeor type
BAUM_ICE_DEFF	Hydrometeor Deff for the BAUM hydrometeor type

In Python, the **itemIdType** class can be imported from **pyrttov** (defined in **wrapper/pyrttov/rttype.py**) and can be used wherever gas/hydrometeor/aerosol item IDs are required, such as with the getItemK method of the Rttov class. These are used in code as, for example,

```
from pyrttov.rttype import itemIdType
itemIdType('Q')
```

where the string argument can be any of the strings 'Q', 'T', etc listed in the table above for the C++ enumeration constants.

Options

The following table lists C++ enumerations for integer option values which can be compared with the corresponding table in Annex K of the user guide. These are defined in **wrapper/RttovOptions.h**.

Option	**Enumeration constants**	**Description**
InterpMode	interp_rochon (1)	Original RTTOV interpolation method, Jacobians may show oscillations.
	interp_loglinear (2)	May be beneficial in execution time for direct-model calculations, but not suitable for TL/AD/K.
	interp_rochon_loglinear (3)	Similar to mode 1, but with somewhat reduced oscillations.
	interp_rochon_wfn (4)	Default, no oscillations, smaller interpolation errors in general compared to modes 1-3, but most computationally expensive method.
	interp_rochon_loglinear_wfn (5)	No oscillations, but Jacobians may show small “artefacts” due to interpolation, smallest interpolation errors in general, slightly faster than mode 4. Recommended as an alternative to mode 4.
SolarSeaReflModel	solar_refl_model_elfouhaily (1)	Elfouhaily et al. (2017) wave spectrum parameterisation for solar sea surface BRDF model
IrSeaEmisModel	ir_emis_model_isem (1)	ISEM IR sea surface emissivity model
	ir_emis_model_iremis (2)	IREMIS IR sea surface emissivity model
MwSeaEmisModel	mw_emis_model_fastem5 (1)	FASTEM-5 MW sea surface emissivity model

	mw_emis_model_fastem6 (2)	FASTEM-6 MW sea surface emissivity model
	mw_emis_model_surface_ocean (3)	SURFEM-Ocean MW sea surface emissivity model
PermittivityParam (for CLW absorption)	clw_perm_liebe (1)	Liebe (1989) permittivity parameterisation
	clw_perm_rosenkranz (2)	Rosenkranz (2015) permittivity parameterisation
	clw_perm_tkc (3)	Turner, Kneifel, Cadeddu (2016) permittivity parameterisation
ThermalSolver	thermal_solver_dom (1)	DOM thermal solver
	thermal_solver_chou (2)	Chou-scaling solver
	thermal_solver_delta_edd (3)	Delta-Eddington solver
SolarSolver	solar_solver_dom (1)	DOM solar solver
	solar_solver_mfasis_nn (2)	MFASIS neural-network solver
BaranIceVersion	baran2014 (1)	Baran 2014 ice cloud scheme
	baran2018 (2)	Baran 2018 ice cloud scheme
MwPolMode	mw_pol_mode_no_pol (0)	Disable polarised scattering
	mw_pol_mode_empirical (1)	Empirical scaling of extinction for V and H polarised channels, scale factor controlled by **IcePolarisation** option
	mw_pol_mode_aro_scaled (2)	ARO scaled polarisation for V, H, QV, QH polarised channels
OverlapParam	cloud_overlap_auto_select (0)	Automatic selection of overlap_param at run-time: for VIS/IR sensors use max/random and for MW sensors use two-column hydro-weighted scheme.
	cloud_overlap_max_random (1)	Maximum-random overlap, recommended for VIS/IR sensors
	cloud_overlap_2col_max (2)	Two column, effective frac is maximum value among layers at pressures below **TwoColMaxFracMaxP** option
	cloud_overlap_2col_weighted (3)	Two column, effective frac is computed as **hydrometeor-weighted** average, recommended for MW sensors
	cloud_overlap_2col_user (4)	Two column, user specifies effective frac in **HydroFracEff** profile variable

In Python, the same constants can be imported from **pyrttov.option** (defined in **wrapper/pyrttov/option.py**), the only difference being that in Python the constants are all in upper case, so that for example, to select the Chou-scaling thermal solver you can do:

```
from pyrttov.option import THERMAL_SOLVER_CHOU
myRttov.Options.ThermalSolver = THERMAL_SOLVER_CHOU
```

Appendix K: Gas IDs

Gas ID list: these are defined in src/wrapper/rttov_wrapper_handle.F90. See user guide Annex J for more information about the profile variables and user guide section 8.4 for information about the hydrometeor and aerosol types. These IDs are not required when using the object-oriented interfaces.

ID	Variable
1	Water vapour (q)
2	Ozone (O3)
3	CO2
4	N2O
5	CO
6	CH4
7	SO2
15	Cloud liquid water (clw) – non-scattering MW simulations
101-130	Aerosol concentration for particle types 1-30
101-113	Aerosol concentration for OPAC aerosol particle types 1-13
101-109	Aerosol concentration for CAMS aerosol particle types 1-9
201-230	Hydro fraction for hydrometeor particle types 1-30 (for per-hydrometeor fractions)
201	Hydro fraction for common case with single hydro fraction per layer
301-330	Hydrometeor concentration for hydrometeor particle types 1-30
301-308	Hydrometeor concentration for UV/VIS/IR hydrometeor particle types
301-305	Hydrometeor concentration for MW hydrometeor particle types
401-430	Hydro Deff for hydrometeor particle types 1-30
406	Hydro Deff for Clw Deff liquid cloud hydrometeor type
407	Hydro Deff for Baum ice cloud hydrometeor type

Appendix L: RTTOV wrapper subroutines

The following table lists the main subroutines coded in Fortran in the underlying RTTOV wrapper. These subroutines are called internally by the object-oriented interfaces.

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_call_direct_scatt_optp	Call the RTTOV direct model for scattering with explicit optical properties
rttov_call_k_scatt_optp	Call the RTTOV K model for scattering with explicit optical properties
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_asym	Calculate asymmetry parameter from given phase function
rttov_lcoef	Calculate Legendre coefficients from given phase function
rttov_get_aer_clim_prof	Generate climatological aerosol profiles based on the OPAC species
rttov_get_major_version	Return the RTTOV major version number
rttov_get_minor_version	Return the RTTOV minor version number

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, store_diag_output, 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 surface emissivity/reflectance arrays, and the members of the RTTOV transmission, radiance, radiance2, diagnostic output, and emissivity retrieval structures returned: see Annex J in the user guide for more information about the RTTOV data structures.

Output surface emissivities and reflectances:

Subroutine	**Array argument and dimensions in C index order**
rttov_get_emis_out	emis_out(nprofiles, nsurfaces, nchannels)
rttov_get_brdf_out	brdf_out(nprofiles, nsurfaces, nchannels)
rttov_get_diffuse_refl_out	diffuse_refl_out(nprofiles, nsurfaces, nchannels)

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)

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 UV/VIS/NIR channels in the btrefl argument to the rttov_call_* subroutines

rttov_get_overcast	radiance%overcast(nchanprof, nlayers)
rttov_get_bt_overcast	radiance%bt_overcast(nchanprof, nlayers)
rttov_get_rad_quality	radiance%quality(nchanprof) – integer array

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)

Diagnostic output structure members:

Subroutine	**Array argument and dimensions in C index order**
rttov_get_diag_output_geometric_height	diag_output%geometric_height(nprofiles, nlayers)
rttov_get_diag_output_geometric_height_half	diag_output%geometric_height_half(nprofiles, nlevels)
rttov_get_diag_output_hydro_frac_eff	diag_output%hydro_frac_eff(nprofiles)

Radar reflectivity structure members:

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

Emissivity retrieval structure members:

Here *ncolumns* is 1 for non-hydrometeor scattering simulations, 2 for hydrometeor scattering with two-column cloud overlap, and *2*nlayers+1* for hydrometeor scattering with max/random overlap.

Subroutine	**Array argument and dimensions in C index order**
rttov_get_emis_terms_bsfc	emis_terms%bsfc(nchanprof)
rttov_get_emis_terms_column_weight	emis_terms%column_weight(nchanprof, ncolumns)
rttov_get_emis_terms_tau_sfc	emis_terms%tau_sfc(nchanprof, ncolumns)
rttov_get_emis_terms_rad_up	emis_terms%rad_up(nchanprof, ncolumns)
rttov_get_emis_terms_rad_down	emis_terms%rad_down(nchanprof, ncolumns)

--END--