XRTM

X Radiative Transfer Model
Version 0.91
March 27, 2012

Greg McGarragh

This manual describes how to install and use The X Radiative Transfer Model (XRTM) version 0.91.

Contents

1 Introduction to XRTM

XRTM (X Radiative Transfer Model) is a plane-parallel multi-layer scalar/vector radiative transfer model with support for absorption, emission, and multiple scattering. In addition to the radiances or Stokes vector elements, XRTM can analytically generate derivatives of these quantities with respect to model inputs either my forward propagation (tangent linear) or backward propagation (adjoint of the tangent linear). XRTM implements several different radiative transfer solvers and includes several features some of which include Delta-M scaling [Wiscombe, 1977], the Nakajima-Tanaka TMS correction [Nakajima and Tanaka, 1988], a pseudo-spherical approximation for solar and line-of-sight beams [Dahlback and Stamnes, 1991], a generalized BRDF formulation [Spurr, 2004], and support to generate results for any number of view angles and/or levels simultaneously.

XRTM's core library is coded in C with a well defined application programming interface (API) and is thread safe so that multiple instances can be called safely in shared memory multi-processor environments. Interfaces are also provided for C++, Fortran 77, and Fortran 90 and as an alternative alternatively XRTM can be executed independently as a stand alone program. Finally, example programs that call XRTM are provided for each language interface.

XRTM includes an extensive test suite that attempts to test the model over the entire range of solvers, features and possible imputes.

For up to date information regarding XRTM, to download the source code distribution, and/or to view the documentation please visit the XRTM web page at

`http://reef.atmos.colostate.edu/~gregm/xrtm/`

For questions or comments or to report a bug email Greg at gregm@atmos.colostate.edu. Bug reports are greatly appreciated! If you would like to report a bug please include sample code that reproduces the bug, along with the inputs and expected outputs.

1.1 License

XRTM is licensed under the GNU General Public License (GPL), Version 3 a copy of which is in the file COPYING in the top level directory of the XRTM source code distribution.

1.2 Conventions used in this manual

Source code such as interface definitions and examples are typeset in `typewriter font`. Source code identifiers such as variable names and function names are also typeset in bold while function

argument types, modifiers, and names, are also typeset in italics. For example a function name will be typeset in bold typewriter font as **`func_name`**, argument types in italic typewriter font as *`int`*, and argument names in bold italic typewriter font as ***`arg_name`***.

Internet links are typeset in the standard color blue. Links that are local to the manual are typeset in a dark red except for citations that link to their corresponding bibliography entries which are in a dark green.

2 Building and Using XRTM

The section discusses, first, the process of building (compiling) XRTM including the core library, the language interfaces, the example programs, and the utility programs. Then the compilation details of using the XRTM core library and the appropriate language interface in your programs is outlined.

2.1 Building XRTM

2.1.1 GNU Make

The standard build system uses GNU Make (other versions of UNIX Make may work but are not tested). This should work on Linux, Unix, Mac OS, and on Windows using the either Cygwin or MinGW (Minimalist GNU for Windows).

The first step is to configure the build for your environment. This includes setting the compiler command and the associated options and setting the appropriate paths to your BLAS and LAPACK libraries. Settings are contained in the file `make.inc` in the XRTM base directory.

Compiler and associated options are contained within the section identified as "Compiler and linker settings". The commands for the compilers to use are represented by the variables `CC`, `CXX`, `F77`, and `F90`, for the C, C++, Fortran 77, and Fortran 90 compilers, respectively, and the associated options are represented by the variables `CCFLAGS`, `CXXFLAGS`, `F77FLAGS`, and `F90FLAGS`. The default settings are appropriate for GCC (GNU Compiler Collection) versions 4.2 and greater and should not have to be modified unless other compilers are being used. Note that XRTM is entirely C89/90 compliant except for the use of complex types. Therefore, the C compiler must be C99 compliant. As an alternative all of XRTM's C code may be built with a C++ compiler in which case the complex support is through the C++ standard library's complex class.

The only external libraries that XRTM currently depends on are BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage). Reference version of both libraries may be obtained from `http://www.netlib.org/` but it is highly recommend, at least for BLAS, that libraries optimized for your platform are used instead. The performance benefits are usually significant. Optimized versions of BLAS include Intel Math Kernel Library (MKL), AMD Core Math Library (ACML), Automatically Tuned Linear Algebra Software (ATLAS), and GotoBLAS.

For each library the appropriate compiler command line additions required to use them are represented by the variables `LIB_BLAS` and `LIB_LAPACK` contained in the section of `make.inc` identified as "BLAS and LAPACK settings". The values of these variables may contain link flags such as `-lblas` and `-llapack` and perhaps flags indicating the location of these libraries such

as `-L/opt/blas` and `-L/opt/lapack`, respectively. The values may also be set to the libraries themselves such as `/opt/blas/libblas.a` or `/opt/lapack/liblapack.a`.

Once the proper settings have been set in `make.inc` XRTM may be compiled by executing the `make` command.

2.1.2 Visual Studio

XRTM may also be built on Windows using Visual Studio along with Intel's Visual Fortran Composer XE for Windows. Supported versions of Visual Studio are 2005, 2008, and 2010. Depending on which version is being used the XRTM Visual Studio solution may be loaded from one of the following solution (`.sln`) files relative to the XRTM base directory:

```
msvs_2005/xrtm.sln
msvs_2008/xrtm.sln
msvs_2010/xrtm.sln
```

2.2 Using XRTM in your code

To use XRTM in your own code either have to include/use the appropriate header/module file and link with the appropriate XRTM library files and BLAS/LAPACK library files.

2.2.1 C

The C interface is part of the core library in the `src/` directory. To use the C interface your code must include the following header file

```
src/xrtm_interface.h
```

and must link with the following libraries:

```
src/libxrtm.a
misc/libxrtm_misc.a
```

or when using Visual Studio the following libraries:

```
$(SolutionDir)/$(ConfigurationName)/libxrtm.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_f.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_misc.lib
```

where the variable `$(SolutionDir)` is `msvs_2005`, `msvs_2008`, or `msvs_2010` and the variable `$(ConfigurationName)` is `Debug` or `Release`.

For example, if one has C code in a file named `my_code.c`, includes the XRTM C interface header file with

```
#include <xrtm_interface.h>
```

and uses gcc to compile and link the code the command may look like this

```
gcc -O2 my_code.c -I$(XRTM_HOME)/src -L$(XRTM_HOME)/src -L$(XRTM_HOME)/misc \
    -lxrtm -lxrtm_misc $(BLAS_STUFF) $(LAPACK_STUFF)
```

where the variable `$(XRTM_HOME)` is the location of the XRTM base directory and the variables `$(BLAS_STUFF)` and `$(LAPACK_STUFF)` represent what is required to link with BLAS and LAPACK, respectively. For more information, take a look at the build details for the C interface example program `examples/example_c.c`.

2.2.2 C++

To use the C++ interface your code must include the following header file

```
interfaces/xrtm_int_cpp.h
```

and must link with the following libraries:

```
src/libxrtm.a
misc/libxrtm_misc.a
interfaces/libxrtm_interfaces.a
```

or when using Visual Studio the following libraries:

```
$(SolutionDir)/$(ConfigurationName)/libxrtm.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_f.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_misc.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_interfaces.lib
```

where the variable `$(SolutionDir)` is `msvs_2005`, `msvs_2008`, or `msvs_2010` and the variable `$(ConfigurationName)` is `Debug` or `Release`. For more information, take a look at the build details for the C++ interface example program `examples/example_cpp.cpp`.

2.2.3 Fortran 77

To use the Fortran 77 interface your code must include the following file

```
interfaces/xrtm_int_f77.inc
```

and must link with the following libraries:

```
src/libxrtm.a
misc/libxrtm_misc.a
interfaces/libxrtm_interfaces.a
```

or when using Visual Studio the following libraries:

```
$(SolutionDir)/$(ConfigurationName)/libxrtm.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_f.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_misc.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_interfaces.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_interfaces_f.lib
```

where the variable `$(SolutionDir)` is `msvs_2005`, `msvs_2008`, or `msvs_2010` and the variable `$(ConfigurationName)` is `Debug` or `Release`. For more information, take a look at the build details for the Fortran 77 interface example program `examples/example_f77.f`.

2.2.4 Fortran 90

To use the Fortran 90 interface your code must `USE` the `XRTM_INT_F90` module and must link with the following libraries:

```
src/libxrtm.a
misc/libxrtm_misc.a
interfaces/libxrtm_interfaces.a
```

or when using Visual Studio the following libraries:

```
$(SolutionDir)/$(ConfigurationName)/libxrtm.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_f.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_misc.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_interfaces.lib
$(SolutionDir)/$(ConfigurationName)/libxrtm_interfaces_f.lib
```

where the variable `$(SolutionDir)` is `msvs_2005`, `msvs_2008`, or `msvs_2010` and the variable `$(ConfigurationName)` is `Debug` or `Release`. For more information, take a look at the build details for the Fortran 90 interface example program `examples/example_f90.f90`.

3 XRTM C Interface

XRTM is unlike other common RT models in that it exists as an object/instance that is created, modified, and destroyed. It maintains a valid state between simulations where only inputs that change need to be updated for subsequent simulations. This design allows input overhead to be minimized and allows redundant calculations to be saved in a transparent manor. Each XRTM instance is contained within an isolated memory scope so that multiple instances may be used in a shared memory multiprocessing environment.

The interface is made up of input configuration constants (section 3.2), functions for creating and destroying an XRTM instance (section 3.3), functions for setting and getting inputs (section 3.4), and functions that run the appropriate calculations and return outputs (section 3.5). Typical use of XRTM would be to create an instance with **xrtm_create()**, set inputs with the **xrtm_set_*()** functions, run the model and get outputs with the **xrtm_calc_*()** functions, loop over the last two steps until the model is no longer needed, and then finally, destroy the instance with **xrtm_destroy()**.

3.1 (Arrays of (arrays of)) ... arrays

Several of the XRTM input and output functions take multi-dimensional arrays as arguments. In C89/90 if true multi-dimensional arrays are passed to functions all but the first dimension must be static. This is an obvious limitation for the XRTM C interface as these sizes are in fact dynamic. So instead, XRTM uses "(arrays of pointers to (arrays of pointers to)) ... 1-d arrays" or as they are referred to in this manual "(arrays of (arrays of)) ... 1-d arrays". These structures may be allocated by the user but as a convenience XRTM provides functions to do this. These functions are efficient in that internally they allocate memory with one call for all the arrays making up the entire structure and then set the pointers to the appropriate locations in memory. This method also has the advantage that the data lies contiguously in memory. So for example, with a 2 dimensional array the rows would lie one after a another where as if the rows were allocated separately they might not lie contiguously in memory.

The following is a formal description of the functions for allocating "(arrays of (arrays of)) ... 1-d arrays" and their corresponding deallocation functions:

`<type name> *...*alloc_array<# of dimensions>_<type id>(int size_1,...,int size_n)`

Description:

Allocate an "(array of (arrays of)) ... 1-d arrays". **<# of dimensions>** is either 1, 2, 3, etc. and **<type>** is `i` or `d` for `int` or `double` arrays, respectively. The arguments are the sizes of

each dimension of the array (***size_1, ... size_n***) where ***n*** is the # of dimensions. The return value is a "(pointer to (a pointer to)) ... a pointer" to <type name> with a pointer depth equal to the # of dimensions.

Arguments:

size_1 size of the first dimension

size_n size of the last (***n***th) dimension

Return value:

A "(pointer to (a pointer to)) ... a pointer" to <type name> represting the allocated "(array of (arrays of)) ... 1-d arrays" or NULL on error.

`void` **free_array<# of dimensions>_<type id>**(*<type name> *...*array*)

Description:

Deallocate (free) an "(array of (arrays of)) ... 1-d arrays". **<# of dimensions>** is either 1, 2, 3, etc. and **<type>** is `i` or `d` for `int` or `double` arrays, respectively. The argument is the "(pointer to (a pointer to)) ... a pointer" to <type name> represting the "(array of (arrays of)) ... 1-d arrays" returned by **alloc_array<# of dimensions>_<type id>**.

Arguments:

array "(pointer to (a pointer to)) ... a pointer" to <type name> represting the "(array of (arrays of)) ... 1-d arrays"

Return value:

None.

So for example, if a 2 dimensional array of arrays of type `double` is to be allocated then the function to call would be

```
double **alloc_array2_d(int size_1, int size_2)
```

and the corresponding deallocation routine would be

```
void free_array2_d(double **array)
```

3.2 Configuration constants

3.2.1 Options

Options are turned on by setting the appropriate bit of a 32 bit wide mask which is the ***options*** argument to **xrtm_create()**. The appropriate bits may be set using masks (declared as enumeration constants) associated with each option. For example, Delta-M scaling and the pseudo spherical approximation may be turned on by using the bitwise inclusive OR operator with something like

```
options = XRTM_OPTION_DELTA_M | XRTM_OPTION_PSA.
```

XRTM_OPTION_CALC_DERIVS

Calculate derivatives with respect to optical property inputs. Requires ***n_derivs*** to be greater than or equal to one.

XRTM_OPTION_DELTA_M
Use Delta-M scaling [Wiscombe, 1977].

XRTM_OPTION_N_T_TMS
Use the Nakajima and Tanaka TMS correction [Nakajima and Tanaka, 1988].

XRTM_OPTION_FOUR_CONV_OLD
Used for testing purposes only.

XRTM_OPTION_FOUR_CONV_NEW
Used for testing purposes only.

XRTM_OPTION_NO_AZIMUTHAL
Include only the first Fourier term of the expansion in azimuth, i.e. the azimuthal average.

XRTM_OPTION_OUTPUT_AT_LEVELS
Output at user specified levels. This is in contrast to output at user specified optical depths. Some solvers support output at TOA only, others support output at TOA and/or BOA, while some support output at any level. Check the solver descriptions for which solver supports what. Requires at least one call to **xrtm_set_out_levels()** once the model is created.

XRTM_OPTION_OUTPUT_AT_TAUS
Output at user specified optical depths from TOA. This is in contrast to output at user specified levels. Some solvers support output at TOA only, others support output at TOA and/or BOA, other support output at any level, while some support output at any optical depth (within layers). Check the solver descriptions for which solver supports what. Requires at least one call to **xrtm_set_out_taus()** once the model is created.

XRTM_OPTION_PHASE_SCALAR
Used for testing purposes only.

XRTM_OPTION_PHASE_MATRIX_GC
Used for testing purposes only.

XRTM_OPTION_PHASE_MATRIX_LC
Used for testing purposes only.

XRTM_OPTION_PSA
Use the so called pseudo-spherical approximation to model the solar beam through a spherical spherical shell atmosphere [Dahlback and Stamnes, 1991].

XRTM_OPTION_QUAD_NORM_GAUS_LEG
Use (standard) Gauss-Legendre quadrature.

XRTM_OPTION_QUAD_DOUB_GAUS_LEG
Use double Gauss-Legendre quadrature.

XRTM_OPTION_QUAD_LOBATTO
Use Lobatto quadrature.

XRTM_OPTION_SAVE_PHASE_MATS
Save phase matrices between XRTM calls.

XRTM_OPTION_SAVE_LOCAL_R_T
Save local **r** and **t** matrices between XRTM calls.

XRTM_OPTION_SAVE_LAYER_R_T_S

XRTM_OPTION_SAVE_TOTAL_R_T_S

XRTM_OPTION_SFI
For solvers that support it use source function integration for output at arbitrary zenith angles otherwise quadrature dummy nodes are used.

XRTM_OPTION_SOURCE_SOLAR
Include solar sources.

XRTM_OPTION_SOURCE_THERMAL
Include thermal sources.

XRTM_OPTION_STACK_REUSE_ADDING

XRTM_OPTION_TOP_DOWN_ADDING
For solvers that use adding add from the top down to get output at the surface only. This is in contrast to full adding and can significantly improve run time.

XRTM_OPTION_BOTTOM_UP_ADDING
For solvers that use adding add from the bottom up to get output at TOA only. This is in contrast to full adding and can significantly improve run time.

XRTM_OPTION_UPWELLING_OUTPUT
Output upwelling values.

XRTM_OPTION_DOWNWELLING_OUTPUT
Output downwelling values.

XRTM_OPTION_VECTOR
Run the model in vector mode.

3.2.2 Solvers

XRTM may be created to use any number of solvers. Limiting the initialization to only the solvers required will in some cases lead to significant memory savings. Solvers are turned on by setting the appropriate bit of a 32 bit wide mask which is the ***solvers*** argument to **xrtm_create()**. The appropriate bits may be set using masks (declared as enumeration constants) associated with each solver. For example, XRTM may be create to use the Eigenmatrix/BVP solver along with the single and second order scattering solvers by using the bitwise inclusive OR operator with something like

```
solvers = XRTM_SOLVER_EIG_BVP | XRTM_SOLVER_SINGLE | XRTM_SOLVER_TWO_OS.
```

XRTM_SOLVER_DOUB_ADD
Doubling/Adding: Use doubling to get global reflection and transmission matrices for each layer. Then use adding to get global reflection and transmission matrices for the entire atmosphere [Grant and Hunt, 1969, de Haan et al., 1987, Liou, 2002].

XRTM_SOLVER_EIG_ADD
Eigenmatrix/Adding: Use the Eigenvalue problem to get global reflection and transmission matrices for each layer. Then use adding to get global reflection and transmission matrices for the entire atmosphere [Aronson, 1972, Nakajima and Tanaka, 1986, Voronovich et al., 2004, Spurr and Christi, 2007].

XRTM_SOLVER_EIG_BVP
Eigenmatrix/BVP (a.k.a. the Discrete Ordinate Method): Use the Eigenvalue problem to obtain the layer homogeneous solution. Then solve a boundary value problem for the entire atmosphere [Liou, 1973, Stamnes et al., 1988, Siewert, 2000, Spurr et al., 2001].

XRTM_SOLVER_MEM_BVP
Matrix exponential by eigenmatrix / BVP: A variant of the Discrete Ordinate Method with a matrix exponential formulation [Doicu and Trautmann, 2009a,b].

XRTM_SOLVER_PADE_ADD
Matrix exponential by Padé approximation / Adding (A.K.A. PARTM): Use the Padé approximation to the matrix exponential to get global reflection and transmission matrices for each layer. Then use adding to get global reflection and transmission matrices for the entire atmosphere [McGarragh and Gabriel, 2010].

XRTM_SOLVER_SINGLE
Includes only first order scattering from the atmosphere and surface.

XRTM_SOLVER_SOS
Successive orders of scattering using an approximate integration in optical thickness [Fymat and Ueno, 1974, Min and Duan, 2004, Lenoble et al., 2007].

XRTM_SOLVER_TWO_OS
Second order scattering with the typical numerical integration over zenith and azimuth but with an analytical integration in optical thickness. [Kawabata and Ueno, 1988, Natraj and Spurr, 2007]

3.2.3 BRDF kernels

[Spurr, 2004]

XRTM_KERNEL_LAMBERTIAN

XRTM_KERNEL_ROUJEAN
[Roujean et al., 1992]

XRTM_KERNEL_LI_SPARSE
[Wanner et al., 1995]

XRTM_KERNEL_LI_DENSE
[Wanner et al., 1995]

XRTM_KERNEL_ROSS_THIN
[Wanner et al., 1995]

XRTM_KERNEL_ROSS_THICK
[Wanner et al., 1995]

XRTM_KERNEL_HAPKE
[Hapke, 1981, Hapke and Wells, 1981]

XRTM_KERNEL_RAHMAN
[Rahman et al., 1993]

XRTM_KERNEL_COX_MUNK
[Cox and Munk, 1954]

3.2.4 Solutions

XRTM_OUTPUT_RADIANCE

XRTM_OUTPUT_RADIANCE_MEAN

XRTM_OUTPUT_FLUX

XRTM_OUTPUT_FLUX_DIVERGENCE

3.3 Initiating and destroying XRTM

int **xrtm_create**(*xrtm_data* ***d**, *int* ***options**, *int* ***solvers**, *int* ***max_coef**, *int* ***n_quad**, *int* ***n_stokes**, *int* ***n_derivs**, *int* ***n_layers**, *int* ***n_kernels**, *int* ***n_kernel_quad**, *int* ***kernels**, *int* ***n_out_levels**, *int* ***n_out_thetas**)

Description:
Create a new XRTM instance. When finished with the instance created, **xrtm_destroy()** must be called to free memory allocated by **xrtm_create()**.

Arguments:

d	the *xrtm_data* structure which will represent the instance created
options	bit mask of XRTM configuration `options`
solvers	bit mask of XRTM `solvers` that will be used
max_coef	maximum number of phase function Legendre expansion coefficients that will be used
n_quad	number of quadrature points in one hemisphere
n_stokes	size of the stokes vector to calculate (set to one for scalar mode)
n_derivs	number of derivatives to calculate
n_layers	number of plane parallel layers in the atmosphere
n_kernels	number of BRDF kernels to use for the BRDF
n_kernel_quad	number of quadrature points to use for BRDF integration
kernels	array of BRDF kernels to use (of length ***n_kernels***)
n_out_levels	number of user defined output levels
n_out_thetas	number of user defined output zenith angles

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`void` **xrtm_destroy**(*xrtm_data* ***d**)

Description:
Destroy an XRTM instance which includes freeing all memory allocated by **xrtm_create()**.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
none

3.4 Setting and getting inputs

`int` **xrtm_get_options**(*xrtm_data* ***d**)

Description:
Get the bit mask of XRTM options with which this instance was created.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The bit mask of XRTM options with which this instance was created or XRTM_INT_ERROR on error.

`int` **xrtm_get_solvers**(*xrtm_data* ***d**)

Description:
Get the bit mask of XRTM solvers for which this instance has been created to use.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The bit mask of XRTM solvers for which this instance has been created to use or XRTM_INT_ERROR on error.

`int xrtm_get_max_coef(xrtm_data *d)`

Description:
Get the maximum number of phase function Legendre expansion coefficients for which this XRTM instance has been created to handle.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The maximum number of phase function Legendre expansion coefficients for which this XRTM instance has been created to handle or XRTM_INT_ERROR on error.

`int xrtm_get_n_quad(xrtm_data *d)`

Description:
Get the number of quadrature points in one hemisphere used by this XRTM instance.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The number of quadrature points in one hemisphere used by this XRTM instance or XRTM_INT_ERROR on error.

`int xrtm_get_n_stokes(xrtm_data *d)`

Description:
Get the size of the stokes vector for which this instance has been created to calculate.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The size of the stokes vector for which this instance has been created to calculate or XRTM_INT_ERROR on error.

`int xrtm_get_n_derivs(xrtm_data *d)`

Description:
Get the number of derivatives for which this instance has been created to calculate.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The number of derivatives for which this instance has been created to calculate or XRTM_INT_ERROR on error.

`int` **`xrtm_get_n_layers`**`(xrtm_data *d)`

Description:
Get the number of plane parallel layers in the atmosphere modeled by this instance.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The number of plane parallel layers in the atmosphere modeled by this instance or XRTM_INT_ERROR on error.

`int` **`xrtm_get_n_kernels`**`(xrtm_data *d)`

Description:
Get the number of BRDF kernels for which this instance has been created to use.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The number of BRDF kernels for which this instance has been created to use or XRTM_INT_ERROR on error.

`int` **`xrtm_get_n_kernel_quad`**`(xrtm_data *d)`

Description:
Get the number of quadrature points for BRDF integration used by this instance.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The number of quadrature points for BRDF integration used by this instance or XRTM_INT_ERROR on error.

`int` **`xrtm_get_kernel`**`(xrtm_data *d, int i_kernel)`

Description:
Get the kernel identifier for a given kernel index. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **`xrtm_create()`**.

Arguments:
d the *xrtm_data* structure which represents the instance created
i_kernel the kernel index, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$

Return value:
The kernel identifier for index ***i_kernel*** or XRTM_INT_ERROR on error.

`int` **`xrtm_get_n_out_levels`**`(xrtm_data *d)`

Description:

Get the number of levels at which this instance has been created to output.

Arguments:

d the ***xrtm_data*** structure which represents the instance created

Return value:

The number of levels at which this instance has been created to output or XRTM_INT_ERROR on error.

`int` **`xrtm_get_n_out_thetas`**`(`*`xrtm_data`* ***`*d`***`)`

Description:

Get the number of zenith angles at which this instance has been created to output.

Arguments:

d the ***xrtm_data*** structure which represents the instance created

Return value:

The number of zenith angles at which this instance has been created to output or XRTM_INT_ERROR on error.

`int` **`xrtm_set_doub_d_tau`**`(`*`xrtm_data`* ***`*d`***`,` *`double`* ***`d_tau`***`)`

Description:

Set the initial layer optical thickness $\Delta\tau$ for the doubling method. To set this value XRTM must have been created to use at least one of the following solvers: XRTM_SOLVER_DOUB_ADD, otherwise it is an error.

Arguments:

d the ***xrtm_data*** structure which represents the instance created

d_tau the initial layer optical thickness $\Delta\tau$, where ***d_tau*** > 0.0

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_doub_d_tau`**`(`*`xrtm_data`* ***`*d`***`)`

Description:

Get the initial layer optical thickness $\Delta\tau$ for the doubling method. To get this value XRTM must have been created to use at least one of the following solvers: XRTM_SOLVER_DOUB_ADD, otherwise it is an error.

Arguments:

d the ***xrtm_data*** structure which represents the instance created

Return value:

The initial layer optical thickness $\Delta\tau$ for the doubling method or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_pade_params`**`(`*`xrtm_data`* ***`*d`***`,` *`int`* ***`pade_s`***`,` *`int`* ***`pade_r`***`)`

Description:

Set the Padé scaling power of two (number of doublings) s and the degree of Padé approximate r. If either value is set to a value that is out of range then s and r are chosen automatically from a lookup table based on layer optical thickness τ and the maximum output zenith angle θ. To set these values, XRTM must have been created to use at least one of the following solvers: XRTM_SOLVER_PADE_ADD, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
pade_s	Padé scaling power of two s, where ***pade_s*** ≥ 0
pade_r	degree of Padé approximate r, where ***pade_r*** > 0

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_get_pade_params`**`(`*`xrtm_data *d, int *pade_s, int *pade_r`*`)`

Description:

Get the Padé scaling power of two (number of doublings) s and the degree of Padé approximate r. To get these values, XRTM must have been created to use at least one of the following solvers: XRTM_SOLVER_PADE_ADD, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
pade_s	(output) Padé scaling power of two s
pade_r	(output) degree of Padé approximate r

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_set_sos_params`**`(`*`xrtm_data *d, int max_os, double max_tau, double sos_tol`*`)`

Description:

Set parameters related to successive order of scattering. They are the maximum order of scattering that will be computed, the maximum layer optical thickness used (all layers of a larger optical thickness are divided evenly into enough sub-layers so that each sub-layer has an optical thickness less than or equal to the maximum allowable value), and the successive order of scattering tolerance limit. The tolerance limit is the minimum radiance contribution from any single quadrature angle with which the succession will continue to the next order of scattering. If this limit is not met the succession will terminate. To set these values, XRTM must have been created to use at least one of the following solvers: XRTM_SOLVER_SOS, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
max_os	maximum order of scattering, where ***max_os*** ≥ 0
max_tau	maximum layer optical thickness used, where ***max_tau*** > 0.0
sos_tol	successive order of scattering tolerance limit, where ***sos_tol*** ≥ 0.0

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int xrtm_get_sos_params(xrtm_data *d, int *max_os, double *max_tau, double *sos_tol)`

Description:
Get parameters related to successive order of scattering. To get these values, XRTM must have been created to use at least one of the following solvers: XRTM_SOLVER_SOS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
max_os	(output) maximum order of scattering
max_tau	(output) maximum layer optical thickness used
sos_tol	(output) successive order of scattering tolerance limit

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`int xrtm_set_fourier_tol(xrtm_data *d, double fourier_tol)`

Description:
Set the tolerance limit for the Fourier expansion in azimuth angle. The tolerance limit is the minimum intensity contribution from any single output level and output angle with which the summation will continue to the next term. If this limit is not met for *all* of the output levels and output angles the summation will terminate. If a single scattering correction is to be applied (XRTM_OPTION_N_T_TMS) then the series starts with the full (untruncated) single scattering contribution while each term includes only the truncated multiple scattering contribution. If it is set to zero then the summation will include all terms ($2n$).

Arguments:

d	the *xrtm_data* structure which represents the instance created
fourier_tol	tolerance limit for the Fourier expansion in azimuth angle, where ***fourier_tol*** ≥ 0.0

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double xrtm_get_fourier_tol(xrtm_data *d)`

Description:
Get the tolerance limit for the Fourier expansion in azimuth angle.

Arguments:

d	the *xrtm_data* structure which represents the instance created

Return value:
The tolerance limit for the Fourier expansion in azimuth angle or XRTM_DBL_ERROR on error.

`int xrtm_set_F_0(xrtm_data *d, double F_0)`

Description:
Set the intensity of the incident parallel beam at TOA F_0. Setting F_0 to zero turns off the solar source. If a thermal source is used (XRTM_OPTION_SOURCE_THERMAL) then the units for F_0 and the TOA, BOA, and level Planck radiances must be the same. Otherwise the units for F_0 are arbitrary.

Arguments:
d the *xrtm_data* structure which represents the instance created
F_0 intensity of the incident parallel beam at TOA F_0, where ***F_0*** ≥ 0.0

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_F_0`**`(`*`xrtm_data *d`*`)`

Description:
Get the intensity of the incident parallel beam at TOA F_0.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The intensity of the incident parallel beam at TOA F_0 or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_theta_0`**`(`*`xrtm_data *d, double theta_0`*`)`

Description:
Set the zenith angle for the incident parallel beam at TOA (the solar zenith angle) θ_0.

Arguments:
d the *xrtm_data* structure which represents the instance created
theta_0 zenith angle for the incident parallel beam at TOA θ_0, where $0.0 \leq$ ***theta_0*** < 90.0

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_theta_0`**`(`*`xrtm_data *d`*`)`

Description:
Get the zenith angle for the incident parallel beam at TOA (the solar zenith angle) θ_0.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The zenith angle for incident parallel beam at TOA θ_0 or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_phi_0`**`(`*`xrtm_data *d, double phi_0`*`)`

Description:
Set the azimuth angle for the incident parallel beam at TOA (the solar azimuth angle) ϕ_0.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
phi_0	azimuth angle for the incident parallel beam at TOA ϕ_0, where $0.0 \leq$ ***phi_0*** < 360.0

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_phi_0`**`(`*`xrtm_data *d`*`)`

Description:

Get the azimuth angle for the incident parallel beam at TOA (the solar azimuth angle) ϕ_0.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created

Return value:

The azimuth angle for incident parallel beam at TOA ϕ_0 or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_out_levels`**`(`*`xrtm_data *d, int *out_levels`*`)`

Description:

Set the levels at which to output results given an array of length ***n_out_levels***. Levels are defined at layer boundaries. For example a 3 layer atmosphere would have 4 levels and TOA and BOA (the surface) would be levels 0 and 3, respectively. Levels must be specified in ascending order. To set this value XRTM must have been created with the option XRTM_OPTION_OUTPUT_AT_LEVELS, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
out_levels	array of output level indices in ascending order, where $0 \leq$ ***out_levels***`[i]` $\leq$ ***n_layers*** and $0 \leq$ `i` $\leq$ ***n_out_levels*** $- 1$

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_get_out_levels`**`(`*`xrtm_data *d, int *out_levels`*`)`

Description:

Get the levels at which results are output at as an array of length ***n_out_levels***. To get this value XRTM must have been created with the option XRTM_OPTION_OUTPUT_AT_LEVELS, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
out_levels	(output) array of output level indices in ascending order of length ***n_out_levels***

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_set_out_taus`**`(`*`xrtm_data *d, double *out_taus`*`)`

Description:
Set the optical depths at which to output results given an array of length ***n_out_levels***. Optical depths must be specified in ascending order. To set this value XRTM must have been created with the option XRTM_OPTION_OUTPUT_AT_TAUS, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
out_taus	array of output optical depths in ascending order, where $0.0 \leq$ ***out_taus***`[i]` $\leq \tau_s$ and $0 \leq$ `i` $\leq$ ***n_out_levels*** $- 1$ and τ_s is the optical depth to the surface

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_get_out_taus`**`(`*`xrtm_data *`***`d`**`, double *`***`out_taus`**`)`

Description:
Get the optical depths at which results are output at as an array of length ***n_out_levels***. To get this value XRTM must have been created with the option XRTM_OPTION_OUTPUT_AT_TAUS, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
out_taus	(output) array of output optical depths in ascending order of length ***n_out_levels***

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_set_out_thetas`**`(`*`xrtm_data *`***`d`**`, double *`***`out_thetas`**`)`

Description:
Set the zenith angles θ at which to output results given an array of length ***n_out_thetas***. To set this value XRTM must have been created with ***n_out_thetas*** > 0, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
out_thetas	array of output zenith angles θ, where $0.0 \leq$ ***out_thetas***`[i]` < 90.0 and $0 \leq$ `i` $\leq$ ***n_out_thetas*** $- 1$

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_get_out_thetas`**`(`*`xrtm_data *`***`d`**`, double *`***`out_thetas`**`)`

Description:
Get the zenith angles θ at which results are output at as an array of length ***n_out_thetas***. To set this value XRTM must have been created with ***n_out_thetas*** > 0, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
out_thetas	(output) array of output zenith angles θ of length ***n_out_thetas***

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_set_top_b`**`(xrtm_data *`***`d`***`, double` ***`top_b`***`)`

Description:
Set the intensity of the downward isotopic radiation at TOA. Must be in the same units as F_0 and the BOA and level Planck radiances. To set this value XRTM must have been created with the option XRTM_OPTION_SOURCE_THERMAL, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
top_b	intensity of the downward isotopic radiation at TOA, where ***top_b*** ≥ 0.0

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_top_b`**`(xrtm_data *`***`d`***`)`

Description:
Get the intensity of the downward isotopic radiation at TOA. To get this value XRTM must have been created with the option XRTM_OPTION_SOURCE_THERMAL, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created

Return value:
The intensity of the downward isotopic radiation at TOA or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_planet_r`**`(xrtm_data *`***`d`***`, double` ***`planet_r`***`)`

Description:
Set the planetary radius for the point located at BOA. The units for this value and the level heights must be the same. To set this value XRTM must have been created with the option XRTM_OPTION_PSA, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
planet_r	planetary radius, where ***planet_r*** ≥ 0.0

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_planet_r`**`(xrtm_data *`***`d`***`)`

Description:
Get the planetary radius for the point located at BOA. To get this value XRTM must have been created with the option XRTM_OPTION_PSA, otherwise it is an error.

Arguments:

d the ***xrtm_data*** structure which represents the instance created

Return value:

The planetary radius for the point located at BOA or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_levels_z`**`(xrtm_data *`**`d`**`, double *`**`levels_z`**`)`

Description:

Set the level heights z as an array of length ***n_layers*** $+ 1$. The units for this value and the planetary radius must be the same. To set this value XRTM must have been created with the option XRTM_OPTION_PSA, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
levels_z	array of level heights z, where ***levels_z***`[i]` ≥ 0.0 and $0 \leq$ `i` $\leq$ ***n_layers***

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_get_levels_z`**`(xrtm_data *`**`d`**`, double *`**`levels_z`**`)`

Description:

Get the level heights z as an array of length ***n_layers*** $+ 1$. To get this value XRTM must have been created with the option XRTM_OPTION_PSA, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
levels_z	(output) array of level heights z of length ***n_layers*** $+ 1$

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_set_levels_b`**`(xrtm_data *`**`d`**`, double *`**`levels_b`**`)`

Description:

Set the level Planck radiances B as an array of length ***n_layers*** $+ 1$. Must be in the same units as F_0 and the TOA and BOA Planck radiances. To set this value XRTM must have been created with the option XRTM_OPTION_SOURCE_THERMAL, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
levels_b	array of level Planck radiances B, where ***levels_b***`[i]` ≥ 0.0 and $0 \leq$ `i` $\leq$ ***n_layers***

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_get_levels_b`**`(xrtm_data *`**`d`**`, double *`**`levels_b`**`)`

Description:

Get the level Planck radiances B as an array of length ***n_layers*** + 1. To get this value XRTM must have been created with the option XRTM_OPTION_SOURCE_THERMAL, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
levels_b	(output) array of level Planck radiances B of length ***n_layers*** + 1

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

```
int xrtm_set_coef_1(xrtm_data *d, int i_layer, int n_coef, double **coef)
int xrtm_set_coef_n(xrtm_data *d, int *n_coef, double ***coef)
```

Description:

Set layer phase matrix expansion coefficients $B_{ij,l}$, where $i, j = 1$ for scalar transport (***n_stokes*** $= 1$) and $1 \leq i, j \leq 4$ for vector transport (when option XRTM_OPTION_VECTOR has been specified and ***n_stokes*** > 1). In the case of scalar transport a single element $B_{0,0,l}$ is given to XRTM which are the Legendre expansion coefficients β_l in the equation for the phase function given by

$$P(\cos\Theta) = \sum_{l=0}^{N} \beta_l P_l(\cos\Theta),$$

where Θ is single scattering angle, P_l are Legendre polynomials of degree l, and N is the maximum degree term in the expansion. In the case of vector transport 6 elements are given to XRTM, the so "Greek constants" [Siewert, 1982] of the matrix

$$\mathbf{B}_l = \begin{bmatrix} \beta_l & \gamma_l & 0 & 0 \\ \gamma_l & \alpha_l & 0 & 0 \\ 0 & 0 & \zeta_l & -\varepsilon_l \\ 0 & 0 & \varepsilon_l & \delta_l \end{bmatrix},$$

which are the coefficients for expansion in terms of generalized spherical functions of the phase matrix

$$\mathbf{F}(\Theta) = \begin{bmatrix} a_1(\Theta) & b_1(\Theta) & 0 & 0 \\ b_1(\Theta) & a_2(\Theta) & 0 & 0 \\ 0 & 0 & a_3(\Theta) & b_2(\Theta) \\ 0 & 0 & -b_2(\Theta) & a_4(\Theta) \end{bmatrix},$$

where

$$\begin{aligned} a_1(\Theta) &= \sum_{l=0}^{N} \beta_l P^l_{0,0}(\cos\Theta), \\ a_2(\Theta) + a_3(\Theta) &= \sum_{l=2}^{N} (\alpha_l + \zeta_l) P^l_{2,2}(\cos\Theta), \\ a_3(\Theta) - a_3(\Theta) &= \sum_{l=2}^{N} (\alpha_l - \zeta_l) P^l_{2,-2}(\cos\Theta), \end{aligned}$$

$$
\begin{aligned}
a_4(\Theta) &= \sum_{l=0}^{N} \delta_l P^l_{0,0}(\cos\Theta), \\
b_1(\Theta) &= \sum_{l=0}^{N} \gamma_l P^l_{0,2}(\cos\Theta), \\
b_2(\Theta) &= \sum_{l=0}^{N} \varepsilon_l P^l_{0,2}(\cos\Theta).
\end{aligned}
$$

The "Greek constants" are given to XRTM as a $(6 \times \textbf{\textit{n\_coef}})$ array of arrays according to the mapping

$$
\begin{aligned}
\textbf{\textit{coef}}\texttt{[0][l]} &= \beta_l \\
\textbf{\textit{coef}}\texttt{[1][l]} &= \alpha_l \\
\textbf{\textit{coef}}\texttt{[2][l]} &= \zeta_l \\
\textbf{\textit{coef}}\texttt{[3][l]} &= \delta_l \\
\textbf{\textit{coef}}\texttt{[4][l]} &= -\gamma_l \\
\textbf{\textit{coef}}\texttt{[5][l]} &= -\varepsilon_l,
\end{aligned}
$$

where as explanined below, ***n_coef*** is the number of coefficients, ***coef*** is the input array, and $0 \leq \texttt{l} \leq \textbf{\textit{n\_coef}}$.

Any number of phase matrix coeffieicents may be supplied by the user for each layer with the restriction that the number must be less than or equal to ***max_coef*** which is set when an XRTM instance is created. (It is up to the user to set ***max_coef*** to the maximum number of coefficients to be supplied for all layers.) It is important that the user is aware of the optimal choice in different cases. With XRTM_OPTION_DELTA_M turned off $2 * \textbf{\textit{n\_quad}} - 1$ coeffieicents will be used so if your phase function has less than or equal to as many coeffieicents then supply them all, otherwise you only need to supply a maximum of $2 * \textbf{\textit{n\_quad}} - 1$. If XRTM_OPTION_DELTA_M is turned on then an additional coefficient must be supplied for a total of $2 * \textbf{\textit{n\_quad}}$. If your phase function has less than as many coeffieicents then it does not need to be delta-M scaled and will not be affected by delta-M scaling. If XRTM_OPTION_N_T_TMS is turned on (which requires XRTM_OPTION_DELTA_M to be turned on) then the all the coefficients required to represent the *full* phase matrix should be supplied.

A choice of two functions are provided for this purpose of setting these elements. **xrtm_set_coef_1()** sets the values for a given layer index ***i_layer*** given an $(\textbf{\textit{n\_elem}} \times \textbf{\textit{n\_coef}})$ array of arrays, where ***n_coef*** is the number of coefficients given for the layer. **xrtm_set_coef_n()** sets the values for all layers given an $(\textbf{\textit{n\_layers}} \times \textbf{\textit{n\_elem}} \times \textbf{\textit{n\_coef}}\texttt{[i]}, 0 \leq \texttt{i} \leq \textbf{\textit{n\_layers}} - 1)$ array of arrays of arrays where ***n_coef*** is an array of length ***n_layers*** giving the number of coefficients for each layer. In both cases $\textbf{\textit{n\_elem}} = 1$ for scalar mode and $\textbf{\textit{n\_elem}} = 6$ when option XRTM_OPTION_VECTOR has been specified.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_layer	index of layer to set, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
n_coef	number of coefficients given as a scalar (_**1**) or an array (_**n**) of length ***n_layers***, depending on the function called, where $0 \leq$ ***n_coef***, ***n_coef***`[i]` $\leq$ ***max_coef***
coef	phase matrix expansion coefficients β_{ij} as an array of arrays (_**1**) or an array of arrays of arrays (_**1**), depending on the function called, where $-\infty \leq$ ***coef***`[i][j]`, ***coef***`[i][j][k]` $\leq \infty$, except that ***coef***`[0][0]`, ***coef***`[i][0][0]` $= 1$

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`int` **xrtm_get_n_coef**(*xrtm_data* ***d**, *int* ***i_layer***)

Description:
Get the number of phase matrix expansion coefficients for layer index ***i_layer***.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_layer	index of layer to get, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$

Return value:
The number of phase matrix expansion coefficients for layer index ***i_layer*** or XRTM_INT_ERROR on error.

`double` **xrtm_get_coef**(*xrtm_data* ***d**, *int* ***i_layer**, *int* ***i_elem**, *int* ***i_coef***)

Description:
Get the phase matrix expansion coefficient for layer index ***i_layer***, matrix element index ***i_elem***, and coefficient index ***i_coef***.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_layer	index of layer to get, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
i_elem	index of phase matrix element to get, where $0 \leq$ ***i_elem*** $\leq$ ***n_elem*** and ***n_elem*** $= 0$ for scalar mode and ***n_elem*** $= 6$ when option XRTM_OPTION_VECTOR has been specified
i_coef	index of phase matrix coefficient to get, where $0 \leq$ ***i_coef*** $\leq$ ***n_coef***`[i]` $- 1$ and $0 \leq$ `i` $\leq$ ***n_layers*** $- 1$

Return value:
The phase matrix expansion coefficient (***i_elem***, ***i_coef***) for layer index ***i_layer*** or XRTM_DBL_ERROR on error.

`int` **xrtm_set_coef_l_11**(*xrtm_data* ***d**, *int* ***i_layer**, *int* ***i_deriv**, *double* ****coef_l***)
`int` **xrtm_set_coef_l_n1**(*xrtm_data* ***d**, *int* ***i_deriv**, *double* *****coef_l***)
`int` **xrtm_set_coef_l_1n**(*xrtm_data* ***d**, *int* ***i_layer**, *double* *****coef_l***)
`int` **xrtm_set_coef_l_nn**(*xrtm_data* ***d**, *double* ******coef_l***)

Description:

Set layer linearized phase matrix expansion coefficients $\partial\beta_{ij}/\partial x$. A choice of four functions are provided for this purpose. **xrtm_set_coef_l_11()** sets the values for a given layer index ***i_layer*** and a given derivative index ***i_deriv*** given an (***n_elem*** × ***n_coef***) array of arrays. **xrtm_set_coef_l_n1()** sets the values for all layers for a given derivative index ***i_deriv*** given an (***n_layers*** × ***n_elem*** × ***n_coef***[i], $0 \leq$ i $\leq$ ***n_layers*** $- 1$) array of arrays of arrays. **xrtm_set_coef_l_1n()** sets the values for all derivatives for a given layer index ***i_layer*** given an (***n_derivs*** × ***n_elem*** × ***n_coef***) array of arrays of arrays. Finally, **xrtm_set_coef_l_nn()** sets the values for all layers and all derivatives given an (***n_layers*** × ***n_derivs*** × ***n_elem*** × ***n_coef***[i], $0 \leq$ i $\leq$ ***n_layers*** -1) array of arrays of arrays of arrays. In all cases ***n_elem*** $= 1$ for scalar mode and ***n_elem*** $= 6$ when option XRTM_OPTION_VECTOR has been specified and ***n_coef*** is defined for each layer by xrtm_set_coef(). To set these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_layer	index of layer to set, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
i_deriv	index of derivative to set, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$
coef_l	linearized phase matrix expansion coefficients $\partial\beta_{ij}/\partial x$ as an array of arrays (**_11**), an array of arrays of arrays (**_n1** or **_1n**), or an array of arrays of arrays of arrays (**_nn**), depending on the function called, where $-\infty \leq$ ***coef_l***[i][j], ***coef_l***[i][j][k], ***coef_l***[i][j][k][l] $\leq \infty$, except that ***coef_l***[0][0], ***coef_l***[i][0][0], ***coef_l***[i][j][0][0] $= 0$

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`double` **xrtm_get_coef_l**(*xrtm_data* ***d**, *int* ***i_layer**, *int* ***i_deriv**, *int* ***i_elem**, *int* ***i_coef**)

Description:

Get the linearized phase matrix expansion coefficient for layer index ***i_layer***, derivative index ***i_deriv***, matrix element index ***i_elem***, and coefficient index ***i_coef***. To get these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_layer	index of layer to get, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
i_deriv	index of derivative to get, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$
i_elem	index of phase matrix element to get, where $0 \leq$ ***i_elem*** $\leq$ ***n_elem*** and ***n_elem*** $=$ 0 for scalar mode and ***n_elem*** $= 6$ when option XRTM_OPTION_VECTOR has been specified
i_coef	index of phase matrix coefficient to get, where $0 \leq$ ***i_coef*** $\leq$ ***n_coef***[i] $- 1$ and $0 \leq$ i $\leq$ ***n_layers*** $- 1$

Return value:

The linearized phase matrix expansion coefficient (***i_elem***, ***i_coef***) for layer index ***i_layer*** and derivative index ***i_deriv*** or XRTM_DBL_ERROR on error.

`int` **xrtm_set_omega_1**(*xrtm_data* ***d**, *int* ***i_layer**, *double* ***omega**)

`int xrtm_set_omega_n(xrtm_data *d, double *omega)`

Description:
Set layer single scattering albedo ω. A choice of two functions are provided for this purpose. `xrtm_set_omega_1()` sets the value for a given layer index ***i_layer*** given a scalar value. `xrtm_set_omega_n()` sets the values for all layers given an array of length ***n_layers***.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_layer	index of layer to set, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
omega	single scattering albedo ω as a scalar (`_1`) or an array (`_n`), depending on the function called, where $0.0 \leq$ ***omega***, ***omega***`[i]` ≤ 1.0

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double xrtm_get_omega(xrtm_data *d, int i_layer)`

Description:
Get single scattering albedo ω for layer index ***i_layer***.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_layer	index of layer to get, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$

Return value:
The single scattering albedo ω for layer index ***i_layer*** or XRTM_DBL_ERROR on error.

```
int xrtm_set_omega_l_11(xrtm_data *d, int i_layer, int i_deriv, double omega_l)
int xrtm_set_omega_l_n1(xrtm_data *d, int i_deriv, double *omega_l)
int xrtm_set_omega_l_1n(xrtm_data *d, int i_layer, double *omega_l)
int xrtm_set_omega_l_nn(xrtm_data *d, double **omega_l)
```

Description:
Set layer linearized single scattering albedo $\partial\omega/\partial x$. A choice of four functions are provided for this purpose. `xrtm_set_omega_l_11()` sets the value for a given layer index ***i_layer*** and a given derivative index ***i_deriv*** given a scalar value. `xrtm_set_omega_l_n1()` sets the values for all layers for a given derivative index ***i_deriv*** given an array of length ***n_layers***. `xrtm_set_omega_l_1n()` sets the values for all derivatives for a given layer index ***i_layer*** given an array of length ***n_derivs***. Finally, `xrtm_set_omega_l_nn()` sets the values for all layers and all derivatives given an (***n_layers*** $\times$ ***n_derivs***) array of arrays. To set these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_layer	index of layer to set, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
i_deriv	index of derivative to set, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$
omega_l	linearized single scattering albedo $\partial\omega/\partial x$ as a scalar (`_1`), an array (`_n1` or `_1n`), or an array of arrays (`_nn`), depending on the function called, where $-\infty \leq$ ***omega_l***, ***omega_l***`[i]`, ***omega_l***`[i][j]` $\leq \infty$

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_omega_l`**`(xrtm_data *`**`d`**`, int` **`i_layer`**`, int` **`i_deriv`**`)`

Description:
Get linearized single scattering albedo $\partial\omega/\partial x$ for layer index ***i_layer*** and derivative index ***i_deriv***. To get these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_layer	index of layer to get, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
i_deriv	index of derivative to get, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$

Return value:
The linearized single scattering albedo $\partial\omega/\partial x$ for layer index ***i_layer*** and derivative index ***i_deriv*** or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_ltau_1`**`(xrtm_data *`**`d`**`, int` **`i_layer`**`, double` **`ltau`**`)`
`int` **`xrtm_set_ltau_n`**`(xrtm_data *`**`d`**`, double *`**`ltau`**`)`

Description:
Set layer optical thickness τ. A choice of two functions are provided for this purpose. **`xrtm_set_ltau_1()`** sets the value for a given layer index ***i_layer*** given a scalar value. **`xrtm_set_ltau_n()`** sets the values for all layers given an array of length ***n_layers***.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_layer	index of layer to set, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
ltau	optical thickness τ as a scalar (**_1**) or an array (**_n**), depending on the function called, where ***ltau***, ***ltau***$[i] \geq 0.0$

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_ltau`**`(xrtm_data *`**`d`**`, int` **`i_layer`**`)`

Description:
Get optical thickness τ for layer index ***i_layer***.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_layer	index of layer to get, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$

Return value:
The optical thickness τ for layer index ***i_layer*** or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_ltau_l_11`**`(xrtm_data *`**`d`**`, int` **`i_layer`**`, int` **`i_deriv`**`, double` **`ltau_l`**`)`
`int` **`xrtm_set_ltau_l_n1`**`(xrtm_data *`**`d`**`, int` **`i_deriv`**`, double *`**`ltau_l`**`)`
`int` **`xrtm_set_ltau_l_1n`**`(xrtm_data *`**`d`**`, int` **`i_layer`**`, double *`**`ltau_l`**`)`
`int` **`xrtm_set_ltau_l_nn`**`(xrtm_data *`**`d`**`, double **`**`ltau_l`**`)`

Description:

Set linearized layer optical thickness $\partial\tau/\partial x$. A choice of four functions are provided for this purpose. **xrtm_set_ltau_l_11()** sets the value for a given layer index ***i_layer*** and a given derivative index ***i_deriv*** given a scalar value. **xrtm_set_ltau_l_n1()** sets the values for all layers for a given derivative index ***i_deriv*** given an array of length ***n_layers***. **xrtm_set_ltau_l_1n()** sets the values for all derivatives for a given layer index ***i_layer*** given an array of length ***n_derivs***. Finally, **xrtm_set_ltau_l_nn()** sets the values for all layers and all derivatives given an (***n_layers*** $\times$ ***n_derivs***) array of arrays. To set these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_layer	index of layer to set, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
i_deriv	index of derivative to set, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$
ltau_l	linearized optical thickness $\partial\tau/\partial x$ as a scalar (**_11**), an array (**_n1** and **_1n**), or an array of arrays (**_nn**), depending on the function called, where $-\infty \leq$ ***ltau_l***, ***ltau_l***$[i]$, ***ltau_l***$[i][j] \leq \infty$

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`double` **xrtm_get_ltau_l**(*xrtm_data* ***d**, `int` **i_layer**, `int` **i_deriv**)

Description:

Get linearized optical thickness $\partial\tau/\partial x$ for layer index ***i_layer*** and derivative index ***i_deriv***. To get these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_layer	index of layer to get, where $0 \leq$ ***i_layer*** $\leq$ ***n_layers*** $- 1$
i_deriv	index of derivative to get, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$

Return value:

The linearized optical thickness $\partial\tau/\partial x$ for layer index ***i_layer*** and derivative index ***i_deriv*** or XRTM_DBL_ERROR on error.

`int` **xrtm_set_surface_b**(*xrtm_data* ***d**, `double` **surface_b**)

Description:

Set the surface Planck radiance. Must be in the same units as F_0 and the TOA and level Planck radiances. To set this value XRTM must have been created with the option XRTM_OPTION_SOURCE_THERMAL, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
surface_b	surface Planck radiance, where ***surface_b*** ≥ 0.0

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_surface_b`**`(`*`xrtm_data *d`*`)`

Description:
Get the surface Planck radiance. To get this value XRTM must have been created with the option XRTM_OPTION_SOURCE_THERMAL, otherwise it is an error.

Arguments:
d the *xrtm_data* structure which represents the instance created

Return value:
The surface Planck radiance or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_kernel_ampfac`**`(`*`xrtm_data *d, int i_kernel, double ampfac`*`)`

Description:
Set the amplitude factor for kernel index ***i_kernel***. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **`xrtm_create()`**. To set this value XRTM must have been created with ***n_kernels*** > 0, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_kernel	index of kernel to set, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$
ampfac	the amplitude factor for kernel ***i_kernel***, where $0.0 \leq$ ***ampfac*** ≤ 1.0

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_kernel_ampfac`**`(`*`xrtm_data *d, int i_kernel`*`)`

Description:
Get the amplitude factor for kernel index ***i_kernel***. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **`xrtm_create()`**.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_kernel	index of kernel to get, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$

Return value:
The amplitude factor for kernel index ***i_kernel*** or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_kernel_params_1`**`(`*`xrtm_data *d, int i_kernel, int i_param, double param`*`)`
`int` **`xrtm_set_kernel_params_n`**`(`*`xrtm_data *d, int i_kernel, double *params`*`)`

Description:
Set the kernel parameters for kernel index ***i_kernel***. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **`xrtm_create()`**. A choice of two functions are provided for this purpose. **`xrtm_set_kernel_params_1()`** sets the value for a given parameter index ***i_param*** given a scalar value. **`xrtm_set_kernel_params_n()`** sets the values for all parameters given an array of length ***n_params***, where ***n_params*** is the number of parameters required for kernel ***i_kernel***. The number of parameters and a description of each parameter is given for each kernel in section 3.2.3.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_kernel	index of kernel to set, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$
i_param	index of parameter to set, where $0 \leq$ ***i_param*** $\leq$ ***n_params*** $- 1$
param	kernel parameter as a scalar (**_1**) or an array (**_n**), depending on the function called, where $-\infty \leq$ ***param***, ***param***$[i] \leq \infty$

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_kernel_params`**`(`*`xrtm_data *`***`d`**`,` *`int`* ***`i_kernel`***`,` *`int`* ***`i_param`***`)`

Description:

Get the kernel parameter for kernel index ***i_kernel*** and parameter index ***i_param***. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **`xrtm_create()`**. The parameter indices are described for each kernel in section 3.2.3.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_kernel	index of kernel to get, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$
i_param	index of parameter to set, where $0 \leq$ ***i_param*** $\leq$ ***n_params*** $- 1$

Return value:

The kernel parameter for kernel index ***i_kernel*** and parameter index ***i_param*** or XRTM_DBL_ERROR on error.

`int` **`xrtm_set_kernel_ampfac_l_1`**`(`*`xrtm_data *`***`d`**`,` *`int`* ***`i_kernel`***`,` *`int`* ***`i_deriv`***`,`
`double` ***`ampfac_l`***`)`
`int` **`xrtm_set_kernel_ampfac_l_n`**`(`*`xrtm_data *`***`d`**`,` *`int`* ***`i_kernel`***`,` *`double *`***`ampfac_l`**`)`

Description:

Set the linearized amplitude factor for kernel index ***i_kernel***. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **`xrtm_create()`**. A choice of two functions are provided for this purpose. **`xrtm_set_kernel_ampfac_l_1()`** sets the value for a given derivative index ***i_deriv*** given a scalar value. **`xrtm_set_kernel_ampfac_l_n()`** sets the values for all derivatives given an array of length ***n_derivs***. To set this value XRTM must have been created with ***n_kernels*** > 0, otherwise it is an error. To set these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
i_kernel	index of kernel to set, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$
i_deriv	index of derivative to set, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$
ampfac_l	the linearized amplitude factor for kernel ***i_kernel***, where $-\infty \leq$ ***ampfac***, ***ampfac***`[i]` $\leq \infty$

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

`double` **`xrtm_get_kernel_ampfac_l`**`(`*`xrtm_data *`***`d`**`,` *`int`* ***`i_kernel`***`,` *`int`* ***`i_deriv`***`)`

Description:

Get the linearized amplitude factor for kernel index ***i_kernel*** and derivative index ***i_deriv*** . The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **xrtm_create()**. To get these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_kernel	index of kernel to get, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$
i_deriv	index of derivative to get, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$

Return value:

The linearized amplitude factor for kernel index ***i_kernel*** or XRTM_DBL_ERROR on error.

```
int xrtm_set_kernel_params_l_11(xrtm_data *d, int i_kernel, int i_deriv,
    int i_param, double param_l)
int xrtm_set_kernel_params_l_1n(xrtm_data *d, int i_kernel, int i_deriv,
    double *params_l)
int xrtm_set_kernel_params_l_n1(xrtm_data *d, int i_kernel, int i_param,
    double *params_l)
int xrtm_set_kernel_params_l_nn(xrtm_data *d, int i_kernel, double **params_l)
```

Description:

Set the linearized kernel parameters for kernel index ***i_kernel***. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **xrtm_create()**. A choice of four functions are provided for this purpose. **xrtm_set_kernel_params_l_11()** sets the value for a given derivative index ***i_deriv*** and a given parameter index ***i_param*** given a scalar value. **xrtm_set_kernel_params_l_1n()** sets the value for all parameters given a derivative index ***i_deriv*** given an array of length ***n_params***. **xrtm_set_kernel_params_l_n1()** sets the value for all derivatives given a parameter index ***i_param*** given an array of length ***n_derivs***. **xrtm_set_kernel_params_l_nn()** sets the value for all derivatives given and all parameters given a (***n_derivs*** $\times$ ***n_params***) array of arrays. ***n_params*** is the number of parameters required for kernel ***i_kernel***. The number of parameters and a description of each parameter is given for each kernel in section 3.2.3. To set these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_kernel	index of kernel to set, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$
i_deriv	index of derivative to set, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$
i_param	index of parameter to set, where $0 \leq$ ***i_param*** $\leq$ ***n_params*** $- 1$
param_l	the linearized kernel parameter for kernel ***i_kernel***, where $-\infty \leq$ ***param_l***, ***param_l***[i], ***param_l***[i][j] $\leq \infty$

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

```
double xrtm_get_kernel_params_l(xrtm_data *d, int i_kernel, int i_deriv,
    int i_param)
```

Description:

Get the linearized kernel parameter for kernel index ***i_kernel***, derivative index ***i_deriv***, and parameter index ***i_param***. The kernel index is the index at which the kernel was given in the array ***kernels*** given as input to **xrtm_create()**. The parameter indices are described for each kernel in section 3.2.3. To get these values XRTM must have been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created
i_kernel	index of kernel to get, where $0 \leq$ ***i_kernel*** $\leq$ ***n_kernels*** $- 1$
i_deriv	index of derivative to get, where $0 \leq$ ***i_deriv*** $\leq$ ***n_derivs*** $- 1$
i_param	index of parameter to set, where $0 \leq$ ***i_param*** $\leq$ ***n_params*** $- 1$

Return value:

The linearized kernel parameter for kernel index ***i_kernel*** and parameter index ***i_param*** or XRTM_DBL_ERROR on error.

3.5 Running the model and getting output

```
int xrtm_update_varied_layers(xrtm_data *d)
```

Description:

This function must be called every time the set of layers that vary, including the surface, changes. A layer "varies" if at least one of the linearized values associated with that layer is nonzero. For example, if at least one of the linearized values for a particular layer has been set to a nonzero value where before all the values for that layer were zero then that layer has been added to the set of layers that varies and **xrtm_update_varied_layers()** must be called. Equivalently, If all the linearized values for a layer are set to zero where before at least one of the values was nonzero then that layer has been removed from the set of layers that vary so **xrtm_update_varied_layers()** must be called. On the other hand, if the values of nonzero linearized values are only changed to other nonzero values then **xrtm_update_varied_layers()** does not need to be called. For efficiency, it is a good idea to call the function once, after all changes to all layers and all values have been changed just before running the model. Calling this function is only valid when XRTM has been created with the option XRTM_CALC_DERIVS, otherwise it is an error.

Arguments:

d	the *xrtm_data* structure which represents the instance created

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

```
int xrtm_solution(xrtm_data *d, enum xrtm_solver_mask solver, int solutions,
    int n_out_phis, double **out_phis, double ****I_p, double ****I_m,
    double *****I_p_l, double *****I_m_l, double *mean_p, double *mean_m,
    double **mean_p_l, double **mean_m_l, double *flux_p, double *flux_m,
    double **flux_p_l, double **flux_m_l, double *flux_div, double **flux_div_l)
```

Description:

Run XRTM using a specified solver and return various result types.

Arguments:

d	the *xrtm_data* structure which represents the instance created
solver	bit mask for the `solver` to be used
solutions	bit mask for the `solutions` to return results for
n_out_phis	number azimuth angles ϕ to return results for
out_phis	array of arrays of output azimuth angles with dimensions (***n_out_thetas***, ***n_out_phis***), where $0.0 \leq$ ***out_phis***`[i][j]` ≤ 360.0
I_p	(output) array of arrays of arrays of arrays of upward radiances with dimensions (***n_out_levels***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)
I_m	(output) array of arrays of arrays of arrays of downward radiances with dimensions (***n_out_levels***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)
I_p_l	(output) array of arrays of arrays of arrays of arrays of upward radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)
I_m_l	(output) array of arrays of arrays of arrays of arrays of downward radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)
mean_p	(output) array of arrays of upward mean radiances with dimensions (***n_out_levels***, ***n_stokes***)
mean_m	(output) array of arrays of downward mean radiances with dimensions (***n_out_levels***, ***n_stokes***)
mean_p_l	(output) array of arrays of arrays of upward mean radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)
mean_m_l	(output) array of arrays of arrays of downward mean radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)
flux_p	(output) array of arrays of upward fluxes with dimensions (***n_out_levels***, ***n_stokes***)
flux_m	(output) array of arrays of downward fluxes with dimensions (***n_out_levels***, ***n_stokes***)
flux_p_l	(output) array of arrays of arrays of upward flux derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)
flux_m_l	(output) array of arrays of arrays of downward flux derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)
flux_div	(output) array of arrays of flux divergence with dimensions (***n_out_levels***, ***n_stokes***)
flux_div_l	(output) array of arrays of arrays of flux divergence derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

```
int xrtm_radiance(xrtm_data *d, enum xrtm_solver_mask solver, int n_out_phis,
    double **out_phis, double ****I_p, double ****I_m, double *****I_p_l,
    double *****I_m_l)
```

Description:

Run XRTM using a specified solver and return radiance results.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
solver	bit mask for the `solver` to be used
n_out_phis	number azimuth angles ϕ to return results for
out_phis	array of arrays of output azimuth angles with dimensions (***n_out_thetas***, ***n_out_phis***), where $0.0 \leq$ ***out_phis***`[i][j]` ≤ 360.0
I_p	(output) array of arrays of arrays of arrays of upward radiances with dimensions (***n_out_levels***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)
I_m	(output) array of arrays of arrays of arrays of downward radiances with dimensions (***n_out_levels***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)
I_p_l	(output) array of arrays of arrays of arrays of arrays of upward radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)
I_m_l	(output) array of arrays of arrays of arrays of arrays of downward radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_out_thetas***, ***n_out_phis***, ***n_stokes***)

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

```
int xrtm_mean_radiance(xrtm_data *d, enum xrtm_solver_mask solver, double *mean_p,
    double *mean_m, double **mean_p_l, double **mean_m_l)
```

Description:

Run XRTM using a specified solver and return mean radiance results.

Arguments:

d	the ***xrtm_data*** structure which represents the instance created
solver	bit mask for the `solver` to be used
mean_p	(output) array of arrays of upward mean radiances with dimensions (***n_out_levels***, ***n_stokes***)
mean_m	(output) array of arrays of downward mean radiances with dimensions (***n_out_levels***, ***n_stokes***)
mean_p_l	(output) array of arrays of arrays of upward mean radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)
mean_m_l	(output) array of arrays of arrays of downward mean radiance derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)

Return value:

Zero with successful completion or XRTM_INT_ERROR on error.

```
int xrtm_flux(xrtm_data *d, enum xrtm_solver_mask solver, double *flux_p,
    double *flux_m, double **flux_p_l, double **flux_m_l)
```

Description:

Run XRTM using a specified solver and return flux results.

Arguments:

d	the *xrtm_data* structure which represents the instance created
solver	bit mask for the solver to be used
flux_p	(output) array of arrays of upward fluxes with dimensions (***n_out_levels***, ***n_stokes***)
flux_m	(output) array of arrays of downward fluxes with dimensions (***n_out_levels***, ***n_stokes***)
flux_p_l	(output) array of arrays of arrays of upward flux derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)
flux_m_l	(output) array of arrays of arrays of downward flux derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

`int` **`xrtm_flux_divergence`**`(`*`xrtm_data *`***`d`**`, enum xrtm_solver_mask `***`solver`**`,`
*`double *`***`flux_div`**`, double **`***`flux_div_l`**`)`

Description:
Run XRTM using a specified solver and return flux divergence results.

Arguments:

d	the *xrtm_data* structure which represents the instance created
solver	bit mask for the solver to be used
flux_div	(output) array of arrays of flux divergence with dimensions (***n_out_levels***, ***n_stokes***)
flux_div_l	(output) array of arrays of arrays of flux divergence derivatives with dimensions (***n_out_levels***, ***n_derivs***, ***n_stokes***)

Return value:
Zero with successful completion or XRTM_INT_ERROR on error.

3.6 Miscellaneous functions

`const char *`**`xrtm_get_version`**`()`

Description:
Get the XRTM version sstring.

Arguments:
None.

Return value:
Pointer to the XRTM version string.

3.7 Error Handling

All functions in the XRTM C interface return either an `int` or a `double`. If an error occurs when calling these functions then the return value will be set to one of the follwing error constants depending on the function's return type:

XRTM_INT_ERROR
: Returned from functions returning an `int` when an error has occurred while calling the function.

XRTM_DBL_ERROR
: Returned from functions returning a `double` when an error has occurred while calling the function.

When an error occurs XRTM will print an error message followed by a function call stack to the standard error (`stderr`) stream.

3.8 Example C program using XRTM

An example program using the C interface is at

```
examples/example_c.c
```

and when the XRTM code is compiled properly the C example program should be compiled as

```
examples/example_c
```

4 XRTM Utilities

4.1 Test suite: testxrtm

4.1.1 Testxrtm options

4.2 Stand alone execution: callxrtm

`callxrtm` is a command line program that takes inputs, either from the command line or from an input file, runs the model, and outputs results. `callxrtm` was created primarily as a development and testing tool. Although it is not recommended for production use (it is recommended that the API be used instead) it can also be useful for running XRTM for small studies.

The path to `callxrtm` is

```
utils/callxrtm
```

or when using Visual Studio

```
$(SolutionDir)/$(ConfigurationName)/callxrtm.exe
```

where the variable `$(SolutionDir)` is `msvs_2005`, `msvs_2008`, or `msvs_2010` and the variable `$(ConfigurationName)` is `Debug` or `Release`.

An example run of `callxrtm` is contained in the following test text file:

```
examples/callxrtm.txt
```

Two cases are given. One that runs `callxrtm` using the command line to set all the inputs and another that uses the following XRTM input file (`.xif`) to set the inputs

```
examples/example.xif
```

Please see text file for more details about the example run. The `callxrtm` input format for the command line and an input file is outlined in the next section.

4.2.1 Callxrtm options

4.2.2 Callxrtm input format

Bibliography

Raphael Aronson. General solution for polarized radiation in a homogeneous-slab atmosphere. *The Astrophysical Journal*, 177:411–421, October 1972.

Charles Cox and Walter Munk. Measurement of the roughness of the sea surface from photographs of the sun's glitter. *Journal of the Optical Society of America*, 44(11):838–850, November 1954.

Arne Dahlback and Knut Stamnes. A new spherical model for computing the radiation field available for photolysis and heating at twilight. *Planetary and Space Science*, 39(5):671–683, 1991.

J. F. de Haan, P. B. Bosma, and J. W. Hovenier. The adding method for multiple scattering calculations of polarized light. *Astronomy and Astrophysics*, 183:371–391, 1987.

A. Doicu and T. Trautmann. Discrete-ordinate method with matrix exponential for a pseudo-spherical atmosphere: Scalar case. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 110:146–158, 2009a.

A Doicu and T Trautmann. Discrete-ordinate method with matrix exponential for a pseudo-spherical atmosphere: Vector case. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 110:159–172, 2009b.

A. L. Fymat and Sueo Ueno. Order-of-scattering of partially polarized radiation in inhomogeneous anisotropically scattering atmospheres I: Fundamentals. *Astrophysics and Space Science*, 30: 3–25, 1974.

I. P. Grant and G. E. Hunt. Discrete space theory of radiative transfer I. Fundamentals. *Proceedings of the Royal Society. Series A: Mathematical and Physical Sciences*, 313:183–197, 1969.

Bruce Hapke. Bidirectional reflectance spectroscopy 1. theory. *Journal of Geophysical Research*, 86 (B4):3039–3054, April 1981.

Bruce Hapke and Eddie Wells. Bidirectional reflectance spectroscopy 2. experiments and observations. *Journal of Geophysical Research*, 86(B4):3055–3060, April 1981.

Kiyoshi Kawabata and Sueo Ueno. The first three orders of scattering in vertically inhomogeneous scattering-absorbing media. *Astrophysics and Space Science*, 150:327–344, 1988.

J. Lenoble, M. Herman, J. L. Deuzé, B. Lafrance, R. Santer, and D. Tanré. A successive order of scattering code for solving the vector equation of transfer in the earth's atmosphere with aerosols. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 107:479–507, 2007.

K. N. Liou. *An Introduction to Atmospheric Radiation*. Academic Press, 525 B Street, Suite 1900, San Diego, California 92101-4495, USA, second edition, 2002.

Kuo-Nan Liou. A numerical experiment on Chandrasekhar's discrete-ordinate method for radiative transfer: Applications to cloudy and hazy atmospheres. *Journal of the Atmospheric Sciences*, 30:1303–1326, October 1973.

Greg McGarragh and Philip Gabriel. Efficient computation of radiances and for optically thin media by Padé approximants. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 111: 1885–1889, 2010.

Qilong Min and Minzheng Duan. Successive order of scattering model for solving vector radiative transfer in the atmosphere. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 87: 243–259, 2004.

T. Nakajima and M. Tanaka. Matrix formulation for the transfer of solar radiation in a plane-parallel scattering atmosphere. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 35 (1):13–21, 1986.

T. Nakajima and M. Tanaka. Algorithms for radiative intensity calculations in moderately thick atmospheres using a truncation approximation. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 40(1):51–69, 1988.

Vijay Natraj and Robert J. D. Spurr. A fast linearized pseudo-spherical two orders of scattering model to account for polarization in vertically inhomogeneous scattering-absorbing media. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 107:263–293, 2007.

Hafizur Rahman, Bernard Pinty, and Michel M. Verstraete. Coupled surface-atmosphere reflectance (csar) model 2. semiempirical surface model usable with noaa advanced very high resolution radiometer data. *Journal of Geophysical Research*, 98(D11):20791–20801, November 1993.

Jean-Louis Roujean, Marc Leroy, and Pierre-Yves Deschamps. A bidirectional reflectance model of the earth's surface for the correction of remote sensing data. *Journal of Geophysical Research*, 97(D18):20455–20468, December 1992.

C. E. Siewert. On the phase matrix basic to scattering of polarized light. *Astronomy and Astrophysics*, 109:195–200, 1982.

C. E. Siewert. A concise and accurate solution to Chandrasekher's basic problem in radiative transfer. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 64:109–130, 2000.

R. J. D. Spurr and M. J. Christi. Linearization of the interaction principle: Analytic jacobians in the "Radiant" model. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 103:431–446, 2007.

R. J. D. Spurr, T. P. Kurosu, and K. V. Chance. A linearized discrete ordinate radiative transfer model for atmospheric remote-sensing retrieval. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 68:689–735, 2001.

Robert J. D. Spurr. A new approach to the retrieval of surface properties from earthshine measurements. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 83:15–46, 2004.

Knut Stamnes, Si-Chee Tsay, Warren Wiscombe, and Kolf Jayaweera. Numerically stable algorithm for discrete-ordinate-method radiative transfer in multiple scattering and emitting layered media. *Applied Optics*, 27(12):2502–2509, June 1988.

Alexander G. Voronovich, Albin J. Gasiewski, and Bob L. Weber. A fast multistream scattering-based jacobian for microwave radiance assimilation. *IEEE Transactions on Geoscience and Remote Sensing*, 42(8):1749–1761, August 2004.

W. Wanner, X. Li, and A. H. Strahler. On the derivation of kernels for kernel-driven models of bidirectional reflectance. *Journal of Geophysical Research*, 100(D10):21077–21089, October 1995.

W. J. Wiscombe. The delta-m method: Rapid yet accurate radiative flux calculations for strongly asymmetric phase functions. *Journal of the Atmospheric Sciences*, 34:1408–1422, September 1977.