Python/C/C++ wrapper

for RTTOV v11.3

James Hocking, Pascale Roquet, Pascal Brunel

This documentation was developed within the context of the EUMETSAT Satellite Application Facility on Numerical Weather Prediction (NWP SAF), under the Cooperation Agreement dated 1 December, 2006, 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, KNMI and Météo France.

Change record			
Version	Date	Author / changed by	Remarks
1.0	2015-09-22	J Hocking	First version.

Table of contents

1. Introduction

A new interface has been written for RTTOV v11.3 which allows RTTOV simulations using the direct and K models to be run from Python, C or C++ . It is possible to use this interface to run RTTOV without writing any Fortran code. C++ classes have been created which allow you to interact with RTTOV in an object-oriented style rather than calling the wrapper interface subroutines directly. Similar Python classes are in preparation and will be posted on the website after RTTOV v11.3 is released.

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

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

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

You can call the interface subroutines directly as described in section 3. Sections 4 and 5 provide additional information specific to Python and C/C++ respectively.

Alternatively a collection of C++ classes have been created which enable RTTOV to be called using object-oriented-style programming. These classes are described in section 6. This is a more user-friendly way of calling RTTOV through the wrapper. As noted above, these are being ported to Python and will be made available when complete.

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

Section 7 outlines the current limitations of the wrapper. Finally, the appendices provide some additional information about the Fortran-Python/C/C++ interface and the object-oriented classes.

Currently the wrapper supports calls to rttov_direct and rttov_k for clear-sky and IR scattering calculations optionally including use of the surface emissivity and BRDF atlases. In the future support for RTTOV-SCATT and PC-RTTOV will be added.

2. Compilation and example code

The wrapper Fortran source code is contained in the src/wrapper/ directory. This code depends on the RTTOV emissivity and BRDF atlases which in turn depend on the NetCDF or, as of RTTOV v11.3, the HDF5 library. The HDF5 format atlas files are the recommended ones to use because they are much smaller in size than the NetCDF equivalents.

The easiest way to compile RTTOV is to edit the file build/Makefile.local to point to either your HDF5 (recommended) or NetCDF installation and then do:

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

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

Compiling C/C++ code which calls RTTOV

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

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

Running Python code which calls RTTOV

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

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

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

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

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

interface_example_cpp.cpp	Example of calling interface directly in C++
interface_example_c.c	Example of calling interface directly in C
interface_example_python.py	Example of calling interface directly in Python
Rttov_example.cpp	Example of using Rttov class in C++
RttovSafe_example.cpp	Example of using RttovSafe class in C++
Makefile	Makefile to compile all C and C++ examples

These can be used as templates for 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 and/or the RTTOV libraries.

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

RttovSafe.h, RttovSafe.cpp	Class allowing you to call RTTOV for an instrument – carries out some checks on the profiles to help prevent errors.
Profile.h, Profile.cpp	Class representing a single profile for use with RttovSafe
Rttov.h, Rttov.cpp	Class allowing you to call RTTOV – limited error checking.
Profiles.h, Profiles.cpp	Class representing one or more profiles for use with Rttov
Options.h, Options.cpp	Class representing RTTOV and wrapper options

3. General description of interface

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

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

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

It is important to realise that although the instruments are independent, the emissivity and BRDF atlases are shared between all loaded instruments. More details are given below.

The only restriction on the interface (aside from the available memory) is that you may initialise a maximum of 100 instruments concurrently.

3.1. Initialising the wrapper

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

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

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

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

Notes:

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

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

Specifying the options string

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

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

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

Key	**Value**	**Description**
file_coef	Full path to rtcoef file	Mandatory, path to rtcoef file.
file_scaer	Full path to IR aerosol coef file	For IR aerosol simulations, path to scaer coef file.
file_sccld	Full path to IR cloud coef file	For IR cloudy simulations, path to sccld coef file.

RTTOV options - *optional*

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

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

Wrapper-specific options - *optional*

Set options that are related specifically to the wrapper:

Key	Value	Description
verbose_wrapper	0 or 1	Set to 1 for more verbose output from the wrapper (default 0, all output suppressed except fatal error messages).
nthreads	Integer	If <=1 RTTOV is called via the standard interface (e.g. rttov_direct), if >1 RTTOV is called via the parallel interface (e.g. rttov_parallel_direct) using the specified number of threads (default 1).
nprofs_per_call	Integer – greater than 0	Sets the number of profiles passed to each call to rttov_direct or rttov_k *within* the wrapper (default 1).
check_opts	0 or 1	If set to 1 the Fortran `rttov_user_options_checkinput` subroutine (see user guide Annex N) is called to help ensure consistency between the selected options and the loaded coefficient file (default 1).
store_trans	0 or 1	Set to 1 to enable access to transmittance outputs from RTTOV calls (default 0).
store_rad	0 or 1	Set to 1 to enable access to radiance outputs from RTTOV direct model calls (default 0).
store_rad2	0 or 1	Set to 1 to enable access to secondary radiance outputs from RTTOV direct model calls (default 0). If this is set to 1 then store_rad will automatically be set to 1 as well.

Notes:

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

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

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

If you are performing cloud or aerosol scattering simulations you must ensure the addclouds and/or addaerosl RTTOV options are set in the options string when loading the instrument.

3.2. Changing RTTOV options

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

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

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

You can change any options in the options structure and any of the wrapper-specific options in this call. Setting the coefficient file names has no effect in a call to `rttov_set_options`. Options that were previously set are retained so you only need to specify options you wish to change.

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_rttov11/rttov9pred54L/rtcoef_msg_3_seviri.dat ' \
  'opts%interpolation%addinterp 1 ' \
  'opts%rt_ir%o3_data 1 '           \
  'opts%rt_ir%addsolar 1 '          \
  'nthreads 4 '
```

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

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

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

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

3.3. Using the emissivity and/or BRDF atlases

In order to use the emissivity or BRDF atlases they must be initialised before any calls to RTTOV. There are separate subroutine to set up the BRDF, IR emissivity and MW emissivity atlases.

```
rttov_ir_emis_atlas_setup(err, path, month, version, inst_id, ang_corr)
rttov_mw_emis_atlas_setup(err, path, month, version, inst_id)
rttov_brdf_atlas_setup(err, path, month, version, inst_id)
```

Argument	Type	Intent	Description
err	Integer	out	Return code: non-zero implies error condition.
path	Character string	in	String containing path to atlas data files.
month	Integer	in	Month (1-12) for which to initialise atlas.
version	Integer	in	Atlas version number, set to -1 for default version.
inst_id	Integer	in	ID of instrument (as returned by rttov_load_inst) of instrument for which to initialise atlas (may be 0: see below).
ang_corr	Integer	in	IR atlas only: set non-zero to include the zenith angle emissivity correction (see user guide for more information).

Notes:

The emissivity and BRDF atlases may each be initialised once and are subsequently shared by all initialised instruments until they are deallocated (see section 3.6).

For the IR emissivity and BRDF atlases, currently only one version exists so you can set the version number to -1 for these.

For the MW emissivity atlas, there is a choice between the TELSEM and CNRM MW atlases (see the user guide for the version numbers). TELSEM is the default and can be selected by providing a version number of -1.

It is important to note that the CNRM atlas is always initialised for a specific instrument (see the user guide for details). Therefore you **must** provide a valid inst_id if initialising the CNRM MW atlas and it is not advisable to try to use this atlas for multiple different loaded MW instruments.

The TELSEM atlas is never initialised for any specific MW instrument. Therefore you can always supply an inst_id of 0 for this atlas and you can use this with any number of loaded MW instruments.

The IR emissivity and BRDF atlases may be initialised for a specific instrument by specifying a valid inst_id for a previously loaded instrument in which case the calls to the atlas are significantly faster. Alternatively they may be initialised for multiple instruments by specifying an inst_id of 0: in this case they can be used for any loaded VIS/IR instruments.

If performance is critical you should consider the single-instrument initialisation for the IR emissivity and BRDF atlases. However if you initialise these atlases for one inst_id and you try to use them with a different inst_id you will obtain spurious emissivities/BRDFs.

3.4. Calling the RTTOV direct model

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

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

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

icecloud(nprofiles,2)	Integer	in	(ish, idg) for each profile.
zeeman(nprofiles,2)	Real	in	(Be, cosbk) for each profile.
p(nprofiles,nlevels)	Real	in	Pressure levels for each profile.
t(nprofiles,nlevels)	Real	in	Temperature on pressure levels for each profile.
gas_id(ngases)	Integer	in	List of IDs for gases, aerosol and cloud profiles present in the gases array, see below.
gases(nprofiles,nlevels,ngases)	Real	in	Gas, aerosol and cloud concentrations on levels/layers for each profile: must contain at least water vapour profiles, see below.
surfemisrefl(2,nprofiles,nchannels)	Real	inout	Input surface emissivity and BRDF values for each channel; on output contains the values used by RTTOV, see below.
btrefl(nprofiles,nchannels)	Real	inout	Output total TOA brightness temperatures (for all channels at wavelengths > 3µm) or reflectances (wavelengths < 3µm).
rads(nprofiles,nchannels)	Real	inout	Output total TOA radiances.
nchannels	Integer	in	Number of channels to simulate (not required in Python).
ngases	Integer	in	Size of gas_id(:) array, see below (not required in Python).
nlevels	Integer	in	Number of levels in input profiles (not required in Python).
nprofiles	Integer	in	Number of profiles being passed in (not required in Python).

Notes:

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

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

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

Surface emissivity/BRDF

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

The surfemisrefl(0,:,:) and surfemisrefl(1,:,:) arrays are used to control the input or calculation of surface emissivities and BRDFs respectively for all channels for each profile. If you provide non-negative (i.e. >=0) values for any channel then calcemis (or calcrefl) will be set to false for that channel and the supplied value is used for the surface emissivity (or BRDF). If a value in surfemisrefl(:,:,:) is negative then for profiles over sea (as defined by skin%surftype), calcemis (or calcrefl) will be set to true. Over land and sea-ice, the relevant atlas will be used to provide the emissivity (or BRDF) value if it was initialised, otherwise calcemis (or calcrefl) will be set to true.

On exit from the subroutine call the surfemisrefl array is overwritten with the emissivity and BRDF values used by RTTOV: these will be identical to the input values where the input values were non-negative, otherwise they will be the values calculated by RTTOV's internal surface models or the values obtained from the respective atlases.

NB When making multiple calls to the wrapper interface be sure to re-initialise the surfemisrefl array appropriately between calls to avoid inadvertently passing in emissivity and BRDF values from the previous call.

Specifying gas, aerosol and cloud profiles

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

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

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

Also note that aerosol and cloud inputs are on *layers* rather than *levels*: profiles of these variables should be written to the first nlayers values in the array, the final value (at nlevels) is ignored.

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

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

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

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

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

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

Outputs

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

3.5. Calling the RTTOV K model

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

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

Argument	**Type**	**Intent**	**Description**
skin_k(nprofiles,9)	Real	inout	Calculated Jacobians for (skin%t, skin%salinity, snow_frac, skin%foam_fraction, skin%fastem(1:5)) for each profile.
s2m_k(nprofiles,6)	Real	inout	Calculated Jacobians for (s2m%p, s2m%t, s2m%q, s2m%u, s2m%v, s2m%wfetc) for each profile.
simplecloud_k(nprofiles,2)	Integer	inout	Calculated Jacobians for (ctp, cfraction) for each profile.
zeeman_k(nprofiles,2)	Real	inout	Calculated Jacobians for (Be, cosbk) for each profile.
p_k(nprofiles,nlevels)	Real	inout	Calculated Jacobians for pressure for each profile.
t_k(nprofiles,nlevels)	Real	inout	Calculated Jacobians for temperature for each profile.
gases_k(nprofiles,nlevels,ngases)	Real	inout	Calculated Jacobians for gas, aerosol and cloud, variable order matches the input gas_id and gases arrays, see above.
surfemisrefl_k(2,nprofiles,nchannels)	Real	inout	Calculated Jacobians for surface emissivity and BRDF.
bt_k(nprofiles,nchannels)	Real	in	Input BT perturbations (only for channels at wavelengths > 3µm).
rads_k(nprofiles,nchannels)	Real	in	Input radiance perturbations.

Notes:

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

3.6. 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 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 intent(out) return code (non-zero implies an error condition) and inst_id is the ID of the instrument to deallocate.

You can deallocate any atlases you initialised using:

```
rttov_ir_emis_atlas_dealloc()

rttov_mw_emis_atlas_dealloc()

rttov_brdf_atlas_dealloc()
```

These subroutines have no arguments: they are safe to call even if the corresponding atlas was not initialised.

4. Specific information for Python

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

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

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

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

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

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

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

5. Specific information for C/C++

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

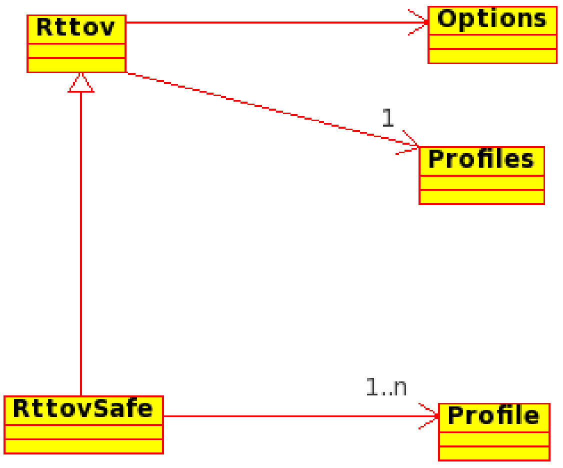
When passing a character string argument to Fortran from C/C++ it is necessary to include the string length as an additional argument. Usually this is appended as the final argument in the call, but for some compilers it may need to be supplied directly following the string argument. See the example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

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

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

6. RTTOV classes

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



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

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

The **RttovSafe** object provides a safer way to call RTTOV because it carries out some checks on the input profiles before passing them to the RTTOV interface. This is a more user-friendly, but 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.

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. Through these classes it is also possible to use the RTTOV land surface emissivity and BRDF atlases.

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

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

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

6.1. General method for calling RTTOV

An instance, say “myRttov”, of either the **Rttov** or **RttovSafe** classes should be declared. Each such instance represents a single instrument to simulate. The methods of the **RttovSafe** and **Rttov** classes are given in Appendix C: the majority of methods are common to both classes. The difference is in the way the profile data are associated with instances of each class.

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

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

Each of these steps is described in more detail below.

6.2. Setting RTTOV options

This myRttov object has a member named “options” which is an instance of the **Options** class. This is used to specify the RTTOV and wrapper-specific options. The methods of this class are listed in Appendix F. The user guide describes the RTTOV options (see Annex O). See section 3.1 above for a description of the wrapper-specific options.

To change an option associated with an **Rttov** or **RttovSafe** instance named “myRttov” you should use, for example:

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

6.3. Loading an instrument

The name of the optical depth (“rtcoef_”) coefficient file should be specified by calling the **myRttov.setFileCoef** method. If required the IR cloud and/or aerosol coefficient file names should also be specified using the **setFileSccld** and **setFileScaer** methods respectively.

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 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 any of the options can still be changed. If you call the **myRttov.updateOptions** method and the wrapper “check_opts” option is set to true this will force a consistency check on the options and loaded coefficients and will report any errors which can be useful for debugging simulations. The **myRttov.printOptions** method will print out the options structure (this calls the rttov_print_opts Fortran subroutine).

6.4. Specifying surface emissivities and reflectances

You can pass your own values for surface emissivity and/or reflectance into RTTOV or RTTOV can provide suitable values. The user guide provides full details of the treatment of surface emissivity and reflectance. You should allocate an array **surfemisrefl** with dimensions *[2][nprofiles][nchannels].*This should be initialised before every call to RTTOV. The first dimension of this array provides access to emissivities (index 0) and reflectances (index 1) for all channels and profiles being simulated. Where values in this input array are greater than or equal to zero the corresponding elements of the RTTOV calcemis and calcrefl arrays will be set to false and these input values of the surface parameters will be used for the simulations. Where the values in **surfemisrefl** are less than zero the internal sea surface emissivity and reflectance models are used for sea surfaces. For land and sea-ice surfaces, if the atlases have been initialised (see next section) these will be used to provide values, otherwise RTTOV will supply default values (as described in the user guide). The use of the atlases is described in section 6.2.

Once RTTOV has been called the **surfemisrefl** array contains the values that were used by RTTOV. Where you supplied non-negative values the elements of the array will be unchanged.

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

6.5. Using the emissivity and BRDF atlases

The emissivity and BRDF atlases may each be initialised once and are subsequently shared by all instruments represented by instances of RttovSafe and Rttov until they are deallocated.

In order to use one or more of the atlases the paths to the atlases must first be specified using the **myRttov.setBrdfAtlasPath** and the **myRttov.setEmisAtlasPath** methods.

Once the instrument has been loaded the atlas(es) can be initialised. This is achieved by calling the **myRttov.brdfAtlasSetup** and/or **myRttov.irEmisAtlasSetup** methods for visible/IR sensors and the **myRttov.mwEmisAtlasSetup** method for MW instruments. There are alternative interfaces for these methods: the simplest take no arguments and they will obtain the month for which to initialise the atlas from the first profile's month (this requires the profile data to have been loaded before the atlases are initialised – see next section). Alternatively you can explicitly select the month for which to initialise the atlas in the setup method calls and also optionally specify other options. The atlas setup methods return boolean values indicating success (true) or failure (false).

In order to use the atlases for land and sea-ice surfaces the input emissivities and/or BRDFs in the **surfemisrefl** array should be set to negative numbers (over sea surfaces the internal RTTOV sea surface emissivity and reflectance models will be used).

When the destructor of an **RttovSafe** and **Rttov** instance is called it will invoke the emissivity and BRDF atlas deallocation subroutines for those atlases which were successfully initialised through the setup methods of the same instance. Therefore if you destroy an **RttovSafe** or **Rttov** object which was used to load the atlases then you must re-initialise the atlases if you want to continue using them with other existing **RttovSafe** or **Rttov** objects. You can manually deallocate the atlas data by calling the **myRttov.deallocBrdfAtlas**, **myRttov.deallocIrEmisAtlas** or **myRttov.deallocMwEmisAtlas** methods, but again note that you can only deallocate an atlas using the method of the instance which was used to allocate that atlas. This is useful if you want to re-initialise an atlas for a different month or if you want to re-initialise an atlas for use with a different instrument (see below), for example.

By default the BRDF and IR emissivity atlases are initialised so that they can be used with all concurrently loaded visible/IR instruments. They can optionally be initialised for use with a single instrument by setting the “single_inst” boolean argument to true when initialising the atlases. This is significantly faster and can be very beneficial if running RTTOV for many profiles. However it is very important to understand that if the atlases are initialised for a particular instrument with a given set of loaded channels ***then they cannot be used by another instance*** of **RttovSafe** or **Rttov** unless that instance represents exactly the same set of channels for the same instrument.

You should be careful when using the CNRM MW atlas with multiple instances of **RttovSafe** or **Rttov** because this atlas is ***always*** initialised for a specific instrument (see the user guide). It is not recommended to use this atlas with multiple instances of the **RttovSafe** or **Rttov** classes.

By contrast the TELSEM atlas (the default MW emissivity atlas) is never initialised for a single instrument and can always be used by all loaded MW instruments.

6.6. Profile data for an RttovSafe object

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 D.

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

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

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

When doing IR cloud and/or aerosol simulations the cloud, cfrac and aerosol profiles input to RTTOV are defined on atmospheric layers. However they must be supplied to the **Profile** object as an array of *nlevels* elements: the final element is ignored.

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

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

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

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

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

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

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

6.7. Profile data for an Rttov object

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

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

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

The data for each profile variable is provided to the Profiles instance as a pointer to an array containing the data for every profile using the relevant method. For example, the **setT** method assigns the temperature profiles to the **Profiles** instance.

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

When doing IR cloud and/or aerosol simulations the cloud, cfrac and aerosol profiles input to RTTOV are defined on atmospheric layers. However they must be supplied to the **Profiles** object as arrays of *[nprofiles][nlevels]* elements (as for temperature and gases): the final element of each profile is ignored. To supply the cloud and aerosol profiles you must use the **setGasItem** method which takes the profile as input and an ID for the profile variable being set. This second argument is of type **rttov::itemIdType**: this enumeration is defined in wrapper/rttov_common.h and a complete list of the associated constants is given in Appendix G. (You can also set the gas profiles using this method, but it is clearer to use the methods like **setQ** which are particular to each gas).

The **setGasUnits** method takes an integer argument: see the RTTOV user guide for valid values. If unspecified the default is ppmv over moist air, but a warning is printed if you do not set this explicitly.

The **setAngles**, **setS2m**, **setSkin, setSurfType**, **setSurfGeom** and **setDateTimes** methods must all be called for each **Profiles** instance. Each of these methods sets a collection of related profile variables. The argument to each method is a two dimensional array (see Appendix E). The first dimension is *nprofiles*, and the second dimension depends on the number of variables being set by each method. 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**, **setIceCloud** and **setZeeman** methods do not need to be called unless you require the corresponding variables to be specified in your simulations. If unspecified the **Profiles** object will set the values of the corresponding profile variables to zero.

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

Once all the necessary profile data have been specified in the Profiles instance it can be associated with the RttovSafe or Rttov object 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.

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.

6.8. Calling RTTOV

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

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

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

6.9. Accessing RTTOV outputs

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

After calling the RTTOV K model the Jacobians can be obtained through the various methods listed in Appendix C. For example the temperature Jacobians are obtained using the **myRttov.getTK** method.

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

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

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

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.

It is also possible to access the full contents of the RTTOV transmission, radiance and radiance2 structures. You must set the relevant flag (store_trans, store_rad, store_rad2) before calling RTTOV otherwise calls to these methods will throw an exception. Each method returns a vector of values for a given profile index or for given profile and channel indices.

6.10. 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.

All instances of these classes are completely independent except, as noted above, in respect of the emissivity and BRDF atlases. It is not necessary to call the atlas deallocation methods as these are executed by the **RttovSafe** and **Rttov** destructors, but as described above you must be aware that the atlases will be deallocated if you destroy the **RttovSafe** or **Rttov** instance that was used to initialise them.

7. Limitations of the wrapper

The wrapper currently has the following limitations:

- Up to 100 instruments may be initialised simultaneously*.
- RTTOV-SCATT unavailable.
- PC-RTTOV unavailable.
- TL/AD unavailable.
- Cannot pass aerosol/cloud optical parameters explicitly for IR scattering simulations (“method 2”).

*For the object-oriented interface this implies a maximum of 100 concurrently instantiated Rttov or RttovSafe objects. If this limitation is problematic you can modify the value of the parameter max_ninst in src/wrapper/rttov_wrapper_handle.F90 to a larger number and recompile RTTOV to increase this limit.

Appendix A: Gas IDs

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

ID	Variable	nlevels or nlayers*
1	Water vapour (q)	nlevels
2	Ozone (o3)	nlevels
3	CO2	nlevels
4	N2O	nlevels
5	CO	nlevels
6	CH4	nlevels
15	Cloud liquid water (clw)	nlevels
20	Cloud fraction (cfrac)	nlayers
21-25	Cloud liquid water types 1-5 (STCO, STMA, CUCC, CUCP, CUMA)	nlayers
30	Ice cloud (CIRR)	nlayers
31	Ice cloud effective diameter (icede)	nlayers
41-53	Aerosol particle types 1-13	nlayers

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

Appendix B: RTTOV wrapper subroutines

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

Subroutine	**Description**
rttov_load_inst	Specify initial RTTOV and wrapper options and load an instrument
rttov_set_options	Modify one or more RTTOV and wrapper options
rttov_print_options	Print the current RTTOV and wrapper options
rttov_call_direct	Call the RTTOV direct model
rttov_call_k	Call the RTTOV K model
rttov_drop_inst	Deallocate the data for a specified instrument
rttov_drop_all	Deallocate all instrument and atlas data
rttov_brdf_atlas_setup rttov_ir_emis_atlas_setup rttov_mw_emis_atlas_setup	Initialise the BRDF and emissivity atlases
rttov_brdf_atlas_dealloc rttov_ir_emis_atlas_dealloc rttov_mw_emis_atlas_dealloc	Deallocate the BRDF and emissivity atlases

The main subroutine calls to the direct and K model 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 and/or store_rad2 wrapper options. They can be accessed via one of the subroutine calls listed below. Note that these outputs are stored independently for each instrument, but for any given instrument they are overwritten by any subsequent direct or K model calls for that instrument.

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

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

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

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

Radiance structure members:

Subroutine	Array argument and dimensions in C index order
rttov_get_rad_clear	radiance%clear(nchanprof)
rttov_get_rad_total	radiance%total(nchanprof) – this is returned in the rads argument to the rttov_call_* subroutines
rttov_get_rad_cloudy	radiance%cloudy(nchanprof)
rttov_get_bt_clear	radiance%bt_clear(nchanprof)
rttov_get_bt	radiance%bt(nchanprof) – this is returned for IR/MW channels in the btrefl argument to the rttov_call_* subroutines
rttov_get_refl_clear	radiance%refl_clear(nchanprof)
rttov_get_refl	radiance%refl(nchanprof) – this is returned for VIS/NIR channels in the btrefl argument to the rttov_call_* subroutines
rttov_get_overcast	radiance%overcast(nchanprof, nlayers)

Radiance2 structure members:

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

Transmission structure members:

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

Appendix C: *RttovSafe* and *Rttov* classes

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

Constructors:

RttovSafe ()
RttovSafe class constructor method.

Rttov ()
Rttov class constructor method.

Associating profile data with an *RttovSafe* object:

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

Associating profile data with an *Rttov* object:

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

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

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

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

void **setFileScaer** (const string &fileScaer)
Set the aerosol coefficient filename.

void **setBrdfAtlasPath** (const string &brdfAtlasPath)
Set the path for the BRDF atlas files.

void **setEmisAtlasPath** (const string &emisAtlasPath)
Set the path for the emissivity atlas files.

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.

void **updateOptions** ()
Update RTTOV options for the currently loaded instrument.

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

bool **brdfAtlasSetup** (int month, bool single_inst=false, int version=-1)
Initialise the BRDF atlas (all options available)

bool **irEmisAtlasSetup** (int month, bool ang_corr=false, bool single_inst=false, int version=-1)
Initialise the IR emissivity atlas (all options available)

bool **mwEmisAtlasSetup** (int month, int version=-1)
Initialise the MW emissivity atlas.

bool **brdfAtlasSetup** ()
Initialise the BRDF atlas for the month of the first profile.

bool **irEmisAtlasSetup** ()
Initialise the IR emissivity atlas for the month of the first profile.

bool **mwEmisAtlasSetup** ()
Initialise the MW emissivity atlas for the month of the first profile.

void **deallocBrdfAtlas** ()
Deallocate memory for the BRDF atlas.

void **deallocIrEmisAtlas** ()
Deallocate memory for the IR emissivity atlas.

void **deallocMwEmisAtlas** ()
Deallocate memory for the MW emissivity atlas.

void **setSurfEmisRefl** (double ***surfemisrefl**)
Set pointer to array containing input/output surface emissivity and reflectance values; this must be previously allocated a double array of dimensions [2][nprofiles][nchannels]; this is used to pass emissivity/reflectance values into RTTOV; if this is not called the ***RttovSafe*** *object will allocate an array containing the values used by RTTOV which can be accessed by getSurfEmisRefl.*

void **printGases** ()
Print gases array contents on standard output.

void **runDirect** ()
Run the RTTOV direct model for all channels.

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

Run the RTTOV direct model for a list of channels.

void **runK** ()
: *Run the RTTOV K model for all channels.*

void **runK** (const vector< int > &channels)
: *Run the RTTOV K model for a list of channels.*

std::vector< double > **getBtRefl** (const int profile)
: *Return vector of brightness temperatures/reflectances computed by the previous run for the given profile number.*

std::vector< double > **getRads** (const int profile)
: *Return a vector of radiances computed by the previous run for the given profile number.*

const double * **getSurfEmisRefl** () const
: *Return a pointer to an array of dimensions [2][nprofiles][nchannels] containing output values of surface emissivity and reflectance; 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 **RttovSafe** object it is deleted at the next run or when the **RttovSafe** instance is destroyed.*

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

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

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

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

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

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

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

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

std::vector< double > **getTauTotal** (int profile)
: *Return RTTOV transmission tau_total output array of size [nchannels] for given profile, requires store_trans true.*

std::vector< double > **getTauLevels** (int profile, int channel)
: *Return RTTOV transmission tau_levels output array of size [nlevels] for given profile and channel, requires store_trans true.*

std::vector< double > **getTauSunTotalPath1** (int profile)
: *Return RTTOV transmission tausun_total_path1 output array of size [nchannels] for given profile, requires store_trans true.*

std::vector< double > **getTauSunLevelsPath1** (int profile, int channel)
: *Return RTTOV transmission tausun_levels_path1 output array of size [nlevels] for given profile and channel, requires store_trans true.*

std::vector< double > **getTauSunTotalPath2** (int profile)
: *Return RTTOV transmission tausun_total_path2 output array of size [nchannels] for given profile, requires store_trans true.*

std::vector< double > **getTauSunLevelsPath2** (int profile, int channel)
: *Return RTTOV transmission tausun_levels_path2 output array of size [nlevels] for given profile and channel, requires store_trans true.*

std::vector< double > **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 > **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.

Appendix D: *Profile* class (used with *RttovSafe* objects)

Profile (int nlevels)
: *Constructor method.*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

void **setS2m** (const double p_2m, const double t_2m, const double q_2m, const double u_10m, const double v_10m, const double wind_fetch)
: *Set surface 2m and 10m parameters.*

void **setSkin** (const double t, const double salinity, const double snow_fraction, const double foam_fraction, const double fastem_coef_1, const double fastem_coef_2, const double fastem_coef_3, const double fastem_coef_4, const double fastem_coef_5)
: *Set skin parameters.*

void **setSurfType** (const int surftype, const int watertype)
: *Set surface type parameters.*

void **setSurfGeom** (const double lat, const double lon, const double elevation)
: *Set surface geometry parameters.*

void **setDateTimes** (const int yy, const int mm, const int dd, const int hh, const int mn, const int ss)
Set date and time.

void **setSimpleCloud** (const double ctp, const double cfraction)
Set simple cloud parameters.

void **setIceCloud** (const int ish, const int idg)
Set ice cloud parameters.

void **setZeeman** (const double Be, const double cosbk)
Set zeeman parameters.

Appendix E: *Profiles* class (used with *Rttov* objects)

Profiles (int nbprofiles, const int nblevels)

Constructor method for individual gas specification.

void **setGasUnits** (int gasUnits)

Set the gas_units.

void **setP** (double *p)

Set the pointer to the p array of size [nprofiles][nlevels].

void **setT** (double *t)

Set the pointer to the t array of size [nprofiles][nlevels].

void **setQ** (double *q)

Set the pointer to the q array of size [nprofiles][nlevels].

void **setO3** (double *o3)

Set the pointer to the o3 array of size [nprofiles][nlevels].

void **setCO2** (double *co2)

Set the pointer to the co2 array of size [nprofiles][nlevels].

void **setCO** (double *co)

Set the pointer to the co array of size [nprofiles][nlevels].

void **setN2O** (double *n2o)

Set the pointer to the n2o array of size [nprofiles][nlevels].

void **setCH4** (double *ch4)

Set the pointer to the ch4 array of size [nprofiles][nlevels].

void **setCLW** (double *clw)

Set the pointer to the clw array of size [nprofiles][nlevels].

void **setAngles** (double *angles)

Set the pointer to the angles array of size [nprofiles][4] containing satzen, satazi, sunzen, sunazi for each profile.

void **setS2m** (double *s2m)

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

void **setSkin** (double *skin)

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

void **setSurfType** (int *surftype)

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

void **setSurfGeom** (double *surfgeom)

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

void **setDateTimes** (int *datetimes)

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

void **setSimpleCloud** (double *simplecloud)

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

void **setIceCloud** (int *icecloud)

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

void **setZeeman** (double *zeeman)

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

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

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

Appendix F: *Option* class

The methods listed below are used to set the RTTOV and wrapper options. Methods also exist to query the options: see wrapper/Options.h.

Options ()

Constructor method.

void **setApplyRegLimits** (bool applyRegLimts)

Set the opts%config%apply_reg_limits option.

void **setDoCheckinput** (bool doCheckinput)

Set the opts%config%do_checkinput option.

void **setVerbose** (bool verbose)

Set the opts%config%verbose option.

void **setAddInterp** (bool addinterp)

Set the opts%interpolation%addinterp option.

void **setInterpMode** (int interpMode)

Set the opts%interpolation%interp_mode option.

void **setRegLimitExtrap** (bool regLimitExtrap)

Set the opts%interpolation%reg_limit_extrap option.

void **setSpacetop** (bool spacetop)

Set the opts%interpolation%spacetop option.

void **setLgradp** (bool lgradp)

Set the opts%interpolation%lgradp option.

void **setDoLambertian** (bool doLambertian)

Set the opts%rt_all%do_lambertian option.

void **setUseQ2m** (bool useQ2m)

Set the opts%rt_all%use_q2m option.

void **setSwitchrad** (bool switchrad)

Set the opts%rt_all%switchrad option.

void **setAddRefrac** (bool addRefrac)

Set the opts%rt_all%addrefrac option.

void **setCLWData** (bool clwData)

Set the opts%rt_mw%clw_data option.

void **setFastemVersion** (int fastemVersion)

Set the opts%rt_mw%fastem_version option.

void **setSupplyFoamFraction** (bool supplyFoamFraction)

Set the opts%rt_mw%supply_foam_fraction option.

void **setOzoneData** (bool ozoneData)

Set the opts%rt_ir%ozone_data option.

void **setCO2Data** (bool co2Data)

Set the opts%rt_ir%co2_data option.

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

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

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

void **setAddSolar** (bool addsolarl)
Set the opts%rt_ir%addsolar option.

void **setDoNlteCorrection** (bool doNlteCorrection)
Set the opts%rt_ir%do_nlte_correction option.

void **setAddAerosl** (bool addaerosl)
Set the opts%rt_ir%addaerosl option.

void **setAddClouds** (bool addclouds)
Set the opts%rt_ir%addclouds option.

void **setCldstrSimple** (bool cldstrCimple)
Set the opts%rt_ir%cldstr_simple option.

void **setCldstrThreshold** (double cldstrThreshold)
Set the opts%rt_mw%cldstr_threshold option.

void **setNthreads** (int nthreads)
Set the number of threads RTTOV will use (compile RTTOV with OpenMP to make use of this)

void **setNprofsPerCall** (int nprofsPerCall)
Set the number of profiles passed into rttov_direct or rttov_k per call.

void **setVerboseWrapper** (bool verboseWrapper)
Set the verbose_wrapper option.

void **setCheckOpts** (bool checkOpts)
Set the check_opts option.

void **setStoreRad** (bool storeRad)
Set the store_rad wrapper option.

void **setStoreRad2** (bool storeRad2)
Set the store_rad2 wrapper option.

void **setStoreTrans** (bool storeTrans)
Set the store_trans wrapper option.

Appendix G: Enumeration types

The enumerations are defined in wrapper/rttov_common.h.

The following table lists the constants of the enumeration **rttov::gasUnitType** used to specify the profile gas_units variable in the **setGasUnits** method of the **Profile** class.

Enumeration constants	**Description**
unknown	Default initialisation, ppmv over moist air will be used
ppmv_dry	Gas units of ppmv over dry air
compatibility_mode	RTTOV v11.2 compatibility mode
kg_per_kg	Gas units of kg/kg over moist air
ppmv_wet	Gas units of ppmv over moist air

The following table lists the constants of the enumeration **rttov::itemIdType** used for setting gas, cloud and aerosol profiles in the **setGasItem** method of the **Profiles** class and to obtain the Jacobians for gases, aerosol and cloud profiles using the **getItemK** method of the **Rttov** and **RttovSafe** classes after running the RTTOV K model.

Enumeration constants	**Description**
Q, O3, CO2, N2O, CO, CH4	RTTOV variable gases
CLW	Cloud liquid water (for non-scattering MW simulations)
CFRAC	Cloud fraction for IR cloud scattering simulations
STCO, STMA, CUCC, CUCP, CUMA	The 5 cloud liquid water particle types for IR cloud scattering simulations.
CIRR	The ice cloud particle type for IR cloud scattering simulations.
ICEDE	The ice particle effective diameter input for IR cloud scattering simulations.
INSO, WASO, SOOT, SSAM, SSCM, MINM, MIAM, MICM, MITR, SUSO, VOLA, VAPO, ASDU	The 13 aerosol particle types for IR aerosol scattering simulations.

--END--